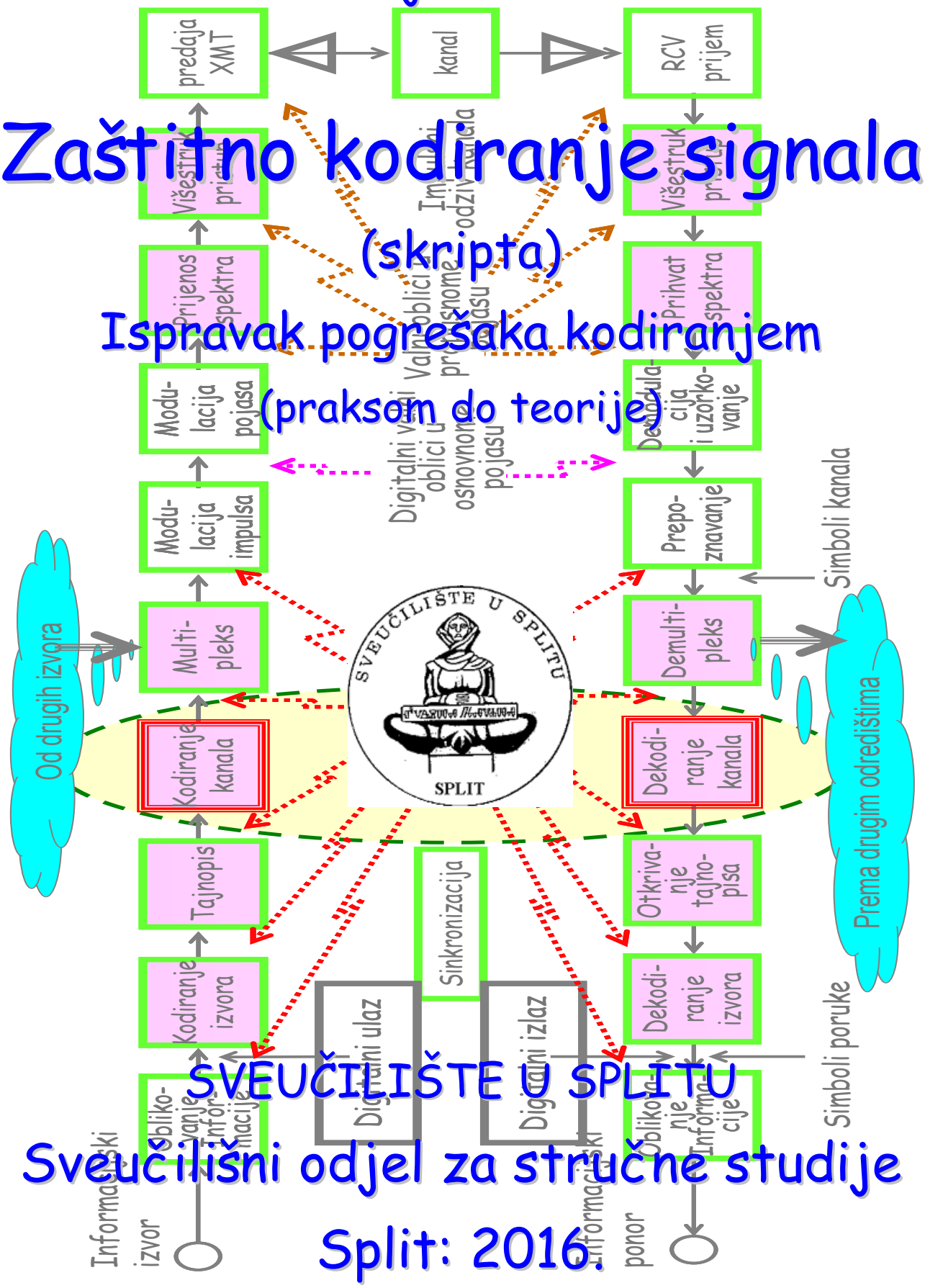
Marijo Nižetić

# Zaštitno kodiranje signala

(skripta)

## Ispravak pogrešaka kodiranjem

(praksom do teorije)




SVEUČILIŠTE U SPLITU

Sveučilišni odjel za stručne studije

Split: 2016.

# *Syllabus:*

| Naziv područja (osnovna poveznica): | Ostale poveznice na mjesta obrade: |
|---|---|
| Uvod | 0. Uvod |
| Osnovni modeli komunikacijskoga kanala | 0.3. Komunikacijski kanal |
| Sadržaj i mjera informacije | 1.2. Izvor, odredište, sadržaj informacije i mjera informacije |
| Izvor i odredište informacije | 1.3. Entropija |
| Komunikacijski kanal sa šumom | 1.3.1. Informacijska mjera |
| Tehnike upravljanja pogreškama (FEC i ARQ) | 0.7. Strategije upravljanja pogreškama |
| Zaštitno kodiranje informacije | 1. Temelji zaštitnoga kodiranja |
| Ravnomjeran kod, neravnomjeran kod, optimalni kodovi, predikcijsko kodiranje | 2.5.3.5. Optimalno kodiranje i primjena Hammingova koda |
| Koder i dekoder signala | 1.3.2. Zaključci o entropiji |
| Pseudo slučajni nizovi i njihove osobine, generatori pseudo slučajnih nizova | 4.5.3. Pseudo-slučajni nizovi |
| Linearni kodovi za kontrolu pogrešaka | 2. Linearni kodovi |
| Konvolucijski kodovi | 6. Konvolucijski kodovi |
| Ciklički kodovi | 4. Ciklički kodovi |
| BCH kodovi i pridruženi algoritmi | 5.13.1. BCH kodovi |
| Turbo kodovi[1] | 7. Turbo kodovi |
| Metode i postupci za preplitanje signala | 7.3.2.1. Sklopovi za preplitanje turbo kodova |
| Vremensko raspršenje pogreške u prijenosu | 6.7.3. Dekodiranje mekom odlukom |
| Trellis kodovi | 8. Trellis kodovi |
| Preplitanje u mobilnim komunikacijama (GSM) | 7.3. Turbo kodovi |
| Preplitanje signala u DVB sustavima | 7.3.1. Serijski ulančani kodovi |
| Vremensko kašnjenje zbog obrade | 6.6.1. Viterbijev algoritam |

[1] Syllabus nije predvidio temu: "Turbo kodovi"

# Popis laboratorijskih vježbi



# Popis primjera

# Popis slika i pripadnih simulacijskih krugova

| Naziv | Simulacijski krug pridružen slici | Naziv | Simulacijski krug pridružen slici |
|---|---|---|---|
| 0. Poglavlje 0 - Uvod 1 | | | |
| Slika 0.1 | Jednosmjeran komunikacisjki kanal | Slika 0.2 | SHANNON00.PPT (slide 3) |
| Slika 0.3 | SHANNON00.PPT (slide 17) | Slika 0.4 | SHANNON00.PPT (slide 18) |
| 1. Poglavlje 1 - Temelji zaštitnoga kodiranja 9 | | | |
| Slika 1.1 | SHANNON00.PPT (slide 1) | Slika 1.2 | SHANNON00.PPT (slide 2) |
| Slika 1.3 | SHANNON00.PPT (slide 4) | Slika 1.4 | SHANNON00.PPT (slide 5) |
| Slika 1.5 | SHANNON00.PPT (slide 6) | Slika 1.6 | SHANNON00.PPT (slide 7) |
| Slika 1.7 | SHANNON00.PPT (slide 8) | Slika 1.8 | A Commonsense approach.ppt (slide 83) |
| Slika 1.9 | SHANNON00.PPT (slide 9) | Slika 1.10 | SHANNON00.PPT (slide 12) |
| Slika 1.11 | SHANNON00.PPT (slide 11) | Slika 1.12 | SHANNON00.PPT (slide 11) |
| Slika 1.13 | Teorija informacija 1.ppt (slide 1) | Slika 1.14 | A Commonsense approach.ppt (slides 104-105) |
| Slika 1.15 | ---------- | Slika 1.16 | A Commonsense approach.ppt (slide 86) |
| Slika 1.17 | A Commonsense approach.ppt (slide 87) | Slika 1.18 | A Commonsense approach.ppt (slide 88) |
| Slika 1.19 | A Commonsense approach.ppt (slide 89) | Slika 1.19 | A Commonsense approach.ppt (slide 90) |
| Slika 1.20 | A Commonsense approach.ppt (slide 94) | Slika 1.21 | A Commonsense approach.ppt (slide 107) |
| Slika 1.22 | A Commonsense approach.ppt (slides 64-68) | Slika 1.24 | ---------- |
| Slika 1.23 | A Commonsense approach.ppt (slide 106) | Slika 1.25 | XOR vrata.circ |
| Slika 1.26 | XOR vrata.circ | Slika 1.27 | a XOR b.circ |
| Slika 1.28 | Slika 1.2.circ | Slika 1.29 | Slika 1.3.circ |
| Slika 1.30 | Slika3.7 b.circ | Slika 1.31 | All about Digital Modulation_.ppt (slide 28) |
| 2. Poglavlje 2 - Linearni kodovi 44 | | | |
| Slika 2.1 | Zbroji 2. riječi od 4 bita.circ | Slika 2.2 | Slika 2.1.circ |
| Slika 2.3 | Slika 2.1_1.circ | Slika 2.4 | Dijagram toka Huffmanovoga algoritma.doc |
| Slika 2.5 | All about Digital Modulation_.ppt (slides 31-32) | Slika 2.6 | SHANNON00.PPT (slide 20) |
| Slika 2.7 | Slika 2.2.circ | Slika 2.8 | Slika 2.2.circ |
| Slika 2.9 | Slika Binarno brojilo.circ | Slika 2.10 | Slika matrica H.circ |
| Slika 2.11 | Slika Count Down.circ | Slika 2.12a | Slika_Paritetna matrica H.circ |
| Slika 2.12b | Slika Count Down.circ | Slika 2.13 | Slika 3.1.circ |
| Slika 2.14 | Hammingov sustavan (7, 4) kod.circ | Slika 2.15a | Slika Count Down.circ |
| Slika 2.15b | Slika matrica H.circ | Slika 2.16 | Hammingov sustavan (7, 4) kod.circ |
| Slika 2.17 | Hammingov sustavan (7, 4) kod.circ | Slika 2.18 | Koder 9,5 D.circ |
| Slika 2.19 | Koder 9,5 D.circ | | |
| 3. Poglavlje 3 - Osnovni strujni krugovi 82 | | | |
| Slika 3.1 | Slika 3.1.circ | Slika 3.2 | Hammingov (7, 4) kod.circ |
| Slika 3.3 | Slika 3.1.circ | Slika 3.4 | Automatsko stvaranje redaka matrice H'.circ |
| Slika 3.4a | Slika 3.4a.circ | Slika 3.5 | Automatsko stvaranje redaka matrice H'.circ |
| Slika 3.6 | Slika 3.2.1.circ | Slika 3.7 | Slika3.2.circ |
| Slika 3.8 | Množenje vektora matricom.circ | Slika 3.9 | Množenje vektora matricom.circ |
| Slika 3.10 | Slika 3.2.1.circ | Slika 3.11 | Slika 3.2.2.circ |
| Slika 3.12 | Slika3.3__.circ | Slika 3.13 | Slika3.3.circ |
| Slika 3.14 | Slika3.3__.circ | Slika 3.15 | Slika3.3__.circ |
| Slika 3.16 | Slika3.3__n=4.circ | Slika 3.17 | Slika3.3__n=i.circ |
| Slika 3.18 | Slika3.3__.circ | Slika 3.19 | Slika3.2.circ |
| Slika 3.20a | Slika3.3__n=4.circ | Slika 3.20b | Slika3.2_7.circ |
| Slika 3.21 | Slika3.3__.circ | Slika 3.22 | Posmik zbrajajući a i h.circ |
| Slika 3.23 | Slika3.2_26.circ | Slika 3.24 | Zbroj 26 bitova.circ |
| Slika 3.25 | Slika 3.4.circ | Slika 3.26 | A Commonsense approach.ppt (slide 30) |
| Slika 3.27 | Slika 3.4a.circ | Slika 3.28a | Slika3.1.circ |
| Slika 3.28b | Slika3.5b.circ | Slika 3.29a | Slika3.6 a i b.circ |
| Slika 3.29b | Slika3.6 a i b.circ | Slika 3.30 | Slika3.7 b.circ |
| Slika 3.31 | Slika3.1.circ | Slika 3.32a | Automatsko stvaranje redaka matricce H'.circ |
| Slika 3.32b | LFSR za generiranje redaka HT matrice (6, 3) koda.circ | Slika 3.33 | Automatsko stvaranje redaka matrice H'.circ |
| 4. Poglavlje 4 - Ciklički kodovi 103 | | | |

| Naziv | Simulacijski krug pridružen slici | Naziv | Simulacijski krug pridružen slici |
|---|---|---|---|
| Slika 4.1 | Slika4.5.circ | Slika 4.2 | Slika4.1.circ |
| Slika 4.3 | ---------- | Slika 4.4 | Slika 2.1.circ |
| Slika 4.5 | Slika4.2.circ | Slika 4.6 | Automatsko stvaranje redaka matrice H'.circ |
| Slika 4.7 | Slika4.2.circ | Slika 4.8 | Slika4.2.circ |
| Slika 4.9 | Slika4.2.circ | Slika 4.10 | Slika4.2.circ |
| Slika 4.11a | Slika4.3.circ | Slika 4.11b | Slika4.3.circ |
| Slika 4.11c | Slika4.3.circ | Slika 4.12 | Slika4.4.circ |
| Slika 4.13a | Automatsko stvaranje redaka matrice H'.circ | Slika 4.13b | Slika4.1.circ |
| Slika 4.13 | Slika4.1.circ | Slika 4.14 | Slika4.1.circ |
| Slika 4.15a | Slika4.1ponovljena.circ | Slika 4.15b | Slika4.1ponovljena.circ |
| Slika 4.16 | Slika 4.2.circ | Slika 4.17a | Slika 4.17.circ |
| Slika 4.17b | Slika 4.17.circ | Slika 4.18a | LFSR s unutarnjim povratnom vezom.circ |
| Slika 4.18b | LFSR s vanjskom povratnom vezom.circ | Slika 4.19 | LFSR 110101111000100.circ |
| Slika 4.20a | Slika 4.22.circ | Slika 4.20b | Slika 4.22.circ |
| Slika 4.21 | A Commonsense approach.ppt (slide 11) | Slika 4.22 | Slika 4.21.circ |
| Slika 4.23 | Slika4.6.circ | Slika 4.24 | Slika4.6.circ |
| Slika 4.25 | Slika4.6.circ | Slika 4.26a | Slika4.7.circ |
| Slika 4.26b | Slika4.7.circ | | |
| 5. Poglavlje 5 - Praskovite pogreške 131 | | | |
| Slika 5.1 | ---------- | Slika 5.2a | Slika5.1 Koder.circ |
| Slika 5.2b | Slika5.1 Dekoder.circ | Slika 5.3 | ---------- |
| Slika 5.4 | ---------- | Slika 5.5 | A Commonsense approach.ppt (slide 2) |
| Slika 5.6a | Ispravak jednostruke praskovite pogreške.circ | Slika 5.6b | Ispravak jednostruke praskovite pogreške.circ |
| Slika 5.7 | Slika5.4.circ | Slika 5.8 | Slika5.5.circ |
| Slika 5.9 | A Commonsense approach.ppt (slide 3) | Slika 5.10 | Slika3.2.circ |
| Slika 5.11a | Slika5.8.a.circ | Slika 5.11b | Slika 5.8.b.circ |
| Slika 5.11c | Slika 5.8.c.circ | Slika 5.11d | ---------- |
| Slika 5.12 | A Commonsense approach.ppt (slide 3) | Slika 5.13 | Slika5.9.circ |
| Slika 5.14 | A Commonsense approach.ppt (slide 3) | Slika 5.15 | Slika5.11.circ |
| Slika 5.16a | L.J. Weng, Usporedbe svojstava BCH kodova za meko i tvrdo dekodiranje, 1979, Sl.3 | Slika 5.16b | Glover-Grant, Error Control Coding, Figure 10.28 |
| Slika 5.17 | A Commonsense approach.ppt (slide 5) | Slika 5.18 | A Commonsense approach.ppt (slide 6) |
| Slika 5.19 | A Commonsense approach.ppt (slide 7) | Slika 5.20 | A Commonsense approach.ppt (slide 8) |
| 6. Poglavlje 6 - Konvolucijski kodovi 163 | | | |
| Vj14 | Sklop za kašnjenje1.circ | Vj14 | Sklop za kašnjenje.circ |
| Vj14 | Slika6.4 Koder.circ | Vj14 | Slika6.4 VRLO DOBAR.circ |
| Vj14 | Slika6.4 SKORO odličan.circ | Vj14 | |
| Slika 6.1 | ---------- | Slika 6.2 | A Commonsense approach.ppt (slide 114) |
| Slika 6.3 | A Commonsense approach.ppt (slide 115) | Slika 6.4 | All about Digital Modulation_.ppt (slide 29) |
| No name | Coding the (2, 1, 4) code.pps | | |
| Slika 6.5 | 120 slika 1.circ, Konvolucijski koder.circ | Slika 6.6 | 120 slika 1 EX2.circ |
| Slika 6.7 | All about Digital Modulation.ppt (slide 5), 120 slika 2.circ | Slika 6.8 | A Commonsense approach.ppt (slides 33-34) |
| Slika 6.9 | All about Digital Modulation.ppt (slide 1) | Slika 6.10 | 120 slika 3.circ |
| Slika 6.11 | Kodiranje i dekodiranje konvolucijskim kodovima.ppt (slide 16) | Slika 6.12 | Kodiranje i dekodiranje konvolucijskim kodovima.ppt (slide 17) |
| Slika 6.13 | All about Digital Modulation.ppt (slides 13-16) | Slika 6.14 | ---------- |
| Slika 6.15 | All about Digital Modulation.ppt (slides 17-25) | Slika 6.16 | Dijagram stanja.pps |
| Slika 6.17 | Dijagram stabla za kôd (2, 1, 4).doc | Slika 6.18 | VD0.ppt (slide 4) |
| Slika 6.19 | All about Digital Modulation.ppt (slide 5) | Slika 6.20 | VD8.ppt (slide 3) |
| Slika 6.21 | (2, 1, 4) kod.circ | Slika 6.22a | Trellis diagrams.ppt (slide 9) |
| Slika 6.22b | Trellis diagrams.ppt (slide 10) | Slika 6.22c | Trellis diagrams.ppt (slide 11) |
| Slika 6.22d | Trellis diagrams.ppt (slide 12) | Slika 6.22e | Trellis diagrams.ppt (slide 13) |
| Slika 6.22f | Trellis diagrams.ppt (slide 14) | Slika 6.22g | Trellis diagrams.ppt (slide 15) |
| Slika 6.22h | Trellis diagrams.ppt (slide 16) | Slika 6.23 | Euklidove udaljenosti.ppt (slide 17) |
| Slika 6.24 | Euklidove udaljenosti.ppt (slide 16) | Slika 6.25 | Kodiranje i dekodiranje konvolucijskim kodovima.ppt (slide 18) |
| Slika 6.26 | Kodiranje i dekodiranje konvolucijskim kodovima.ppt (slide 18-19), DCFaA2E.ppt (slide ) | Slika 6.27 | Kodiranje i dekodiranje konvolucijskim kodovima.ppt (slide 20) |
| Slika 6.28 | ACATTTOECC.ppt (slide 4) | Slika 6.29 | All about Digital Modulation_.ppt (slide 38) |
| Slika 6.30 | All about Digital Modulation_.ppt (slide 36) | Slika 6.31 | All about Digital Modulation_.ppt (slide 40) |
| Slika 6.32a | All about Digital Modulation_.ppt (slide 35) | Slika 6.32b | A Commonsense approach.ppt (slide 113) |
| Slika 6.33 | All about Digital Modulation_.ppt (slide 34) | | |
| 7. Poglavlje 7 - Turbo kodovi 195 | | | |
| Slika 7.1 | A Commonsense approach.ppt (slide 115) | Slika 7.2 | A Commonsense approach.ppt (slide 37-38) |
| Slika 7.3 | A Commonsense approach.ppt (slide 36) | Slika 7.4 | ---------- |
| Slika 7.5 | A Commonsense approach.ppt (slide 39) | Slika 7.6 | A Commonsense approach.ppt (slide 43) |
| Slika 7.7 | A Commonsense approach.ppt (slide 41) | Slika 7.8 | A Commonsense approach.ppt (slide 44) |
| Slika 7.9 | A Commonsense approach.ppt (slide 45) | Slika 7.10 | A Commonsense approach.ppt (slide 47) |
| Slika 7.11 | A Commonsense approach.ppt (slide 48) | Slika 7.12 | A Commonsense approach.ppt (slide 46) |
| Slika 7.13 | A Commonsense approach.ppt (slide 55) | Slika 7.14 | A Commonsense approach.ppt (slide 60) |
| Slika 7.15 | A Commonsense approach.ppt (slide 59) | Slika 7.16 | A Commonsense approach.ppt (slide 57) |
| Slika 7.17 | A Commonsense approach.ppt (slide 53) | Slika 7.18 | A Commonsense approach.ppt (slide 61) |
| Slika 7.19 | A Commonsense approach.ppt (slide 62) | Slika 7.20 | A Commonsense approach.ppt (slide 63) |
| 8. Poglavlje 8 - Trellis kodovi 207 | | | |
| Slika 8.1 | A Commonsense approach.ppt (slide 80) | Slika 8.2 | A Commonsense approach.ppt (slides 81-82) |
| Slika 8.3a | A Commonsense approach.ppt (slide 77) | Slika 8.3b | A Commonsense approach.ppt (slide 78) |
| Slika 8.3c | A Commonsense approach.ppt (slide 79) | Slika 8.4 | ACATTTOECC.ppt (slide 7-8) |
| Slika 8.5 | ACATTTOECC.ppt (slide 9-10) | Slika 8.6 | ACATTTOECC.ppt (slide 11) |

| Naziv | Simulacijski krug pridružen slici | Naziv | Simulacijski krug pridružen slici |
|---|---|---|---|
| Slika 8.7 | ACATTTOECC.ppt (slides 15-16-17) | Slika 8.8 | ACATTTOECC.ppt (slide 12) |
| Slika 8.9 | ACATTTOECC.ppt (slide 13) | Slika 8.10 | ACATTTOECC.ppt (slide 14) |
| Slika 8.11 | A Commonsense approach.ppt (slide 117) | Slika 8.12 | Euklidove udaljenosti.ppt (slides 8-9-10) |
| Slika 8.13 | ---------- | Slika 8.14 | ---------- |
| Slika 8.15 | Stablo.ppt | Slika 8.16 | A Commonsense approach.ppt (slide 71) |
| Slika 8.17 | A Commonsense approach.ppt (slide 109) | Slika 8.18 | A Commonsense approach.ppt (slide 72) |
| Slika 8.19 | ---------- | | |
| 9. Poglavlje 9 - Praktično motrište linearne algebre 219 | | | |
| Slika 9.1 | A Commonsense approach.ppt (slide 20), Slika 9.1.circ | Slika 9.2 | SlikaA.2_.circ |
| Slika 9.3 | A Commonsense approach.ppt (slides 22-24) | Slika 9.4 | Slika3.2.circ |
| Slika 9.5 | Slika3.1.circ | Slika 9.6 | A Commonsense approach.ppt (slide 25) |
| Slika 9.7 | Krug za dijeljenje u primjeru 6.9.circ | Slika 9.8 | A Commonsense approach.ppt (slide 18) |
| Slika 9.9 | A Commonsense approach.ppt (slides 22, 18) | Slika 9.10 | A Commonsense approach.ppt (slides 22, 18) |
| Slika 9.11 | A Commonsense approach.ppt (slide 15) | Slika 9.12 | A Commonsense approach.ppt (slide 2) |
| Slika 9.13 | A Commonsense approach.ppt (slide 17) | Slika 9.14 | A Commonsense approach.ppt (slide 16) |
| Slika 9.15 | A Commonsense approach.ppt (slide 35) | Slika 9.16 | A Commonsense approach.ppt (slide 5) |
| Slika 9.17 | A Commonsense approach.ppt (slide 31) | Slika 9.18 | A Commonsense approach.ppt (slide 8) |
| Slika 9.19 | A Commonsense approach.ppt (slide 6) | | |
| 10. Poglavlje 10 - Dodaci 244 | | | |
| Slika 10.1 | ---------- | Slika 10.2 | ---------- |
| Slika 10.3 | ACATTTOECC.ppt (slide 3) | Slika 10.4 | ACATTTOECC.ppt (slide 3) |
| Slika 10.5 | ACATTTOECC.ppt (slide 3) | Slika 10.6 | ACATTTOECC.ppt (slide 3) |
| Slika 10.7 | ACATTTOECC.ppt (slide 3) | Slika 10.8 | ACATTTOECC.ppt (slide 3) |
| Slika 10.9 | ACATTTOECC.ppt (slide 2) | Slika10.10 | ACATTTOECC.ppt (slide 2) |
| Slika10.11 | ACATTTOECC.ppt (slide 1) | Slika10.12 | A Commonsense approach.ppt (slides 110-111) |
| Slika10.13 | Euklidove udaljenosti.ppt (slides 1, 6) | Slika10.14 | Euklidove udaljenosti.ppt (slides 8-9) |
| Slika10.15 | ---------- | Slika10.16 | A Commonsense approach.ppt (slide 112) |
| | | | |

# *Sadržaj*

# Ispravak pogrešaka kodiranjem – praksom do teorije

## 0. POGLAVLJE 0 – UVOD

Ljudima je teško je raditi u okruženju buke, jer se povećava broj pogrešaka, a isto vrijedi i elektroničke uređaje. U okruženju s puno "buke" (*noisy*), signali se prečesto krivo tumače. Da bi tijekom razgovora utvrdili jesmo li ispravno čuli sugovornika, mi, ljudski prijemnici, koristimo razne oblike otkrivanja i ispravke komunikacijskih pogrešaka pokušavajući shvatiti ono što smo čuli.

Prije svega, koristimo *kontekst*[2]. Ako se čini da riječ ili izraz ne pripadaju temi razgovora, automatski zamijenimo zbunjujući dio onime što smo očekivali čuti. Ako zbunjenost i dalje postoji, zamolimo osobu neka ponovi ono što je upravo rekla i/ili neka govori glasnije. Sve su to oblici shema za ispravak pogrešaka koje se koriste u svakodnevnim ljudskim komunikacijama.

### *0.1. 0.1. Otkrivanje i ispravak pogrešaka*

Primjene tehnika otkrivanja i ispravke pogrešaka pri prijenosu ili pohrani binarnih podataka postalo je životno pitanje. Komunikacijske mreže s jedne te mediji za pohranu podataka (čvrsti diskovi, CD i USB memorijske jedinice) s druge strane, naznačuju područja u kojima se koriste kodovi za otkrivanje i ispravak pogrešaka. Zapravo, sve nabrojeno svodi se na zajednički "nazivnik", a to je trojka (izvor-kanal-odredište). Pri tomu se na medije za pohranu podataka, ova trojka razdvaja u dvije cjeline: izvor-kanal (zapis) podataka i kanal-odredište (čitanje) podataka.

Poduka o teoriji kodova za ispravak pogreška na uvodnoj razini vrlo je tegobna. Teorija koja se neposredno primjenjuje, tiče se apstraktnih algebarskih koncepata kao što su skupovi i operacije nad njima, grupa, prsten, polje vektorski prostori, polinomi, matrice, vektori. itd. Takvi pojmovi, koji se obično nalaze u prvih nekoliko poglavlja standardnih udžbenika zaštitnoga kodiranja, jednostavno preplaše studente. U ovoj skripti, svi temeljni pojmovi kodova za ispravak pogreška objašnjeni su pomoću:

1. izričito ILI (*exclusive*[3] *OR*, *EX-OR* ili kraće XOR) vrata,
2. linearnih registara s povratnom vezom (D-bistabila) te
3. osnovnih tehnika linearne algebre (ništa osim jednostavnoga množenja vektora i matrice).

Koristeći ovakav pristup, obuhvaćen je širok raspon pojedinačnih blok-kodova za ispravak pogrešaka bez oslonca na složene algebarske pojmove. Dekodiranje konvolucijskih kodova razvučenim (*difuzijskim*[4]) pragom, odabrano je kao način svojstven ovoj vrsti kodova. Uz cikličke kodove vezane su praskovite pogreške te njihovo otkrivanje i ispravak. Predstavljaju ih: ciklički Hammingov kod, Fire kod i Reed-Solomonovi kodovi. Primjena dijela linearne algebre u teoriji ispravke pogrešaka, pridružena je skripti kao dodatak. Opisana osnovna načela o kojima se raspravlja u cijeloj skripti, provjeravaju se s teorijskoga motrišta. Algebarska obrada primjenjuje se nakon što je student usvojio neposredno stečene praktične spoznaje i vještine o tome što se događa. Predstavljen materijal može se obraditi u jednome semestru diplomskoga studija elektrotehnike ili informatike. On se također može koristiti kao uvodni napredniji pregled, jer je jednostavan za osnovno razumijevanje, iz čega se može izgraditi ostatak spoznaja.

Ovome materijalu, kao nadgradnja (ili temelj), pridružen je opis simulacijskoga alata LogiSim. Tim simulacijskim alatom zorno je prikazan uvod u Booleovu algebru i uspostavljena je čvrsta povezanost između matematičkih (Booleovih) izraza i simulacijskih primjera. Tako je i materijal u čitavoj skripti upotpunjen simulacijskim primjerima kao sastavnim dijelom nastavnoga materijala.

Skripta sadrži primjere koji slikovito (u stvarnome i prenesenome značenju) objašnjavaju obrađivanu materiju. Skripti su pridruženi riješeni zadaci, čime se dodatno mrvi i bistri osnovna materija. Zbog

---

[2] *kontekst* … (lat. *contextus*) veza misli u govoru; sadržaj nekoga spisa u cjelini, smisao, spoj riječi, misaono dovršen ulomak pisanoga govora (teksta) koji točno određuje smisao pojedine riječi ili fraze koja ulazi u njega

[3] *exclusive* … ekskluzivan (lat. *exclusivus*) isključan, koji isključuje, koji je isključiv

[4] *difuzan* … (lat. *diffusus*) rasut; opširan, razvučen; *difuzna svjetlost* rasuta svjetlost, u svim pravcima odbijena (reflektirana) svjetlost (Klaić, Rječnik stranih riječi, 2002)

cjelovitosti obuhvata, ovoj skripti pridružena je skripta laboratorijskih vježbi (kao neodvojiv dio) s primjerima koji su u predavanjima ovlaš opisani. Ona služi za praktičan rad u laboratoriju simulacijskim alatom LogiSim za sada kao osnovom, s ciljem uvođenja National Instruments (NI) simulacijske i mjerne platforma (Multisim) za mjerenje radnih karakteristika ožičenih elektroničkih komponenata i sklopova, u stvarno-vremenskim uvjetima.

Pri komunikaciji elektroničkih uređaja, također se koristi *otkrivanje i ispravak pogrešaka* EDC (*error detection and correction*), na način sličan opisanome. U stvari, bez EDC većina elektroničkih komunikacija nije moguća. Razina šuma i glasovno ometanje previše su prisutni u elektroničkim medijima. EDC zahtijeva *učinkovitost*, a ona ovisi o veličini omjera signal/šum, ili *S*/*N* (*signal to noise ratio*). EDC se može provoditi na sljedećih pet načina:

1. povećanjem snage signala,
2. smanjenjem šuma signala,
3. ispravkom pogrešaka prema naprijed,
4. ponavljanjem prijenosa,
5. različitim nerazvrstanim metodama.

### 0.1.1. 0.1.1. POVEĆANJE SNAGE SIGNALA

Povećanje snage signala među uređajima isto je što i glasniji govor u bučnoj sobi. Intuicijski znamo, što je omjer signal/šum veći, manje su pogreške. Uz pretpostavku da u okruženju ne možemo ništa učiniti, onda možemo povećati snagu odašiljača. To uvijek nije moguće, jer većina komunikacijskih uređaja nema dodatne mogućnosti povećanja snage i već su oblikovani za rad maksimalnom snagom. Nedostataka pri korištenju ove sheme, nelinearne su karakteristike pojačala pa povećanje snage znači pogoršanje situacije, jer se pojačavaju i signal i šum. Ako se povećava snaga, šum signala raste od ometajućega, preko uznemirujućega do nepodnošljivoga.

### 0.1.2. 0.1.2. SMANJENJE ŠUMA SIGNALA

U komunikacijskome uređaju, jedini šumovi su toplinski i među-modulacijski šum sustava, jer na njih možemo izravno djelovati. Međutim, ne postoji niti jedan način smanjenja osnovnoga šuma pa smo ostavljeni na milost i nemilost okolini i to moramo prihvatiti kao zadano.

### 0.1.3. 0.1.3. ISPRAVAK POGREŠAKA PREMA NAPRIJED

Prepoznaju se tri vrste kanala: sa *slučajnim* pogreškama, *praskovitim* pogreškama te združivanje *slučajnih* i *praskovitih* pogrešaka. Prema njima napravljene su odgovarajuće sheme za njihov ispravak. Prva od njih je *ispravak pogrešaka prema naprijed* FEC. Za slučaj jednosmjernoga sustava, ne može se zatražiti ponavljanje prijenosa pa primatelj sam mora ispraviti pogreške. FEC kodovi i metode dijele se na blok-kodove (linearne i cikličke), konvolucijske kodove (Viterbijevo i serijsko dekodiranje, dekodiranje većinskom logikom te ispravak praskovitih pogrešaka).

U nedostatku dvosmjerne veze, koristi se *kodiranje pogreška prema naprijed* FEC (*Forward Error Coding*). Prijemnik nema stvarno-vremensku povezanost s odašiljačem i ne može provjeriti je li pravilno primio blok podataka. On mora donijeti odluku o primljenim podacima i učiniti sve što može kako bi popravio podatke ili obznanio alarm zbog pogreške. FEC tehnike opterećuju link, dodavanjem *suvišnih podataka* i/ili unošenjem *kašnjenja*. Nedostatak je *manja brzina prijenosa*. Ove tehnike umanjuju potrebu za *promjenom snage*. Za istu snagu, sada može se postići manja količina pogrešaka. Komunikacija ostaje jednosmjerna (*simplex*), a prijemnik otkriva i ispravlja pogreške ali mu je složenost veća.

Tri najpoznatije skupine kodiranja: *blok-kodovi*, *konvolucijski kodovi* i *turbo-kodovi*, koriste FEC metode. Blok-kodovi rade blokovima bitova koristeći programski algoritam. Konvolucijski kodovi su *kontinuirani* kodovi, jer stalno djeluju na određen broj bitova. Preplitanje turbo-kodova uspješno djeluje u kanalima s iščezavanjem signala i spregom prethodnih dvaju vrsta kodiranja. Sve tri tehnike, osim u cjelovitim komunikacijskim kanalima, koriste se i u polu kanalima kao što su računala, CD/DVD uređaji, digitalni snimači zvuka DAR (*digital audio recording*) i televizija visoke razlučivosti HDTV (*high-definition television*) na zahtjev.

### 0.1.4. 0.1.4. AUTOMATSKO PONAVLJANJE PRIJENOSA

Automatski postupci za ponavljanjem prijenosa ARQ (*Automatic Repeat Request*) zahtijevaju prijenos u dva smjera (*duplex*). Uz dodatnu povratnu potvrdu o ispravno primljenome signalu, oblikuje se shema koja se zove *potvrda ponavljanja* ARQ (*Acknowledgement Retransmit ARQ*). Prijemnik provjerava svaki blok podataka te u slučaju pogreške, traži ponavljanje prijenosa. Ako je *S*/*N* velik, ponavljanje prijenosa nije tako često, uz veliku propusnost. Smanjenjem *S*/*N*, povećava se *premašenje* (*overhead*) pa je ovo ustupak između ukupne propusnosti i kašnjenja na jednoj te složenosti prijemnika na drugoj strani. ARQ se najčešće koristi u žičnim i lokalnih mrežama.

U dvosmjernim sustavima, pogreškama se upravlja zahtjevom za automatskim ponavljanjem prijenosa ARQ (*Automatic Repeat Request*). Za jednosmjerne (*half-duplex*) kanale oblikovane su ARQ metode: stani i pričekaj (*Stop-and-Wait*), ACK (*Acknowledgement*) i NAK (*Negative Acknowledgement*). Za potpune dvosmjerne sustave (*full-duplex*) koriste se kontinuirane ARQ: hajde natrag za *N* (*Go-Back-N*) i selektivno ponavljanje (*Selective Repeat*). NAK je oblikovan za potpune dvosmjerne sustave. Hibridna ARQ metoda predstavlja združivanje prvih dviju metoda: FEC + ARQ.

Ako se u prijemniku otkriju pogreške, zahtjev se šalje odašiljaču da ponovi poruku i ponavlja ju se sve dok se poruka ne primi ispravno. Ako se primi NAK (negativna potvrda), odašiljač ponavlja prijenos od pogrešne kodne riječi i još $n$–1 kodnih riječi koje slijede. Prema izboru, odašiljač može poslati samo one kodne riječi koje još nisu potvrđene. U metodi dekodiranja maksimalnom vjerojatnosti (*Maximum Likelihood Decoding*), dekoder minimalne udaljenosti MDD (*Minimum Distances Decoder*) odabire kodnu riječ najmanje udaljenu od poslane riječi primljene preko kanala. To je ekvivalentno[5] smanjenju *prosječne kvadratne pogreške* MSE (*mean squared error*).

### 0.1.5. 0.1.5. RAZLIČITE NERAZVRSTANE METODE

Ako se pri slušanju rado prijemnika, signal pojačava i smanjuje, onda se uređaj premjesti. Pri korištenju bežičnoga telefona, možemo mijenjati kanale, ili zamoliti sugovornika da uspostavi novu vezu. U vlažnim područjima, često imamo dvije međusobno razdvojene zemaljske postaje, obje primaju isti signal pa se tako povećava *S*/*N*. U satelitskim komunikacijama mogu se koristiti različite polarizacije za povećati iskoristivost spektra pri slanju istih informacija pomoću ispravke pogrešaka. Sve ove tehnike pripadaju skupu *različitih nerazvrstanih metoda*, a glavna im je svrha poboljšanje kakvoće signala korištenjem raspoložive zalihosti. Pojava neočekivanoga izvora u slučaju mobitela također spada u ovu skupinu, jer kretanjem signala u prostoru uzrokuje smanjenje amplitude, ali pomoć složene obrade možemo iskoristiti i malu snagu ometanoga signala za sjediniti[6] i poboljšati odnos *S*/*N*.

## 0.2. 0.2. Uvod u problematiku

Razumijevanje zaštitnoga kodiranja, započinje Shannonovim komunikacijskim kanalom (*izvor-prijenosni kanal-odredište*), sagledavanjem i razumijevanjem globalne slike te određivanja razloga i mjesta zaštite informacija pri prijenosu. Utemeljitelj teorije informacija C. E. Shannon objavio je 1948. godine *Matematičku teoriju komuniciranja*[7]. Na informaciju koja se prenosi od izvora do odredišta općenito djeluju smetnje pa se informacija u prijenosu oštećuje do razine potpunoga gubitka. Zato se prijenos informacija nastoji razdijeliti u cjeline pogodne za rukovanje i obradu. Pogodnost koja stoji na raspolaganju jest *digitalizacija* svih vrsta informacija, koja prevladava u svijetu razmjene informacija.

Sama digitalizacija temelji se na impulsno-kodnoj modulaciji PCM (*Pulse Code Modulation*) i njezinim načelima *uzorkovanja*, *kvantizacije* i *kodiranja*. Ova modulacija široko se obrađuje kroz nekoliko kolegija pa se ovdje samo spominje.

Sasvim normalno koristimo CD i DVD uređaje, računala s pripadnom opremom, pametne telefone i umrežene komunikacijske sustave širom svijeta. Osnova rada svega toga je rad tranzistorske komponente kao sklopke u logičkome sklopu poznatome pod nazivom "vrata isključivo (*exclusive*)

---

[5] *ekvivalent* … (lat. *aequi-valens*) istovrijedan, jednakovrijedan; isto značenje; jednaka vrijednost, ista vrijednost; stvar iste vrijednosti, zamjena za vrijednost, odšteta, naknada;

[6] *kombinirati* … (lat. *combinare*) sastaviti, složiti, sjediniti, spajati, spojiti, srediti; praviti plan za neki posao, smisliti

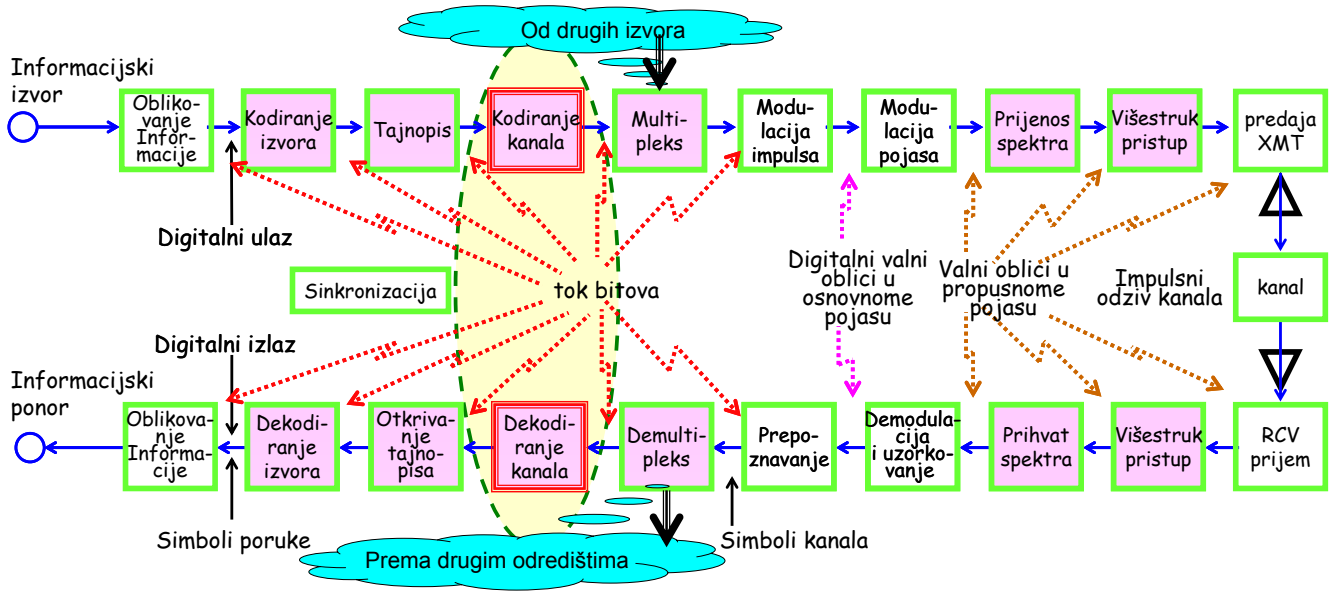
[7] *A Mathematical Theory of Communication*, Bell System Tech. J. 27

ILI" (EXOR *gate*) i ništa više. Sva složenost komunikacijskoga svijeta oslanja se samo na to. Naravno sve je podržano procesorski upravljanim sklopovima, ali i njihov rad leži na potpuno istim temeljima.

Namjera ove skripte jest ogoliti problematiku do samih navedenih temelja, odnosno počevši od njih ocjeloviti sklopove, prikazujući njihov rad prikladnim simulacijskim alatima (LogiSim). Druga, ne manje važna stavka, jest matematički alat *linearne algebre* koji teorijskim temeljima prati i podupire praktične primjene te učvršćuje čitavu cjelinu. Napomenimo da je sam matematički alat linearne algebre još stariji od prije spomenutih informacijskih temelja. Uz linearnu algebru, primjenjuje se Booleova algebra čiji začeci sežu od kraja pretprošloga stoljeća. Dakle usklađen rad Booleove algebre, linearne algebre te prakse i teorije informacija, učinkovito oživotvoruje današnji komunikacijski svijet.

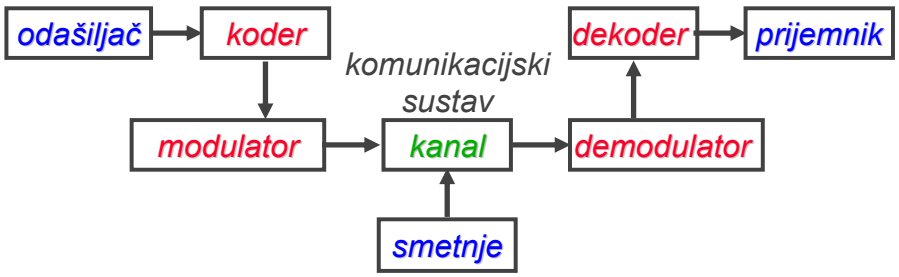
## *0.3. 0.3. Komunikacijski kanal*

Slika 0.1 prikazuje cjelovit komunikacijski sustav s označenim područjima kojima se bavi ova skripta, a to su *kodiranje* i *dekodiranje* informacija.




*Slika 0.1. Cjelovit prikaz komunikacijskoga kanala i područje koje se proučava u ovoj skripti*

Na nekoliko mjesta u komunikacijskome kanalu, digital(izira)na informacija pretvara se u analognu i obrnuto s ciljem povećanja pouzdanosti, brzine, kakvoće i količine informacije pri prijenosu od izvora do odredišta. Kako? Opisat će se u ovoj skripti. Digitalni komunikacijski sustavi često se opisuju slikom 0.2.




*Slika 0.2 Sustav digitalne komunikacije*

Komunikacijski kanal (*Communications Channel*) dio je komunikacijskoga sustava koji unosi pogreške. Medij u kanalu može biti radio, žica, koaksijalni kabel, svjetlovodni kabel, magnetska traka, običan disk, optički disk, USB uređaj ili bilo koji zamisliv medij.

*Modulator* pretvara izlaz iz kodera digitalnoga oblika, u format prikladan za prijenos kanalom, koji je obično analogan. Nakon što prođe kanalom sa šumom, *demodulator* pokuša oporaviti otkriven, oštećen simbol i pretvoriti ga u ispravan simbol. Ako se otkrije neispravan simbol, dekoder pokuša ispraviti otkrivene pogreške.

Koder (*encoder*), toku podataka dodaje suvišne bitove, da bi se stvorila kodna riječ. Dekoder (*decoder*) koristi suvišne bitova za otkrivanje i/ili ispravak onoliko mnogo pogrešaka koliko dopušta određeno upravljanje pogreškama kodiranjem.

Glavni problem je osmisliti i primijeniti par koder/dekoder preko kanala tako da se informacije prenose (ili snimaju) u okruženju šuma toliko brzo (ili gusto) koliko je to moguće, odnosno, da se informacije mogu pouzdano obnoviti na izlazu dekodera kanala.

## 0.4. 0.4. Vrste kodova

Kodiranje kanala dijeli se u: proučavanje valnoga oblika (ili oblikovanja signala) te proučavanje kodiranja i ustrojenih (*structured*) nizova ili ustrojene zalihosti (*structured redundancy*), kao što prikazuje tablica 0.1.

*Tablica 0.1: Okvir zaštitnoga kodiranja*

| Kodiranje kanala | |
|---|---|
| Valni oblici | Ustrojeni nizovi |
| M-struka signalizacija<br>Dijametralno različiti<br>Okomiti<br>Modulacija kodirane rešetke | Blok-kodovi<br>Konvolucijski kodovi<br>Turbo kodovi |

Kodiranje *valnoga oblika* bavi se pretvorbom valnih oblika u "bolje" valne oblike, kako bi postupak otkrivanja bio manje podložan pogreškama. *Ustrojeni nizovi* bave se pretvorbom podatkovnih nizova u "bolje" nizove, koji imaju ugrađenu zalihost (zalihosne bitove). Onda se zalihosni bitovi mogu koristiti za otkrivanje i ispravak pogrešaka. Danas su u općoj upotrebi dvije (tri) strukturno različite vrste kodova (ovo se može prispodobiti pojmovima logike kao *teza-antiteza-sinteza*):

1. blok-kodovi,
2. konvolucijski[8] kodovi i
3. turbo kodovi.

Koder za blok-kod dijeli informacijski niz u blokove poruka od kojih je svaki blok dugačak $k$ informacijskih bitova (simbola). Blok poruke predstavljaju se binarnom $k$-torkom $\mathbf{u} = [u_0, u_1, \ldots, u_{k-1}]$, koja se naziva *poruka*. Postoji ukupno $2^k$ različitih mogućih poruka. Koder potpuno nezavisno pretvara svaku poruku u $n$-torku $\mathbf{v} = [v_0, v_1, \ldots, v_{n-1}]$ diskretnih simbola, što se zove *kodna riječ*.

Druga skupina kodova, *konvolucijski kodovi*, vrsta su koda za ispravak pogrešaka koji generira paritetne[9] simbole načelom klizne primjene Booleove polinomske funkcije pri protoku podataka. Klizna primjena predstavlja "konvoluciju" kodera nad podacima, što dovodi do pojma *konvolucijskoga kodiranja*.

*Turbo kodovi* su sinteza prethodno opisanih kodova. To je usavršena struktura[10] ulančanoga koda i iteracijskoga[11] algoritma za dekodiranje. Kao zasebna kategorija nalazi se na popisu ustrojenih nizova (*structured sequences*) (tablica 0.1) pod imenom *turbo*.

## 0.5. 0.5. Upravljanja pogreškama kodiranjem

Blok-dijagrami prikazani na slikama 0.1 i 0.2 predstavljaju jednosmjeran komunikacijski sustav. Prijenos (ili digitalni snimak) strogo je jednosmjeran, od odašiljača do prijemnika. Tradicijska uloga upravljanja pogreškama kodiranjem je, kanal sa smetnjama, napraviti prihvatljivim zbog smanjenja čestoće događanja pogrešaka. Pogreške su: *pogreške bitova*, *pogreške poruka*, ili *neotkrivene pogreške* pa se može učiniti sljedeće:

---

[8] Konvolucija je integral koji opisuje veličinu preklapanja jedne funkcije drugom dok se jedna pomiče preko druge (vidi http://mathworld.wolfram.com/Convolution.html).

[9] *parity* … parnost ili paritet (vidi poglavlje 1.7 Pojam pariteta)

[10] *struktura* … (lat. *structura*) način građenja, sastav, ustrojstvo; raspored; geol. način kako su spojeni dijelovi stijena i planina; tvorevina, građevina; usp. konstrukcija

[11] *iteracija* … ustrajno ponavljanje

- *Smanjiti pojavu neotkrivene pogreške*. Današnji kodovi za otkrivanje pogrešaka toliko su učinkoviti, da u svim praktičkim primjenama nema neotkrivenih pogrešaka.
- *Smanjenje gubitaka u komunikacijskim sustavima*. Za satelitske komunikacije, kodiranje može smanjiti potrebu za snagom, jer poruke čija je snaga blizu toplinske razine šuma i dalje se mogu ispravno oporaviti.
- *Nadvladavanje zagušenja* (*jam*). U prisutnosti impulsa ometanja, kodiranjem se može postići *dobitak kodiranja* od preko 35 [dB].

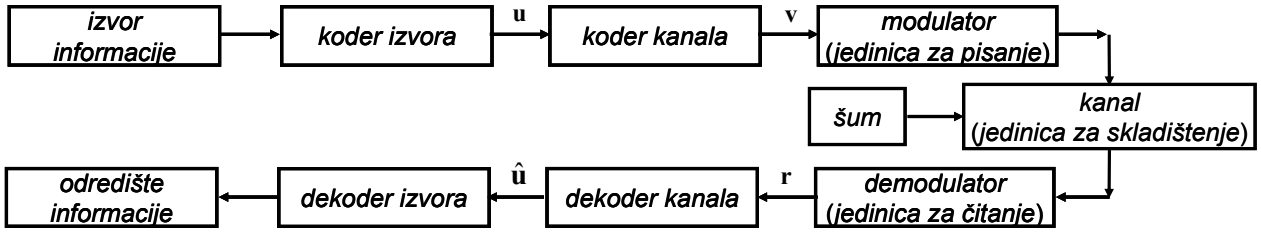
Postoje tzv. "polu-kanali" koji se često ne prepoznaju, a veoma široko se koriste. Primjeri polu-kanala su svi današnji memorijski uređaji (nekada diskete), čvrsti diskovi, memorijski USB uređaji, a naročito CD i DVD mediji) koji pohranjuju informacije (to je samo pola kanala), a pružaju ih na korištenje prema potrebi u neko pogodno vrijeme preko druge polovice (istoga ili nekoga drugoga) kanala.

Matematički alati koji se još uvijek koriste u primjeni zaštitnoga kodiranja su: *teorija vjerojatnosti* i *matematička statistika*. Ono što se stidljivo pojavljivalo kao cilj i sredstvo istraživanja u posljednja dva desetljeća, su *ekspertni sustavi* i *neizrazita logika*. Tek se očekuje njihova šira primjena, jer su to temelji *umjetne inteligencije* kojoj se teži u svim znanstvenim i stručnim područjima.

## 0.6. 0.6. Pouzdan prijenos i pohrana digitalnih informacija

U posljednjih tridesetak godina, došlo je do povećanja potražnje za učinkovitim i pouzdanim digitalnim prijenosom podataka odnosno sustavima za pohranu podataka. Ovaj zahtjev ubrzao se pojavom podatkovnih mreža velikih brzina za razmjenu, obradu i skladištenje digitalnih podataka u poslovanju. Pri oblikovanju ovih sustava nužno se spajaju *komunikacijske* i *računalske* tehnologije.

Shannon je pokazao da se pogreške što ih izaziva šum kanala ili medij za pohranu podataka, mogu svesti na željenu razinu bez žrtvovanja brzine prijenosa informacija ili brzine skladištenja, dok god je brzina prijenosa manja od kapaciteta kanala. Današnjim digitalnim sustavima velikih brzina, potrebna je pouzdanost te je korištenje kodiranja za kontrolu pogrešaka postao sastavni dio oblikovanja modernih komunikacijskih i digitalnih sustava za pohranu podataka. Tipičan sustav prijenosa (ili pohrane) može se prikazati blok-dijagramom na slici 0.3.




*Slika 0.3*: *Blok-dijagram tipičnoga sustava za prijenos i/ili skladištenje podataka*.

*Izvor informacije* je osoba ili stroj. Izlaz izvora je kontinuiran valni oblik ili niz diskretnih simbola. *Koder izvora* pretvara izvorni izlaz u niz binarnih znamenaka (bitova), a zove se *informacijski niz* **u** (niz se označava i razumijeva kao vektor). Koder izvora idealno je oblikovan tako, da je broj bitova u jedinici vremena potrebnih za prikaz izlaza izvora, sveden na minimum pa se izlaz izvora može jednoznačno obnoviti iz informacijskoga niza **u**. Ova skripta **ne obrađuje kodiranje izvora** niti impulsno-kodnu modulaciju PCM.

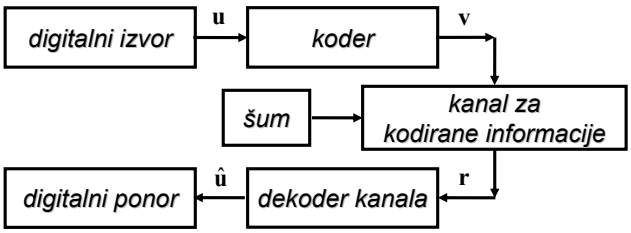
*Koder kanala* pretvara *informacijski niz* **u** u diskretan *kodiran niz* koji se zove *kodna riječ*. U većini slučajeva to je binaran niz, iako se u nekim primjenama koriste i ne-binarni kodovi. Prva središnja tema ove skripte su *oblikovanje i primjena kodera kanala* za borbu protiv šuma u okruženju u kojemu se kodne riječi moraju prenositi i/ili pohranjivati.

Diskretni simboli nisu pogodni za prijenos preko fizičkoga kanala niti za snimanje na digitalni skladišni medij. *Modulator* (ili *jedinica za pisanje*) pretvara svaki izlazni simbol kodera kanala u valni oblik trajanja $T$ sekundi, pogodan za prijenos (ili snimku). Ovaj valni oblik ulazi u kanal (ili medij za pohranu) i na njega djeluje šum. Tipični prijenosni kanali su telefonske veze, mobilna stanična telefonija, radio telemetrija visoke frekvencije, mikrovalne i satelitske veze, svjetlovodni kabeli itd..

Tipični mediji za pohranu uključuju osnovne i poluvodičke memorije, magnetske trake, diskove, kompaktne diskove, optičke memorijske jedinice i tako dalje. Svaki od tih primjera podložan je raznim vrstama šuma kao poremećaju. *Demodulator* (ili *jedinica za čitanje*) obrađuje svaki primljeni valni oblik trajanja $T$ i proizvodi diskretan (kvantiziran) ili kontinuiran (ne-kvantiziran) izlaz. Niz na izlazu iz demodulatora podudaran je kodiranome nizu **v**, a zove se *primljen niz* **r**.

*Dekoder kanala* pretvara primljen niz **r** u binarni niz $\hat{\mathbf{u}}$ (*procijenjen informacijski niz*). Strategija dekodiranja temelji se na pravilima kodiranja kanala i karakteristikama šuma u kanalu (ili mediju za pohranu). U idealnome slučaju, $\hat{\mathbf{u}}$ je preslik informacijskoga niza **u**, iako šum može izazvati neke pogreške prije dekodiranja. Druga središnja tema ove skripte su oblikovanje i primjena dekodera kanala da bi se smanjila vjerojatnost pogreške dekodiranja.

*Dekoder izvora* pretvara primljen niz $\hat{\mathbf{u}}$ (*informacijski*) u *procjenu* izlaza izvora pa ovu procjenu isporučuje *odredištu*. Procjena je u pravilu vjerno obnovljen izvorni izlaz osim ako je šum u kanalu (ili mediju za pohranu) prevelik. Usmjerimo li pozornost na koder i dekoder kanala, izvor informacija i koder izvora mogu se sjediniti u *digitalni izvor* s izlazom **u**. Modulator (ili jedinica za pisanje), kanal (ili medij za pohranu) i demodulator (ili jedinica za čitanje) mogu se sjediniti u *kanal za kodiranje* s ulazom **v** i izlazom **r**. Dekoder izvora i odredište mogu se sjediniti u *digitalni ponor* s ulazom $\hat{\mathbf{u}}$. Rezultat ovih povezivanja prikazuje skraćen blok-dijagram na slici 0.4.




*Slika 0.4*: *Pojednostavnjen model kodiranoga sustava*.

Glavni problem što ga obrađuje ova skripta je osmisliti i primijeniti par koder/dekoder tako da se informacije mogu prenositi (ili snimiti) u okruženju šuma toliko brzo (ili gusto) koliko je to moguće. Također, informacije se mogu i moraju pouzdano obnoviti na izlazu dekodera.

## *0.7. 0.7. Strategije upravljanja pogreškama*

### 0.7.1. 0.7.1. ZALIHOST

Prvi način korištenja zalihosti za otkrivanje je li se dogodila pogreška, jest korištenje *paritetnih bitova* (zalihosni bitovi dodani podacima). Prijemnik ne pokušava ispraviti pogreške, već od odašiljača traži ponovno slanje podataka za što je potrebna dvosmjerna veza. U načinu jednosmjerne veze, paritetni bitovi oblikovani su, i za otkrivanje i za ispravak pogrešaka. Strategije upravljanja pogreškama je automatski ispravak pogrešaka prema naprijed FEC pa se u skripti analiziraju i oblikuju FEC sustavi.

Glavna prednost ARQ u odnosu na FEC je, da otkrivanje pogreške zahtijeva mnogo jednostavniju opremu za dekodiranje nego za ispravak pogrešaka. Također, ARQ je prilagodljiva u smislu da se informacija emitira samo ako dođe do pogreške. Nasuprot tomu, ako je brzina pogrešaka u kanalu velika, prečesto se ponavlja slanje pa brzina kojom se novo generirane poruke pravilno primaju - *propusnost sustava* (*system throughput*), spušta se prema ARQ. U ovoj situaciji, hibridna kombinacija FEC za najčešće uzorke pogrešaka uz otkrivanje pogreške i ponovno odašiljanje za manje vjerojatne uzorke pogrešaka, učinkovitija je od same ARQ.

### 0.7.2. 0.7.2. PARAMETRI KOJI SU RASPOLOŽIVI ZA UGAĐANJE

Temeljni raspoloživi resursi[12] su: *snaga signala*, *vrijeme* i *pojasna širina*. Ta tri parametra čine međusobne ustupke (jedan prema ostalima). Temelji na kojima se rade ustupci, ovisit će o onome što se u danoj situaciji traži od svakoga parametra. Opći cilj je, postizanje maksimalnoga prijenosa podataka uz minimalno kašnjenje te održavanje prihvatljive kakvoće prijenosa. Kakvoća prijenosa bavi

[12] *resurs* … (fr. *ressource*) pomoćno sredstvo, izvor pomoći; izvor (privrede) iz kojega se dobavljaju sirovine

se *vjerojatnošću pogreške bita* $P_b$, u prijemniku. Postoje i drugi čimbenici koji određuju kakvoću prijenosa, u najširemu smislu ali naglasak je na $P_b$.

Shannon-Hartleyev zakon za kapacitet komunikacijskoga kanala pokazuje dvije stvari. Prvo, pokazuje (količinski) kako se u idealnome sustavu mogu ugađati *propusnost* u odnosu na *snagu signala* i drugo, daje teorijsku granicu za brzinu prijenosa, pouzdanu i bez pogrešaka. Kako bi se postigla ta teorijska granica, mora se pronaći odgovarajuća shema kodiranja. Treba napomenuti da postoji *kašnjenje* kao još jedna veličina kojom se može/mora upravljati.

Cilj je ostvariti potrebnu brzinu prijenosa podataka uz ograničenu raspoloživu širinu pojasa kanala i ograničenu raspoloživu snagu napajanja. Brzina prijenosa podataka mora zadovoljiti prihvatljiv BER (*Bit Error Rate*) i vrijeme kašnjenja. Ako unutar tih ograničenja nekodiran prijenos ne može postići željeni BER, onda mora pomoći upravljanje pogreškama kodiranjem bez narušavanja Shannon-Hartleyeva zakona.

### 0.7.3. 0.7.3. KODIRANJE KANALA

Upravljanje pogreškama kodiranjem (također se naziva i kodiranje kanala) koristi se za otkrivanje i ispravak simbola koji se prime kao pogrešni. Za upravljanje pogreškama služi oblikovanje spektra, kodiranje izvora, itd.. Otkrivanje pogrešaka može se koristiti kao početan korak u tehnici ispravke pogrešaka, npr. uputiti prijemnik da automatski zahtijeva ponavljanje već prenesenoga signala. Rezultat uspješnoga ponovljenoga slanja oštećenih podataka jest ispravan prijem informacije.

Već prije, ukratko se opisao rad ARQ tehnika. Ako su ARQ tehnike nezgodne, kao što je slučaj, na primjer pri velikome propagacijskome kašnjenju u mediju za prijenos, onda je prikladnije *kodiranje prema naprijed s ispravkom pogrešaka* FECC (*forward error correction coding*). FECC sadrži dodatne informacije (tj. zalihost) u prenesenim podacima, a to se može iskoristiti ne samo za *otkrivanje* pogrešaka, već i njihov *ispravak*, bez potrebe za ponovnim slanjem.

Ovo poglavlje započelo je općom raspravom o upravljanju pogreškama u najširem smislu. Prepoznalo se i ukratko opisalo pet posebnih metoda upravljanja pogreškama. Jedna od njih, FECC metoda, detaljno se opisuje u nastavku. Definiraju se: Hammingova udaljenost između para kodnih riječi i težine kodnih riječi. Rasprava o FEC kodovima, (tablici 0.2), počinje opisom blok-kodova uz posebnu pozornost na skupinu linearnih kodova: ciklički, Hammingov[13], Golayev, BCH i Reed-Solomonov kod.

*Tablica 0.2: Pregledna tablica kodova za upravljanje pogreškama.*

<table>
<tr><th colspan="3">ARQ</th><th colspan="6">FEC kodovi</th></tr>
<tr><td>stani i pričekaj</td><td colspan="2">kontinuirana ARQ (povezano kao cjevovod)</td><td colspan="5">blok</td><td>konvolucijski</td><td>turbo</td></tr>
<tr><td></td><td>hajde natrag za N</td><td>selektivno ponavljanje</td><td>nelinearne skupine</td><td colspan="4">linearne skupine</td><td rowspan="4"></td><td rowspan="4"></td></tr>
<tr><td></td><td></td><td></td><td></td><td colspan="4">generirane polinomom (ciklički)</td></tr>
<tr><td></td><td></td><td></td><td></td><td>Golay</td><td colspan="3">BCH</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td>Reed-Solomon</td><td>Binarni BCH</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td>Hamming (e=1)</td><td>e>1</td></tr>
</table>

U skripti su prikazani koncepti kodiranja (ne strogo matematički), već prema specifičnim primjerima skupnih, cikličkih, konvolucijskih i turbo kodova. Detalji pojedinih primjena opisani su u skripti laboratorijskih vježbi i dosljedno prate predstavljenu teorijsku materiju.

[13] Hammingovi kodovi (1950) izvorno su se koristili za nadvladavanje pogrešaka u međumjesnim tefonskim pozivima.

# 1. *Laboratorijska vježba 0: Brojevni sustavi, kodovi, osnovni i složeni logički sklopovi, logička algebra, uvod u LogiSim*

Napomena! Detalji ove vježbe su na MOODLE pod:
Laboratorijska vježba 0, u datotekama: Logički sklopovi, Flip Flop i Logisim

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*



LogiSim 2.7.1

# 1. POGLAVLJE 1 - TEMELJI ZAŠTITNOGA KODIRANJA

Osnove digitalnoga prijenosa informacija, a time i zaštitnoga kodiranja, postavio je Claude Elwood Shannon 1948. Za diskretan kanal sa šumom, definiraju se pojmovi: apriorna i aposteriorna vjerojatnost, statističke zavisnosti kanala, mjera kvalitete prijemnoga uređaja, svojstva i vrste sadržaja informacije, logaritamske mjerne jedinice bît, nît, dît, entropija, transinformacija, irelevantnost, ekvivokacija, brzina informacije, kapacitet kanala i njihove međusobne ovisnosti.

Slijedi kratak opis digitalnih, osnovnih i izvedenih modulacija (BPSK, QPSK, 8-QAM, 16-QAM, …, OFDM, …), optimalan kod, …. Poglavlje završava uvodom u rad digitalnih sklopova i u simulacijski alat LogiSim opisom pojma *pariteta* i praktičnim primjerima vježbi. Na kraju su popisane definicije svih potrebnih, uvedenih pojmova.

## *1.1. 1.1. Zaštitno kodiranje signala*

Objašnjenje naziva ovoga kolegija: "*Zaštitno kodiranje signala*", započinje najprije objašnjenjem riječi *signal* u kontekstu pojma *informacije*, a onda i same riječi *kodiranje*. Na kraju se objašnjava složenica *zaštitno kodiranje*.

### 1.1.1. 1.1.1. INFORMACIJA …

Signali su materijalni nositelji informacije u najširemu smislu i mogu poprimiti razne oblike.[14] Najčešći materijalni nositelji informacija u komunikacijama su naponi, struje, svjetlo i općenito elektromagnetski valovi. *Reverzibilan* postupak,[15] pretvorba je informacije iz jednoga oblika u drugi te ponovna pretvorba u izvoran oblik. Primjer *ireverzibilnoga* postupka,[16] prijevod je iz jednoga prirodnoga jezika u drugi.

### 1.1.2. 1.1.2. KODIRANJE …

*Kodiranje* je postupak kojime se ostvaruje maksimalna reverzibilnost. Utjecaj smetnji nikada se ne može potpuno ukloniti pa ni uvođenjem kodova s otkrivanjem i ispravkom pogrešaka. Postupak kodiranja generira uzorak podataka. Kodiranje informacije prikaz je skupa informacija pomoću niza simbola druge abecede. To je dio mehanizma kojime se omogućuje prijenos podataka velikim brzinama kroz kanale sa šumom.

### 1.1.3. 1.1.3. ZAŠTITNO …

*Zaštitno* kodiranje, metoda je zaštite od smetnji zbog prolaza informacije kroz kanal sa smetnjama. Tajnopis (kriptografija), viša je razina zaštite informacija od zloupotrebe, a u nju spadaju: tajnopis zasnovan na kodovima s ispravkom, kriptografski sustavi s tajnim ključem, DES (*data encryption standard*) algoritam, vjerodostojnost (*authentication*), potpisi i zaštita od prijevara. *Linijsko* kodiranje dio je zaštitnoga kodiranja korištenjem modulacijskih postupaka digitalnih modulacijskih metoda. Njihovi ciljevi su: prijenos podataka od *izvora* do *odredišta* preko *pojedinoga komunikacijskoga kanala* te združivanje više kanala u jedan kanal s ciljem smanjenja pogrešaka i učinkovitijim korištenjem pojasne širine (*bandwidth*) kanala.

---

[14] Teorija informacija 1.doc

[15] *reverzibilan* … (lat. *reversibilis*) povratan, koji se može povratiti; koji se može prevrtati, okretati (sukno); fiz. povratan (supr. *ireverzibilan*); budući da u prirodi ne postoji nijedno zbivanje (nijedan proces) koje nije povezano trenjem ili provođenjem topline, onda su sva zbivanja u prirodi zapravo ireverzibilna, a reverzibilna zbivanja su samo idealan graničan slučaj; tijek svakoga zbivanja u prirodi usmjeren je tako da se zbroj entropija svih tijela koja sudjeluju u zbivanju povećava; kod reverzibilnoga zbivanja, graničnoga slučaja, ovaj zbroj ostaje nepromijenjen

[16] *ireverzibilan* … (lat. *irreversibilis*) nepovratan; *ireverzibilni procesi*, procesi čiji se smjer i tijek ne mogu vratiti, npr. životni procesi (razvoj od jajeta do pune zrelosti organizma), povijesni događaji i dr.; fiz. proces koji se ni na jedan jedini način ne može potpuno vratiti (svi drugi procesi su reverzibilni, tj. mogu se vratiti); drugim riječima: ireverzibilan proces je onaj nakon kojega se ne može, čak ni uporabom svih u prirodi postojećih sredstava (reagensa) uspostaviti svuda točno početno stanje, tj. ne može se cjelokupna priroda vratiti u stanje koje je imala u početku procesa

### 1.1.4. 1.1.4. DETERMINISTIČKI I STOHASTIČKI SUSTAVI

Pretpostavlja se da se u svakomu sustavu može definirati skup *S* svih mogućih stanja toga sustava. Rad sustava u vremenu jest zauzimanje određenoga stanja iz skupa *S.* Ako se na temelju poznavanja stanja sustava u određenomu trenutku i spoznaja zakonitosti koje djeluju (u sustavu ili na sustav), može jednoznačno predvidjeti stanje sustava u budućnosti, kaže se da je sustav *odrediv* (*deterministički*).

Ako se na osnovi poznatih zakonitosti i stanja sustava u sadašnjosti može odrediti samo vjerojatnost da će sustav poprimiti određeno stanje iz uočenoga skupa mogućih stanja u budućnosti, onda se kaže da je sustav *vjerojatnosni* (*stohastički*/*probabilistički*). Onda se samo može pronaći određena razdioba vjerojatnosti na skupu svih mogućih stanja u budućnosti pa ostaje "manja ili veća neizvjesnost" u vezi stânja koje će sustav imati u budućnosti. Naravno, u prirodi su vrlo česti sustavi koji su svrstani između dviju spomenutih krajnosti, tj. takvi sustavi koji nisu ni deterministički, a ni stohastički pa se prognoze o zauzimanju njihovih stanja u budućnosti, mogu dati manjom ili većom vjerojatnošću.

### 1.1.5. 1.1.5. ENTROPIJA SUSTAVA

Korisno je imati određen brojčani pokazatelj za mjerenje neizvjesnosti pa će taj parametar (obično se naziva *entropija*[17] sustava), također poslužiti i kao mjera nereda ili kao stupanj razlikovanja od determinističkoga sustava. Entropija izražava stupanj jakosti statističkih veza unutar promatranoga sustava. Ako statističke zavisnosti prerastu u funkcijske veze, to je deterministički sustav i onda je entropija nula. Entropija se podrobnije opisuje u poglavlju 1.3.

Informacija je jedna od četiri životno razlučive komponente: *materija, energija, informacija i svijest*. Odnosi između ovih komponenata, mogu se opisati kao međudjelovanje dviju ili više jedinki koje se ostvaruje posredstvom materije, energije ili svijesti, a koje upravlja jedinke prema razlogu njihova postojanja.

*Sadržaj informacije* mjera je *neodređenosti*. Prijemnik poruke, mora poznavati jezik poruke. Poruka treba biti korisna prijemniku, kako bi bila poticaj za donošenje novih odlukâ i akcijâ. Razmjenu poruka jedinki prate emocijsko-afekcijske reakcije. Na temelju iznijetoga, moguće je prepoznati slijedeća informacijska motrišta: *sintaktičko* (*uređivačko*), *semantičko*[18] (*označivačko*), *pragmatičko* (*djelatničko*) *i estetsko* (*emocijska i poslovna inteligencija*).

Informacija *zagađuje* okolinu kao što to čini i trošenje energijskih resursa. Ona može srušiti ili posve uništiti uvjete življenja čovjeka u nekoj okolini. Informacijsko zagađenje je za čovjeka pogubnije to više što se odvija na višoj razini između četiriju spomenutih motrišta. Informacijsko doba prati problem *informacijske zasićenosti*. Paradoks je, da ako čovjek raspolaže s više informacija, onda je nesigurniji pri odlučivanju. U njegovu djelovanju prisutna je sve veća doza zanemarivanja.

### 1.1.6. 1.1.6. PODATAK I INFORMACIJA

Pojmovi *podatak* i *informacija* nisu sinonimi. Podatak je skup znakova, koji prikazuju jedan ili nekoliko dijelova informacije. Podatak je podskup *informacije*. Iako *podatak* posjeduje i sintaktičko i semantičko svojstvo, pojam *informacije* prvenstveno je vezan za semantička svojstva, a pojam *podatka* prvenstveno je vezan za sintaktička svojstva.

*Informacije* su podaci povezani u smislene cjeline odnosno. Informacije su protumačeni podaci koji primatelju donose novost čiju vrijednost on sam mora procijeniti. Pojam informacije bliži je koncepcijskome pogledu na informacijski sustav. Na fizičkoj razini, češće se govori o *podatku*, a na koncepcijskoj o *informaciji*. Dakle *podatak* je sačuvana (zapisana - materijalizirana) informacija.

---

[17] *entropija* … (grč. *en u, trope*) pretvaranje, *entropia* sadržaj pretvaranja; fiz. funkcija čija veličina služi kao mjera za vjerojatnost danoga stanja tijela ili sustava tijela; entropija tijela ili sustava tijela jest količnik iz zbroja povećanja unutarnje energije i vanjskoga izvršenoga rada pri širenju toga tijela i apsolutne temperature; pri svakome procesu u prirodi zbroj energija ostaje stalan, ali ta pretvorba energije iz jednoga oblika u drugi povećava zbroj entropija svih jedinki koje sudjeluju u procesu; u graničnome, zamišljenome slučaju za povratan proces, ovaj zbroj entropija ostaje nepromijenjen; u prirodi su svi procesi nepovratni, ireverzibilni; otuda, osim načela održanja energije, u prirodi vrijedi načelo povećanja entropije; prvo načelo je nužno za objašnjenje zbivanja u prirodi, ali nije dovoljno te ga u tu svrhu dopunjuje drugo načelo

[18] *semantika* … grana filologije u kojoj se proučava značenje riječi

Podaci prikazuju, odnosno opisuju promatran dio stvarnoga svijeta. Podaci najčešće ne mogu potpuno opisati stvaran svijet, ali ga mogu do određene razine dovoljno detaljno opisati. Podaci necjelovito opisuju objekte, događaje i njihova svojstva, ali korisniku dovoljno, jer odluku o tomu kako i kojim podacima opisati stvaran svijet donosi korisnik ovisno o svojim potrebama. Prikupljanje i pohrana podataka jest preduvjet ostvarivanja svrhe informacijskoga sustava ili nekoga njegovoga dijela.

Čovjek tijekom svoga života prihvaća mnogo podataka, ali ih ne pamti odvojeno. Razvrstava ih i stavlja u međusobne odnose (*relacije*). Isti koncept koristi se i u informacijskim sustavima. Zbog toga, potrebno je poznavanje i razumijevanje strukture podataka. To se razumijevanje olakšava i bilježi za buduću upotrebu, izradom *modela podataka*.

## 1.2. 1.2. Izvor, odredište, sadržaj informacije i mjera informacije

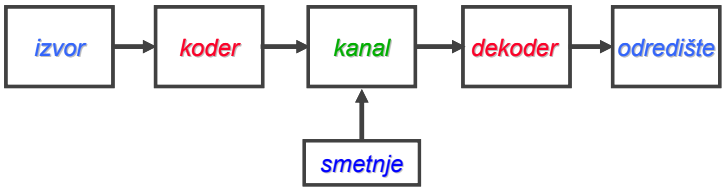
Osnovni model diskretnoga izvora informacije karakteriziraju[19] tri veličine:

- $x \in X = \{x_1, x_2, \ldots, x_m\}$, $X$ … skup od $m$ poruka
- $A = \{a_1, a_2, \ldots, a_n\}$, $A$ … abeceda izvora s $n$ znakova abecede
- $p = \{p(x_1), p(x_2), \ldots, p(x_o)\}$, $p$ … razdioba vjerojatnosti pojave određenih poruka $x_i$.


*Slika 1.1: Komunikacijski sustav*

### 1.2.1. 1.2.1. OSNOVE OPISA

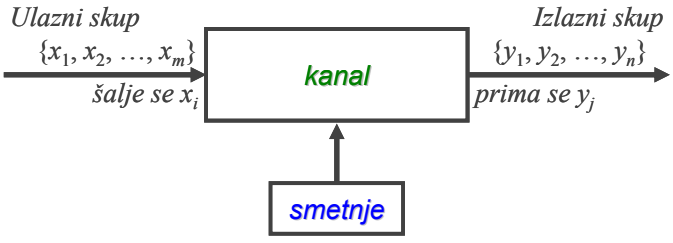
Ako saznamo za događaj, čiji nastup nije bio unaprijed određen, intuicijski možemo zaključiti da primamo informaciju. Određenost nastupa nekoga događaja može se mjeriti njegovom *vjerojatnošću*. Što je nastup događaja vjerojatniji, to ćemo manje informacije primiti poslije no što se on stvarno i dogodi. Nakon što smo na ovaj način *sadržajno* opisali informaciju, u sustavu na slici 1.2, odredit ćemo *količinske* odnose.




*Slika 1.2: Shannonov jednosmjeran komunikacijski kanal*

Na *ulazu* nekoga komunikacijskoga sustava nalazi se konačan skup mogućih *različitih* osnovnih simbola (npr., slova hrvatske abecede $\{A, B, C, \ldots, Ž\}$). Označimo *neki element* toga skupa kao:

$$x_i, \quad i = 1, 2, \ldots, m.$$




*Slika 1.3: Komunikacijski sustav kroz koji prolaze informacijski simboli*

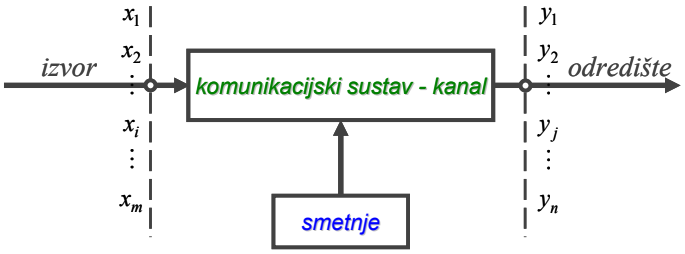
[19] *karakter* … (grč. *charasso*, *charakter*) urezujem, zbroj osobina ili svojstava kojima se neki čovjek, neka stvar ili pojava odlikuju; trajna osobina volje i načina djelovanja nekog čovjeka koja ga čini onim što je, tj. različitim od svih drugih ljudi; u užem, psihološkom smislu: trajnost, stalnost i dosljednost u volji, u načinu djelovanja; znak, obilježje, bitna oznaka, osobitost, osobita odlika, osobina, značajka, svojstvo; priroda, narav; osoba s dobrim osobinama duše i srca: položaj i čin, zanimanje, stalež

Kroz ulaz u komunikacijski sustav, cilj nam je propustiti *neki* (jedan određen) element

$$x_i,\ i = 1, 2, \ldots, m$$

ulaznoga skupa $X$. Zbog fizičkih svojstava komunikacijskoga sustava, ne očekuje se da će se nakon prolaza elementa $x_i$ kroz komunikacijski sustav, na izlazu pojaviti isti taj element $x_i$, nego neki *drugi* element također pripadnik ulaznoga skupa.

Napomena: Teorijski, moguće je da element nakon prolaza *kroz* kanal i pojave na *izlazu* kanala, ne pripada ulaznome skupu $\{x_1, x_2, \ldots, x_m\}$ već nekome proširenome skupu $\{x_1, x_2, \ldots, x_m, \ldots, x_p\}$, ali to nema trenutnoga značaja i to se u ovoj skripti neće razmatrati.[20]

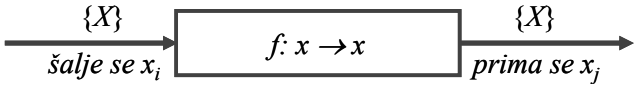



*Slika 1.4: Simboli izvora i odredišta*

Nakon prolaza *nekoga* elementa $x_i$ naznačenoga skupa kroz komunikacijski sustav, na *izlazu* toga sustava pojavljuje se općenito *neki element* $y_j$, *preslikan* iz ulaznoga skupa između SVIH elementarnih simbola $\{x_1, x_2, \ldots, x_m\}$. Označimo ga:

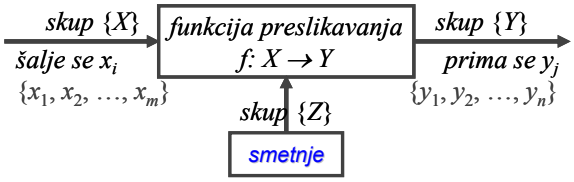
$$y_i, \qquad i = 1, 2, \ldots, n.$$

Ovako opisan *serijski* prolaz ulaznoga elementa $x_i$, kroz komunikacijski sustav, matematički se prikazuje kao na slici 1.5.




*Slika 1.5: Matematički prikaz prolaza elementa nekoga skupa kroz blok komunikacijskoga sustava*

Ako na komunikacijski sustav ne bi djelovale *smetnje*, onda se svaki element iz skupa ulaznih simbola $X$, bijekcijski preslikava u isti takav elementarni simbol izlaznoga skupa $Y \Rightarrow X$, tj. svaki element ulaznoga skupa $X$ preslikavao bi se u samoga sebe ali u izlaznome skupu $Y$. Međutim, na blok-komunikacijskoga sustava djeluju smetnje (slika 1.6).

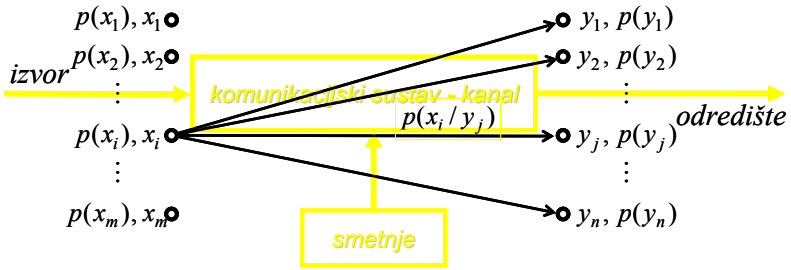



*Slika 1.6: Smetnje djeluju na blok-komunikacijskoga sustava*

Zbog utjecaja tih smetnji, a nakon prolaza kroz sustav, simbol na izlazu iz sustava, $x_i$, $i = 1, 2, \ldots, m$, ne mora se podudarati ni s jednim elementom ulaznoga skupa $X$, umjesto elementa $y_i \Rightarrow x_i$, pojavljuje se neki element izlaznoga skupa, $y_j$, $j = 1, 2, \ldots, n$, koji je različit od onoga preslika $X \Rightarrow Y$ dok nije bilo smetnji. Očito, ulazni skup $X$ različit je od izlaznoga skupa $Y$ ($m \neq n$), u najširemu smislu. Skicirajmo novonastalu situaciju[21] (slika 1.7).

[20] U binarnome sustavu, ova pojava značila bi da se od jedne binarne znamenke (ne kao simbola nego kao signala), nakon prolaza informacije kroz kanal, stvore dvije ili više znamenaka. Međutim, svi algoritmi usmjereni su uglavnom na pretvorbe jednoga binarnoga simbola (0 ili 1) u drugi i obrnuto.

[21] *situacija* … (lat. *situs*, *situatio*) položaj, stanje, položaj (kuće, mjesta); društveni položaj, imovinsko stanje; opće stanje

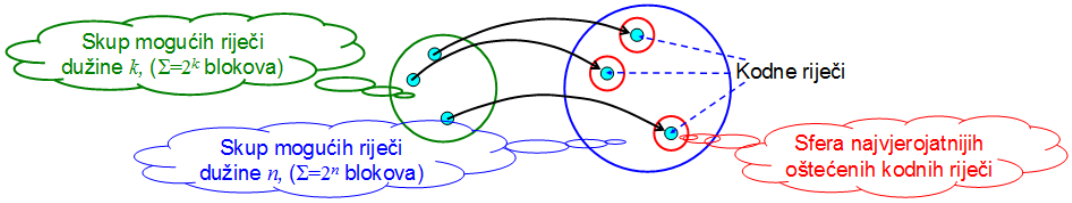



*Slika 1.7: Ulazna i izlazna abeceda*

Na primjer, pod simbolima $x_i$ možemo podrazumijevati simbole ("slova") abecede izvora informacija, a pod $y_j$ simbole kodirane informacije (slika 1.4):

- ulaz odnosno izlaz kodera informacije,
- ili da $x_i$ budu simboli kodirane informacije, a $y_j$ signali na izlazu kodera signala,
- ili $x_i$ signali na ulazu prijenosnoga sustava, a $y_j$ signali na izlazu prijenosnoga sustava
- itd.

Promatramo li postupak prijenosa informacija kroz odijeljene blokove sustava, u najjednostavnijemu slučaju, od $n$ simbola na ulazu bloka izabire se neki simbol $x_i$, za koji se na izlazu uočava simbol $y_j$. Na temelju primljenoga simbola $y_j$ potrebno je odlučiti, koji je simbol $x_i$ predan na ulazu (slika 1.8).




*Slika 1.8: Kodiranje i dekodiranje:Pridruživanje zalihosnih bitova s ciljem zaštite informacije od pogrešaka u kanalu.*

Veći dio informacija kojima se koristimo, izražava se posredstvom nekoga "jezika" u kojemu vladaju određene statističke zakonitosti. Ovdje ćemo razmatrati jednostavan slučaj, gdje se informacije predstavljaju pomoću $n$ simbola čije su *relativne frekvencije pojavljivanja* međusobno nezavisne i u potpunosti se određuju *apriornim vjerojatnostima* $p(x_i)$. Sasvim je razumljivo da je skup svih simbola potpun, iz čega proizlazi:

$$\sum_{i=1}^{n} p(x_i) = 1 . \qquad (1.1)$$

### 1.2.2. 1.2.2. OSNOVNI POJMOVI TEORIJE INFORMACIJA

- Na *ulazu* bloka odabire se neki simbol $x_i$,
- za koji se na *izlazu* uočava simbol $y_j$.

1. *Relativne frekvencije* ….

   Pojave simbola $x_i$, međusobno su nezavisne i potpuno su određene recipročnim odnosom ukupnoga broja raspoloživih simbola ulazne abecede. Na primjer, iz abecede hrvatskih slova {skup koji ima npr. 27 slova}, relativna frekvencija pojave svakoga pojedinoga slova (općenito) izražena je kao:

$$f_i = \frac{1}{27} .$$

   Uzmemo li u obzir ukupan broj slova, zbroj relativnih frekvencija SVIH slova je:

$$\sum_{i=1}^{i=27} f_i = 1 .$$

Izračunata relativna frekvencija pojave nekoga simbola između svih raspoloživih simbola ulazne abecede, za međusobno neovisne simbole ulaznoga skupa $X$, jednaka je …

2. … *apriornim vjerojatnostima* $p(x_i)$

Apriorne vjerojatnosti određuju relativne frekvencije, što za slučaj hrvatske abecede znači da je apriorna vjerojatnost pojave pojedinoga slova između 27 slova abecede:

$$p(x_i) = \frac{1}{27} = 0{,}\dot{0}3\dot{7}\,.$$

3. *Združene vjerojatnosti* $p(x_i, y_j)$ …

… opisuju *istovremenu* pojavu *parova* $(x_i, y_j)$ (**i** jedan **i** drugi ≡ oba) → logičko **&** → množenje. Prema tvrdnji o množenju vjerojatnosti:

$$p(x_i, y_j) = p(x_i) \cdot p(y_j / x_i) = p(y_j) \cdot p(x_i / y_j)\,. \tag{1.2}$$

4. *Uvjetna vjerojatnost* $p(y_j/x_i)$

Ako je poznat *fizikalni mehanizam pretvorbe* simbola $x_i$ u simbol $y_j$, može se izračunati *uvjetna vjerojatnost* $p(y_j/x_i)$, Ona opisuje pojavu simbola $y_j$ na *izlazu* promatranoga bloka, ako je poznato da je na *ulazu* poslan simbol $x_i$. Djelovanje smetnji i slučajnih izobličenja, uzrokuju ove pojave pa *uvjetna vjerojatnost* opisuje *stupanj djelovanja smetnji*.

Združene vjerojatnosti $p(x_i, y_j)$ mogu se odrediti jednadžbom (1.2) ako su poznate *apriorne vjerojatnosti* $p(x_i)$ i uvjetne vjerojatnosti. Uvjetne vjerojatnosti $p(y_j/x_i)$ određuju načini djelovanja smetnji i slučajnih izobličenja, a mogu se pribaviti sustavnim višestrukim mjerenjima. Za ovako definirane skupove simbola, vrijede relacije za *apriorne vjerojatnosti* $p(x_i)$ i $p(x_j)$:

$$p(x_i) = \sum_{j=1}^{n} p(x_i, y_j)\,, \qquad p(y_j) = \sum_{i=1}^{m} p(x_i, y_j)\,,$$
$$\sum_{i=1}^{m} p(x_i) = \sum_{j=1}^{n} p(y_j) = 1\,. \tag{1.3}$$

5. *Sadržaj informacije* $I$, …

… koji sa sobom nosi pojava događaja $y_j$ u odnosu na pojavu događaja $x_i$.

S motrišta promatrača na izlazu sustava, pojava točno određenoga ishoda $y_j$ svodi se na to, da prvobitnu nesigurnost u odnosu na generiranje događaja $x_i$ [karakteriziranoga *apriornom vjerojatnosti* $p(x_i)$], zamjenjuje nesigurnost koja ostaje nakon pojave $y_j$. Ovu *preostalu nesigurnost* u potpunosti opisuje *aposteriorna vjerojatnost* $p(x_i/y_j)$.

6. *Aposteriorna vjerojatnost* $p(x_i/y_j)$

Pojava događaja $y_j$ mijenja vjerojatnost pojave događaja $x_i$ iz *apriorne* $p(x_i)$ u *aposteriornu* $p(x_i/y_j)$. Aposteriornu vjerojatnost možemo izraziti koristeći relacije (1.2) i (1.3) kao:

$$p(x_i / y_j) = \frac{p(x_i, y_j)}{p(y_j)} = \frac{p(x_i, y_j)}{\sum_{i=1}^{m} p(x_i, y_j)}\,. \tag{1.4}$$

Dakle, postoji *fizikalna osnova* (smetnje i izobličenja) za tvrdnju da je ~~*sadržaj informacije I*, odnosno za sada~~ *aposteriorna vjerojatnost* $p(x_i/y_j)$, što se odnosi na događaj $x_i$, a statistički je povezana pojavom događaja $y_j$, funkcija samo odnosa *združene vjerojatnosti* $p(x_i, y_j)$ i *apriorne vjerojatnosti* $p(x_i)$. Zbog simetrije vrijedi:

$$p(y_j / x_i) = \frac{p(x_i, y_j)}{p(x_i)} = \frac{p(x_i, y_j)}{\sum_{j=1}^{n} p(x_i, y_j)}\,. \tag{1.5}$$

Isti zaključak vrijedi za sadržaj informacije što se odnosi na događaj $y_j$, a iskazuje da je pojava događaja $x_i$, samo funkcija odnosa vjerojatnosti $p(x_i/y_j)$ i $p(x_j)$. Ovime smo se približili količinskoj definiciji *sadržaja informacije I*.

7. *Uzajamni sadržaj informacije* $I(x_i; y_j)$

*Uzajamni sadržaj informacije* $I(x_i; y_j)$ odnosi se na događaj $x_i$, a dobije se pojavom događaja $y_j$ kao *logaritamski* odnos[22] *aposteriorne vjerojatnosti* $p(x_i/y_j)$ i *apriorne vjerojatnosti* $p(x_i)$:

$$I(x_i; y_j) = \log \frac{p(x_i / y_j)}{p(x_i)}. \tag{1.6}$$

Matematički izvod formule za *sadržaj informacije* temelji se na nizu postulata i neće se ni dokazivati niti obrazlagati.

### 1.2.3. 1.2.3. BÎT, NÎT, DÎT

Izbor baze logaritama određuje jedinicu sadržaja informacije. S motrišta tehnike, najprikladnija je primjena logaritama s bazom 2 (*dualni logaritam*) što je povezano primjenom binarnih digitalnih sustava u prijenosu i obradi informacija.

Jedinica za sadržaj informacije *I* jest *binarna jedinica* ili kraće [bît] (***bi**nary digi**t***). Za matematičke operacije informacijama, prikladno je koristiti se *prirodnim* logaritmima (bâza *e*). U tomu slučaju govori se o *prirodnim jedinicama* za sadržaj informacije [nît]. Primjeni li se baza 10, informacija se mjeri u *dekadskim jedinicama* [dît]. S načelnoga motrišta, izbor baze logaritama nema bitnoga značenja, budući da se iz logaritama po jednoj bazi, može preći u logaritme po drugim bazama poznatom relacijom:

$$\log_b k = \log_b a \cdot \log_a k. \tag{1.7}$$

Prirodna jedinica je veća 1,443 puta od binarne, a dekadska 3,322 puta. U nastavku, ako se posebno ne napomene, upotrijebit će se logaritmi baze 2 (*dual logarithms*).

### 1.2.4. 1.2.4. SLOŽENI POJMOVI TEORIJE INFORMACIJA

Pomnožimo li brojnik i nazivnik u izrazu (1.6) s $p(y_j)$ dobivamo:

$$I(x_i; y_j) = \mathrm{ld} \frac{p(x_i / y_j) \cdot p(y_j)}{p(x_i) \cdot p(y_j)} = \mathrm{ld} \frac{p(x_i, y_j)}{p(x_i) \cdot p(y_j)}. \tag{1.8}$$

8. *Uzajamni sadržaj informacije* $I(x_i; y_j)$ (nastavak)

Veličina $I(x_i; y_j)$ s informacijskoga motrišta, karakterizira *statističku vezu* (ovisnost) između događaja $x_i$ i $y_j$.

$$I(x_i; y_j) = I(y_j; x_i) \tag{1.9}$$

Informacija što ju donosi pojava događaja $y_j$, a koja opisuje događaj $x_i$, jednaka je informaciji koja opisuje događaj $x_i$, u odnosu na $y_j$ i *simetrična* je funkcija u međusobnim odnosima događaja $x_i$ i $y_j$. To znači da je sasvim svejedno promatramo li pojave simbola s ulaza prema izlazu ili obrnuto. Prema tomu, mjeru *sadržaja informacije* koju smo uveli prema izrazu (1.6), možemo nazvati *uzajamnim sadržajem informacije* dvaju slučajnih događaja, jedan u odnosu prema drugome. Ako su događaji $x_i$ i $y_j$ međusobno nezavisni onda je:

$$p(x_i, y_j) = p(x_i) \cdot p(y_j),$$

iz čega slijedi

$$I(x_i; y_j) = \mathrm{ld} \frac{p(x_i, y_j)}{p(x_i) \cdot p(y_j)} = \mathrm{ld} \frac{p(x_i, y_j)}{p(x_i, y_j)}, \tag{1.10}$$

[22] Složen matematički postupak!

dakle:

$$I(x_i; y_j) = 0, \tag{1.11}$$

odnosno opisom: Ako ne postoji nikakva statistička povezanost između dvaju događaja $x_i$, $y_j$, onda pojava jednoga neće donijeti nikakve informacije o drugome, pa je prirodno da uzajamna informacija bude jednaka nuli.

9. *Vlastiti sadržaj informacije* $I(x_i)$ događaja $x_i$

Veličina $I(x_i)$ predstavlja sadržaj informacije što ga donosi sam događaj $x_i$ ili bilo koji drugi događaj potpuno jednoznačno njime povezan. Iz izraza (1.6) slijedi, da će uz čvrst iznos vjerojatnosti $p(x_i)$, *uzajamna informacija* $I(x_i; y_j)$ imati maksimalnu vrijednost, ako je $p(x_i/y_j) = 1$ tj., ako $y_j$ sigurno i jednoznačno određuje $x_i$. Zbog statističke nezavisnosti, brojnik u izrazu (1.6) jednak je nuli pa je maksimalna vrijednost vlastitoga sadržaja informacije:

$$I(x_i) = -\mathrm{ld}\, p(x_i) \tag{1.12}$$

Razumljivo je da će neki događaj svojom pojavom, sam o sebi donijeti najviše informacije nego bilo koji drugi koji je njime statistički povezan. Prema tomu …

10. … *uzajamni sadržaj informacije između dvaju događaja*, …

… nikada nije veći od *vlastitoga sadržaja informacije* svakoga od njih, tj.

$$I(x_i; y_j) \le I(x_i)\,, \qquad I(x_i; y_j) \le I(y_j)\,. \tag{1.13}$$

Napomenimo da je *vlastiti* sadržaj *informacije* uvijek pozitivna veličina, jer je vjerojatnost:

$$0 \le p(x_i) \le 1.$$

*Uzajamni sadržaj informacije* može imati kako *pozitivnu* tako i *negativnu* vrijednost.

(a) Pozitivnu vrijednost ima u slučaju, ako je vjerojatnost $p(x_i, y_j)$ zajedničkoga nastupa *para zavisnih* događaja $x_i$ i $y_j$, veća od umnoška njihovih *bezuvjetnih vjerojatnosti* $p(x_i)\cdot p(y_j)$ (što u stvari predstavlja *vjerojatnost zajedničkoga nastupa*, ako bi događaji bili *nezavisni*).

(b) U obrnutomu slučaju, *uzajamni sadržaj informacije* ima negativnu vrijednost.

### 1.2.5. 1.2.5. VRSTE SADRŽAJA INFORMACIJE

Pretpostavimo nadalje, da događaj $x_i$ nije statistički povezan samo događajem $y_j$ već i nekim trećim događajem $z_k$, $k = 1, 2, \ldots, l$, pri čemu su poznate vjerojatnosti $p(x_i, y_j\, z_k)$. Zanima nas koliki je *sadržaj informacije* što ga donosi događaj $y_j$, a koji se odnosi na događaj $x_i$, ako je prethodno poznat događaj $z_k$. Sadržaj informacije $I(x_i; y_j/z_k)$ koji odgovara ovakvoj situaciji nazivamo …

11. … *uvjetni uzajamni sadržaj informacije*

Nakon pojave događaja (koji se događa prije $y_j$), spoznaja što se odnosi na događaj $x_i$, mijenja se od apriorne vjerojatnosti $p(x_i)$ u uvjetnu vjerojatnost $p(x_i/z_k)$. Nakon pojave događaja $y_j$ ta se vjerojatnost opet mijenja te se spoznajom o događaju $x_i$ određuje aposteriorna vjerojatnosti $p(x_i/y_j\cdot z_k)$. Drugim riječima u razmatranoj situaciji (uz prethodnu pojavu događaja $z_k$) početna nesigurnost u odnosu na nastup događaja $x_i$ karakterizirana je uvjetnom vjerojatnosti $p(x_i/z_k)$, a konačna aposteriornom vjerojatnosti $p(x_i/y_j\cdot z_k)$. Prema tomu logičkim poopćenjem izraza (1.6), dolazimo do izraza za *uvjetni uzajamni sadržaj informacije*:

$$I(x_i; y_j / z_k) = \mathrm{ld}\,\frac{p(x_i / y_j \cdot z_k)}{p(x_i / z_k)} = \mathrm{ld}\,\frac{p(x_i, y_j / z_k)}{p(x_i / z_k)\cdot p(y_j / z_k)}\,. \tag{1.14}$$

Izraz (1.14) možemo poopćiti i za slučajeve ako je prethodno poznato više događaja, a ne samo jedan.

12. *Sadržaj informacije zajedničke pojave događaja*

Sadržaj informacije što ga donosi zajednička pojava događaja $y_j$ i $z_k$, a odnosi se na događaj $x_i$, jednak je zbroju sadržaja informacije događaja $y_j$ u odnosu na događaj $x_i$ i događaja $z_k$ u odnosu na $x_i$ uz poznati $y_j$.

$$I(x_i; y_j \cdot z_k) = I(x_i; y_j) + I(x_i; z_k / y_j)\,. \tag{1.15}$$

Taj rezultat podudara se našom intuicijskom predodžbom o informaciji.

13. *Uzajamni sadržaj informacije složenih događaja*

$$\begin{aligned} I(x_i u_\alpha; y_j v_\beta) &= I(x_i u_\alpha; y_j) + I(x_i u_\alpha; v_\beta / y_j) = I(x_i; y_j) + I(x_i; v_\beta / y_j) + \\ &+ I(u_\alpha; y_j / x_i) + I(u_\alpha; v_\beta / x_i \cdot y_j) \end{aligned}\,. \tag{1.16}$$

U slučaju statistički nezavisnih događaja, na desnoj strani izraza (1.16), drugi i treći član postaju jednaki nuli, a *uvjetan sadržaj informacije* izražen posljednjim članom, prelazi u *bezuvjetan sadržaj informacije*. Prema tomu, za nezavisne događaje iz izraza (1.16) slijedi *adicijsko svojstvo* uzajamnoga sadržaja informacije:

$$I(x_i u_\alpha; y_j v_\beta) = I(x_i; y_j) + I(u_\alpha; v_\beta)\,. \tag{1.17}$$

Za slikovit prikaz izraza (1.17) možemo razmatrati $x_i$ i $u_\alpha$ kao nezavisne simbole na ulazu dvaju odijeljenih kanala, a $y_j$ i $v_\beta$ kao odgovarajuće simbole na izlazu spomenutih kanala (slika 1.9).


*Slika 1.9: Statistička zavisnost dvaju kanala*

Sadržaj informacije, primljen pojavom para izlaznih simbola što se odnosi na par ulaznih simbola, jednak je zbroju sadržaja informacije, što ga donosi svaki izlazni simbol o svomu ulaznomu paru.

Prema analogiji s *uvjetnim uzajamnim sadržajem informacije* možemo uvesti i …

14. *… uvjetni vlastiti sadržaj informacije*

Uvjetni vlastiti sadržaj informacije $I(x_i/z_k)$ događaja $x_i$ ako je poznat događaj $z_k$, određuje se prema izrazu:

$$I(x_i / z_k) = -\mathrm{ld}\ p(x_i / z_k)\,. \tag{1.18}$$

Uvjetni vlastiti sadržaj informacije može se predstaviti na dva načina:

- kao maksimalan sadržaj informacije što ga može donijeti pojava događaja $x_i$ uz poznat događaj $z_k$,
- kao sadržaj informacije što ga mora izraziti neki drugi događaj

tako da jednoznačno odredi događaj $x_i$, uz poznat $z_k$.

15. *Uzajamni sadržaj informacije izražen pomoću vlastitih sadržaja informacije je*:

$$I(x_i; y_j) = I(x_i) - I(x_i / y_j) = I(y_j) - I(y_j / x_i)\,. \tag{1.19}$$

Iz relacije (1.8) također možemo izvesti da je:

$$I(x_i; y_j) = I(x_i) + I(y_j) - I(x_i, y_j) \tag{1.20}$$

gdje je $I(x_i, y_j)$ …

16. … *vlastiti sadržaj informacije zajedničkoga nastupa para događaja* $x_i \cdot y_j$ koji iznosi:

$$I(x_i, y_j) = -\text{ld}\ p(x_i, y_j)\,. \tag{1.21}$$

Ako izraz (1.20) drugačije napiše, tj. članovi $I(x_i; y_j)$ i $I(x_i, y_j)$ međusobno zamijene mjesta, izlazi:

$$I(x_i, y_j) = I(x_i) + I(y_j) - I(x_i; y_j). \tag{1.22}$$

Izrazi (1.20) odnosno (1.21) opisuju da je …

17. … *sadržaj informacije složenoga događaja* $(x_i, y_j)$,

ako se dogode oba događaja i $x_i$ i $y_j$, jednak je ***zbroju*** *sadržaja informacije* potreban za odrediti događaje $x_i$ i $y_j$ ***umanjen*** za *uzajamni sadržaj informacije* između događaja $x_i$ i $y_j$.

### 1.2.6. 1.2.6. MJERA KAKVOĆE PRIJEMNOGA UREĐAJA

U određenim slučajevima komunikacijskih sustava, *sadržaj informacije* odnosi se *samo* na *pojavu odijeljenih znakova* i ne daje informaciju o *sustavu kao cjelini*. Zbog toga je potrebno uvesti *veličine* koje *cjelovito opisuju svojstva sustava*. Ti podaci dobiju se uvođenjem *prosječnih* (*srednjih*) *sadržaja informacije* za razne situacije, koje se odnose na čitav skup događaja. *Sadržaj informacije* što ga donosi pojedini događaj, možemo smatrati *slučajnom veličinom*, a pojavljuje se *istom vjerojatnošću* kao i sam događaj. Dakle, sadržaj informacije i pripadajuće vjerojatnosti čine *statistički skup*, za koji se mogu izračunati *prosjeci*.

Najprije odredimo …

18. … *srednji iznos uzajamnoga sadržaja informacije* $I(x_i; y_j)$,

koji se odnosi na čitav skup događaja $X = \{x_i\}$, $i = 1, 2, \ldots, m$, a njega unosi pojava nekoga događaja $y_j$.

| $i$ | 1 | 2 | … | $m$ |
|---|---|---|---|---|
| $I(x_i; y_j)$ | $I(x_1; y_j)$ | $I(x_2; y_j)$ | … | $I(x_m; y_j)$ |
| $p[I(x_i; y_j)] = p(x_i/y_j)$ | $p(x_1/y_j)$ | $p(x_2/y_j)$ | … | $p(x_m/y_j)$ |

Tablica prikazuje pojedine vrijednosti sadržaja informacije $I(x_i; y_j)$ i odgovarajuće vjerojatnosti kojima se one pojavljuje. Tražen prosječan iznos može se izračunati kao:

$$I(X; y_j) = \sum_{i=1}^{m} I(x_i; y_j)\, p(x_i / y_j) = \sum_{i=1}^{m} p(x_i / y_j)\ \text{ld}\ \frac{p(x_i / y_j)}{p(x_i)}\,. \tag{1.23}$$

Veličinu $I(X; y_j)$ opisuje …

19. … *srednji sadržaj informacije*

što ga donosi primljen simbol $y_j$, a odnosi se na skup *svih* poslanih *simbola* iz skupa $X$. Ta veličina uvijek je pozitivna tj.

$$I(X; y_j) \geq 0\,. \tag{1.24}$$

Ako smatramo da skup $X$ predstavlja bilo koji simbol što ga se može poslati, onda nejednadžba (1.24) opisuje da neki primljen simbol $y_j$, uvijek donosi pozitivan sadržaj informacije o mogućim poslanim simbolima. To nam može poslužiti kao …

20. … *mjera kakvoće prijemnoga uređaja*.

Ako je taj sadržaj informacije *veći*, prijemni uređaj je *bolji*. Analognim postupkom možemo izračunati …

21. … *srednje vrijednosti uzajamnoga sadržaja informacije*

za skup $Y = \{y_j\}, j = 1, 2, \ldots, n$, uz učvršćen događaj $x_i$:

$$I(x_i;Y)=\sum_{j=1}^{n} p(y_j/x_i)\,\mathrm{ld}\,\frac{p(y_j/x_i)}{p(y_j)}. \tag{1.25}$$

Odredimo dalje …

22. … *potpun srednji uzajamni sadržaj informacije* …

… u skupu događaja *Y* što se odnosi na skup događaja *X*, izrazom:

$$I(X;Y)=\sum_{i=1}^{m}\sum_{j=1}^{n} p(x_i,y_j)\,\mathrm{ld}\,\frac{p(x_i/y_j)}{p(x_i)}. \tag{1.26}$$

I gornji izraz zadovoljava uvjet (1.24). Veličina $I(X; Y)$ karakterizira *sadržaj informacije* što se u prosjeku sadrži u skupu primljenih simbola, a odnosi se na skup poslanih simbola i to prije slanja bilo kojega određenoga simbola.

Sada je potpuno razumljivo, da će nas u praksi najviše zanimati tako definiran *potpun srednji uzajamni sadržaj informacije* $I(X; Y)$, a ne *uzajamni sadržaj informacije* $I(x_i; y_j)$ što se odnosi samo na određene elementarne simbole odnosno elementarne događaje.

# 1. *Auditorna vježba 1: Statistička svojstva jezika*

Napomena: Cjelovit opis nalazi se na sustavu za podršku nastavi!

## 1.3. 1.3. Entropija

### 1.3.1. 1.3.1. INFORMACIJSKA MJERA

Osnovni zadatak komunikacijskoga sustava je učinkovit prijenos raznih informacija od izvora do odredišta, a znači predaju što većega broja istinitih podataka u zadanome vremenu kroz komunikacijski sustav. U *Matematičkoj teoriji komuniciranja*,[23] Shannon je izradio matematičke modele za opis osnovnih pojava u komunikacijskim sustavima. Odredio je mjeru za informaciju i parametre za količinski opis komunikacijskoga sustava. Upozorio je na moguće načine poboljšanja rada komunikacijskih sustava te je dao njegov shematski prikaz (poglavlje 0, slika 0.2).

Da bi se mogli usporediti različiti komunikacijski sustavi prema učinkovitosti, nužno je ustanoviti zajedničku mjeru, koja brojčano ocjenjuje volumen poslanih podataka i sposobnost sustava da te podatke prenese. Istražujući kako matematički definirati mjeru analize informacijskih sustava, H. Nyquist i R. Hartley zaključili su da mjera što ju "nosi" pojedini signal mora imati *logaritamsko svojstvo*. Hartley je predložio da se signalu, kojega ga se bira iz skupa od $n$ mogućih signala, pridruži informacija

$$I(n) = \log n. \tag{1.27}$$

Osnovni razlog za tako definiranu mjeru za informaciju, Hartley je našao u činjenici da je informacija koju nosi poruka, sastavljena od dvaju signala. Pri tomu se jedan bira iz skupa od $m$, a drugi iz skupa od $n$ mogućih signala pa je mjera jednaka zbroju informacija što ju nose pojedini signali. Budući da takvih uređenih parova signala ima $m{\cdot}n$, pojedini uređen par signala nosi informaciju $I(m{\cdot}n)$, tako da naveden zahtjev glasi

$$I(m{\cdot}n) = I(m) + I(n) \tag{1.28}$$

pa se odmah nameće ideja da se za $I$ uzme logaritamska funkcija

$$\log(m{\cdot}n) = \log m + \log n.$$

Uzimanje veličine $\log n$ za mjeru informacije može se potkrijepiti i ovim tvrdnjama:

1. $\log n > 0$, za svaki prirodan broj $n$;
2. $m < n \Rightarrow \log m < \log n$.

Izlazi da je informacija uvijek pozitivna veličina. Dakle, prijemom jednoga od $n$ mogućih signala, uklanja se određena neizvjesnost koja je postojala prije prijema signala. Prema tomu, $\log n$ može se predstaviti i kao *mjera neizvjesnosti* povezana odabirom jednoga od $n$ mogućih izbora. Intuicijski, veoma je prihvatljiva osobina da većemu broju izbora, pripada i veća neizvjesnost.

Glavni nedostatak Hartleyeve mjere informacije jest u tome, što se sve poruke iz skupa svih mogućih poruka promatraju ravnopravno (jednako vjerojatno), a to nije u skladu s fizikalnom stvarnošću. Shannon je uočio da signali i poruke, kao i šumovi u komunikacijskim sustavima, imaju statistička svojstva pa je razumno pretpostaviti da se pojavljuju u skladu s određenim razdiobama vjerojatnosti.

On je uzeo da signal koji se pojavljuje vjerojatnošću $p_i > 0$, nosi informaciju $I(p) = \log(1/p_i) > 0$. Ako ima $n$ mogućih signala $\sum_{i=1}^{n} p_i = 1$ onda je srednja (prosječna) količina informacije koju nosi pojedini signal:

$$H(p_1, \ldots, p_n) = \Sigma\, p_i\, I(p_i) = -\,\Sigma\, p_i \log p_i. \tag{1.29}$$

Jasno, Hartleyeva definicija informacije, poseban je slučaj Shannonove definicije informacije uz $p_i = 1/n$. Shannonova definicija informacije uzima u obzir činjenica da se signali pojavljuju u skladu s određenom razdiobom vjerojatnosti $(p_1, \ldots, p_n)$, pri čemu je $p_i > 0$ i $\sum p_i = 1$, tako da srednja informacija po signalu ovisi jedino o toj razdiobi vjerojatnosti, a ne o nekim drugim veličinama koje su

[23] *A Mathematical Theory of Communication*, Bell System Tech. J. 27

svojstvene signalima. Zato je Shannon i nazvao veličinu $H(p_i, \ldots, p_n)$ ***entropija***[24] *konačne razdiobe vjerojatnosti*, jer se ona može predstaviti i kao neizvjesnost u pogledu slučajnoga izbora jedne od $n$ mogućih vrijednosti kojima pripadaju vjerojatnosti $p_1, \ldots, p_n$. Navedimo neka jednostavna svojstva što ih ima Shannonova entropija (slika 1.10), a koja će još više potvrditi njezinu prikladnost kao mjeru informacijske neizvjesnosti:

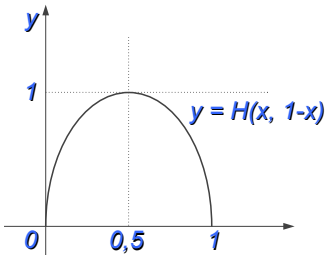
(I) $H(1) = 0$

(II) $H(p_1, \ldots, p_n)$ ne ovisi o permutaciji[25] vjerojatnosti, $p_1, \ldots, p_n$

(III) $H(p_1, \ldots, p_n, 0) = H(p_1, \ldots, p_n)$ (uz dogovor $0 \cdot \log 0 = 0$).

(IV) $H(1/n, \ldots, 1/n) = h(n)$, rastuća je funkcija od $n \in N$.

(V) $H(p, 1-p)$, neprekinuta je funkcija za $0 < p < 1$




*Slika 1.10: Krivulja entropije za binaran sustav uz vjerojatnosti p i (1–p)*

(VI) Ako je:

$$1 \le r \le n,\ q_1 = p_1 + \ldots + p_r \text{ i } q_2 = p_{r+1} + \ldots + p_n,$$

onda je:

$$H(p_1, \ldots, p_n) = H(q_1, q_2) + q_1 H(p_1/q_1, \ldots, p_r/q_1) + q_2 H(p_{r+1}/q_2, \ldots, p_n/q_2). \quad (1.30)$$

Ovom posljednjom relacijom iskazuje se da je neizvjesnost $H(p_1, \ldots, p_n)$, u pogledu izbora jednoga od elemenata iz skupa $\{x_1, \ldots, x_r, x_{r+1}, \ldots, x_n\}$, jednaka neizvjesnosti u pogledu izbora jednoga od skupova $Y_1$ ili $Y_2$, uvećanoj za zbroj otežanih (*ponderiranih*) neizvjesnosti izbora jednoga elementa iz skupa $Y_1$, odnosno skupa $Y_2$.

Nakon tih spoznaja o Shannonovoj entropiji, učvršćuje se uvjerenje da je ona prikladna mjera za neizvjesnost situacije opisane pomoću konačne razdiobe vjerojatnosti. Prirodno se nameće i pitanje, je li navedenim svojstvima (I) - (VI) sasvim opisana *mjera neizvjesnosti*, odnosno je li Shannonova entropija *jedina moguća mjera neizvjesnosti* navedenih svojstava.

Odgovor na to pitanje sadržan je u tzv. *teoremu jedinstvenosti*, koji ističe upravo ona svojstva pomoću kojih se može dokazati da je mjera za neizvjesnost, kao određena funkcija prirodnoga broja $n$ i konačne razdiobe vjerojatnosti $(p_1, \ldots, p_n)$, $(p_i \ge 0,\ \sum p_i = 1)$, upravo Shannonova entropija, uz logaritamsku bazu jednaku 2 (*dual logarithms*). To je povezano primjenom binarnih digitalnih sustava u prijenosu, preradi i zaštiti informacija. Prema tomu, binarna jedinica [bît], mjerna je jedinica sadržaja informacije.

### 1.3.2. 1.3.2. ZAKLJUČCI O ENTROPIJI

Mjera "sadržaj informacije" jest opće količinsko svojstvo nezavisno o određenoj fizikalnoj prirodi vijesti. Uvođenje pojma *sadržaja informacije* može se iznijeti u dvije osnovne tvrdnje (teorema):

1. Broj binarnih znamenaka koji je u prosjeku potreban da se predstavi bilo kakva vijest, jednak je sadržaju informacije što ju ta vijest posjeduje.

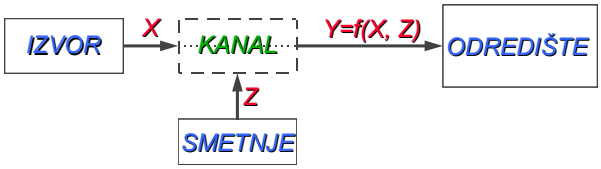
---

[24] Već je R. Boltzman (1844-1906) upotrebljavao izraz: $H = -k\sum_i p_i \log p_i$, $p_i \ge 0$, $H = \sum_i \log p_i = 1$, ($k$ … Boltzmanova konstanta), koji je nazvao *entropijom* idealnoga plina, za opisivanje stanja idealnoga plina čija se molekula vjerojatnošću $p_i$ nalazi u $i$-tome dijelu faznoga prostora. Za spontano odvijanje fizikalnih procesa karakteristično je povećanje entropije.

[25] *permutacija* … (lat. *permutatio*) promjena, razmjena, razmjenjivanje, zamjenjivanje; mat. premještanje, mijenjanje mjesta u nizu određenoga broja danih elemenata

2. Moguće je kodirati i dekodirati signale tako, da vjerojatnost pogrešnoga prijema predane informacije bude po volji mala.

Postupak odašiljanja i prijenosa informacije možemo shematski ovako opisati (slika 1.11):




*Slika 1.11: Odašiljanje i prijenos informacija*

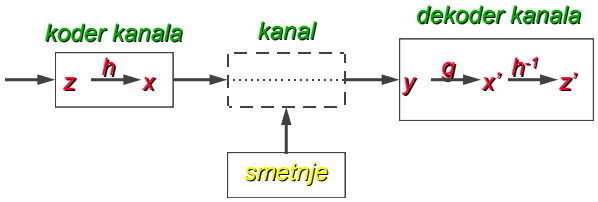
Izvor šalje signal $X$ (slučajna varijabla definirana na skupu svih mogućih stanja izvora informacije) koji se u kanalu "miješa šumom" $Z$ (slučajna varijabla) pa primatelj prima signal $Y$ (slučajna varijabla ili funkcija varijabli $X$ i $Z$). Zato je uobičajeno da se definira *vlastita informacija* $I(X)$ što ju "nosi" signal $X$ i *uzajamna informacija* $I(X, Y)$ što ju primatelj dobiva o $X$ kada primi $Y$. Neka je $X$ diskretna slučajna varijabla u skupu vrijednosti $R(x) = \{x_1, x_2, \ldots\}$ s vjerojatnostima $p_i = P(X{=}x_i) \geq 0$, $\sum_i p_i = 1$.

Ako je varijabla $X$ signal što ga šalje dani izvor informacije u određenomu trenutku, onda kažemo da se signal $x_i$ šalje vjerojatnošću $p_i$ i da se pri tomu generira informacija $I(x_i)$. Slučajna varijabla $I(X)$ zove se *vlastita informacija slučajne varijable* $X$. Promotrimo li očekivanje slučajne varijable $I(X)$, odmah vidimo da je:

$$E[I(x)] = -\sum_i I(x_i)p_i = -\sum_i p_i \log_2 p_i = H(X). \tag{1.31}$$

Dakle, očekivana ili srednja vrijednost $E[I(x)]$ vlastite informacije $I(X)$ slučajne varijable $X$, isto je što i *entropija slučajne varijable* $X$. Entropija je mjera neizvjesnosti stanja sustava koji svoja stanja poprima u skladu s danom razdiobom vjerojatnosti. Možemo reći da je $H(X)$ u izvoru informacije, mjera za *neizvjesnost odašiljanja* signala. Konačno $E[I(x)]$ je prosječna informacija što ju nosi signal odaslan iz toga izvora.

Intuicijski je posve prihvatljivo da se te dvije veličine podudaraju, jer generiranjem signala "nestaje neizvjesnost" koja je postojala prije odašiljanja signala. Ako, uz $p = 1$, očekujemo generiranje određenoga signala, neizvjesnosti u vezi emitiranja poruke nema - entropija je nula kao i količina informacije. Koder i dekoder kanala vezani su usko uz sam kanal odnosno sustav modulacije. Postupke kodiranja i dekodiranja uz komunikacijski kanal možemo opisati odgovarajućim funkcijama (vidi sliku 1.12):




*Slika 1.12: Komunikacijski kanal opisan odgovarajućim funkcijama*

### 1.3.3. 1.3.3. KODIRANJE I DEKODIRANJE

Kao nositelji informacije u svim fazama komuniciranja, javljaju se određeni $n$-struki nizovi simbola koji se podvrgavaju postupcima *kodiranja* i *dekodiranja*. Uobičajeno je da se takvi kodovi nazivaju blok-kodovi. Govori se o blok-kodu duljine $n$. Pri ustroju kodera i dekodera kanala, postoji mogućnost izbora prirodnoga broja $n$. Važna veličina jest *koeficijent prijenosa* ili brzina blok-koda:

$$r = \frac{\ln k}{n}. \tag{1.32}$$

**Definicija 1.1:** Ako su zadani prirodni brojevi:

$$k \text{ i } n \geq \frac{\ln k}{\ln n} \text{ i ako je odabrano } k \text{ različitih elemenata } \{x_1, \ldots, x_k\} \in U^n, \tag{1.33}$$

onda kažemo da je zadan *koder kanala* ili *blok-kod* duljine $n$, a označavamo ga $(n, k)$ kôd. Broj svih mogućih poruka $k$ koje dolaze u koder kanala $r$, možemo razumjeti kao maksimalnu količinu informacije koja otpada na jedan signal u $n$-članome nizu $x \in U^n$. Poželjno je da $r$ bude što veći, jer se onda informacija brže prenosi kroz zadani komunikacijski kanal. Vjerojatnost pogreške pri dekodiranju, ključan je pojam u ovomu području, a odnosi se na uvjetnu vjerojatnost

$$P(y \in S_i/x_i) \; (i = 1, 2, \ldots, k),$$

da se na izlazu kanala dobije niz $y$ koji pripada skupu $S_i \subseteq V^n$, ako je izlazni niz $x_i$ ($x_i \in M \subseteq U^n$). U tomu slučaju, uz primjenu dekodiranja pomoću dane funkcije $g$, imamo prijenos poruke $z_i$ bez pogreške, pa je:

$$P(y \notin S_i/x_i) = 1 - P(y \in S_i/x_i) = P(E/z_i), \quad i - 1, \ldots, k,$$

vjerojatnost pogreške pri prijenosu poruke $z_i$. Funkcije $g$, $h$, …, biraju se tako da pogrešnoga dekodiranja bude što manje. Ako stavimo da je:

$$\max P(E/z_i) = \varepsilon \qquad\qquad 1 \le i \le k,$$

onda parametar $\varepsilon$ ($0 \le \varepsilon \le 1$) karakterizira koder i dekoder u smislu pouzdanosti prijenosa informacije. Jedan od glavnih problema sastoji se upravo u tomu da se nađu uvjeti što ih moraju zadovoljiti koder i dekoder tako da "signal prenese kroz kanal" svu informaciju koju je preuzeo od izvora informacije i to vjerojatnošću što bližoj jedinici.

U temeljnoj tvrdnji o diskretnomu kanalu bez memorije, kapaciteta $C > 0$, dokazuje se da je uz uvjet $r < C$, moguće ustrojiti koder i dekoder tako da, ako $n \to \infty$, $\varepsilon$ eksponencijski teži nuli. Nemoguće je ustrojiti blok-kod s više od $k = e^{nC}$ kodnih riječi i odgovarajuću shemu odlučivanja za koju bi maksimalna vjerojatnost pogreške težila nuli, ako $n$ teži u beskonačnost. Shema odlučivanja ili *koder kanala*, svaka je funkcija $g$, koja *bijekcijski* preslikava vektorski prostor $V^n$ u skup raspoloživih poruka $M$:

$$g\colon V^n \to M = \{x_1, \ldots, x_k\}. \tag{1.34}$$

**Definicija 1.2:** Svaka funkcija $g\colon V^n \to M = \{x_1, \ldots, x_k\}$ zove se *dekoder* kanala ili *shema odluke*.

### 1.3.4. 1.3.4. INFORMACIJSKI KAPACITET ABECEDE

Razmotrimo događaje $x_i$ kao elementarne simbole abecede, a u slučaju da se svi simboli pojavljuju jednakim vjerojatnostima. Nejednadžba $H(X) \le \mathrm{ld}\, n$ pokazuje, da je *sadržaj informacije*, što ga u prosjeku donosi jedan simbol abecede, maksimalan i jednak dualnomu logaritmu broja simbola abecede. Ako na taj način promatramo situaciju, možemo veličinu $H_m(X)$ nazvati *informacijskim kapacitetom abecede*, što predstavlja maksimalan prosječan sadržaj informacije, koji se može dobiti pomoću neke abecede. U najjednostavnijemu slučaju, ako se abeceda sastoji od dvaju znakova, informacijski kapacitet takve abecede bit će $\mathrm{ld}\, 2 = 1$ [bît].

### 1.3.5. 1.3.5. UVJETNE ENTROPIJE

Na temelju sljedećih izraza od (1.35) do (1.37) mogu se prepoznati tri različita smisla srednjega uzajamnoga sadržaja informacije.

1. Iz izraza

$$I(X;Y) = H(X) + H(Y) - H(X,Y), \tag{1.35}$$

proizlazi da je *srednji uzajamni sadržaj informacije* jednak *razlici* između *srednjega sadržaja informacije potrebnoga za određivanje skupova događaja* $X$ i $Y$ odvojeno (kao da su oni statistički nezavisni) i *srednjega sadržaja informacije potrebnoga za određivanje skupa parova događaja* $XY$.

2. Iz izraza

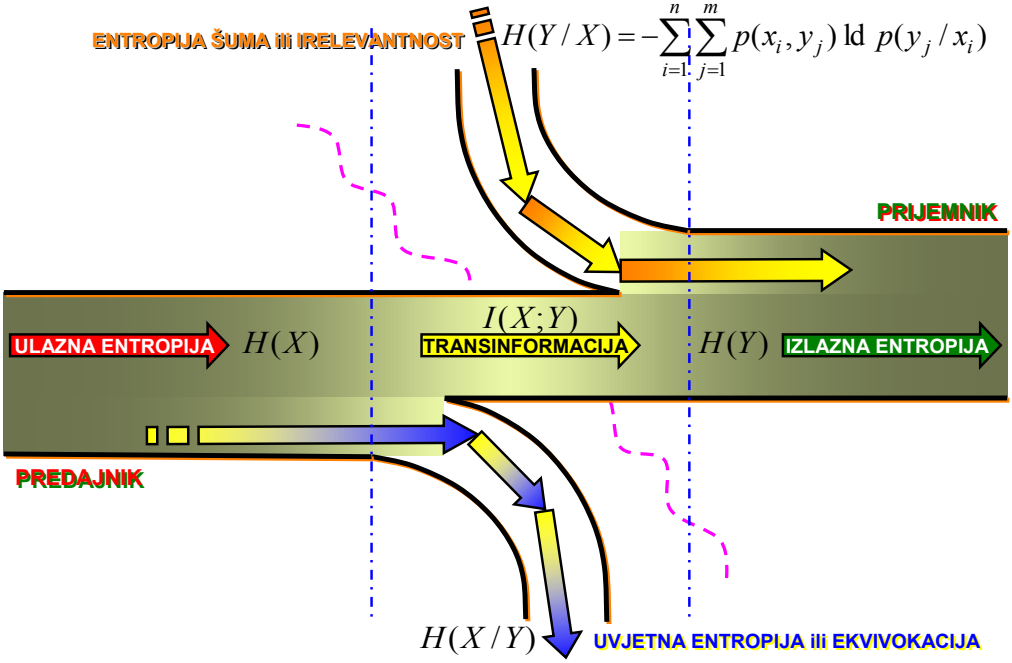
$$H(X \cdot Y) = H(X) + H(Y/X), \tag{1.36}$$

slijedi da za nezavisne događaje vrijedi $H(XY) = H(X) + H(Y)$, što ima za posljedicu da je srednji uzajamni sadržaj informacije jednak nuli, $I(X; Y) = 0$. Prema tomu ovakvo tumačenje znači, da veličina $I(X; Y)$ predstavlja mjeru *statističke veze među skupovima događaja* $X$ i $Y$.

$$I(X;Y) = H(X) - H(X/Y) \tag{1.37}$$

$$I(X;Y) = H(Y) - H(Y/X) \tag{1.38}$$

3. Za tumačenje *srednjega uzajamnoga sadržaja informacije* veoma su važni izrazi (1.37) i (1.38). Razmatramo li događaje $x_i$ kao elementarne simbole informacije na *predajnoj* strani komunikacijskoga sustava uz djelovanje smetnji (slika 1.13), a $y_j$ kao elementarne simbole na *prijemnoj* strani, spomenuti izrazi pokazuju fizikalnu prirodu prijenosa informacija.




*Slika 1.13: Informacijski tokovi*

*1.3.5.1.* *1.3.5.1. Transinformacija*

Veličina $I(X; Y)$ ovdje jasno poprima smisao *transinformacije* $H_T$, jer je to upravo

- onaj dio sadržaja informacije koji generira izvor (*entropija* izvora), a
- koji dolazi do prijemnika, a to je preneseni dio.

Prijenos informacija možemo promatrati s dvije točke motrišta:

1. predajne strane i
2. prijemne strane.

1. Fizikalnu sliku prijenosa informacije gledano ***s motrišta predajnika*** daje izraz (1.37):

$$I(X;Y) = H(X) - H(X/Y)\,.$$

U tomu slučaju, srednji *uzajamni sadržaj informacije* $I(X; Y) = H_T$ (***transinformacija***), jednaka je ***razlici*** između:

- onoga *srednjega sadržaja informacija* potrebnoga da se odredi {skup simbola $X$} ***prije*** *prijema* simbola $Y$, tj. $H(X)$ i onoga
- {skupa simbola $X$} koji je za to potreban ***poslije*** *prijema* $Y$, tj. $H(X/Y)$.

*1.3.5.2.* *1.3.5.2. Ekvivokacija*

Prema tomu očito je da:

- *entropija* $H(X)$ opisuje *srednji sadržaj* ***predane*** *informacije*,

- veličina $I(X; Y) = H_T$, opisuje *srednji sadržaj **primljene** informacije*, koji se odnosi na *predanu vijest*, a
- *uvjetna entropija* $H(X/Y)$ opisuje *srednji sadržaj **izgubljene** informacije* (*istisnute iz kanala*) zbog *utjecaja smetnji*.
- Veličina $H(X/Y)$ opisuje *neodređenost* u odnosu na skup $X$ koja još ostaje nakon prijema $Y$, zbog
  - ***ne-jednoznačnoga** pridruživanja* odgovarajućih simbola (izlaz-ulaz), a
  - što nastupa zbog utjecaja smetnji.
- Zato se veličina $H(X/Y)$ još naziva "*ekvivokacija*" (mnogoznačnost)[26].

2. Izraz (1.38)) $I(X;Y) = H(Y) - H(Y/X)$ prikazuju nam situaciju promatranu ***s prijemne strane***.

Uslijed djelovanja šumova, prijemnik ne može sa sigurnošću ustanoviti kojemu simbolu s *predajne* strane pridružiti primljeni simbol.

Izraz (1.38) izražava *srednji sadržaj prenesene informacije* $I(X; Y) = H_T$, kao *razliku*:

- *srednjeg sadržaja informacije* $H(Y)$ koju prijemnik stvarno prima i
- *srednjeg sadržaja informacije*, $H(Y/X)$ koji je sadržan u primljenim simbolima pod uvjetom da je prethodno poznata predana informacija.

#### *1.3.5.3. 1.3.5.3. Irelevantnost*

Veličina $H(Y/X)$ određuje se strukturom smetnji i opisom njihova međudjelovanja (interakcije) signalom. Zato se veličina $H(Y/X)$ često naziva i *entropija šuma*. Budući da ona u ukupnome sadržaju informacije što dolazi do prijemnika, predstavlja dio koji se ne odnosi na predanu informaciju, naziva se još i *irelevantnost*[27].

#### *1.3.5.4. 1.3.5.4. Uvjeti optimalnoga kodiranja*

Ako su kodne riječi binarnih kodova sastavljene od različitoga broja binarnih simbola, kodovi su neravnomjerni i to predstavlja poteškoću pri dekodiranju. Ravnomjerni binarni kodovi imaju kodne riječi sastavljene od jednakoga broja binarnih simbola.

Entropija $H(U)$ skupa elementarnih vijesti, izražava prosječan sadržaj informacije potreban za jednoznačno određivanje bilo koje vijesti (događaja) iz toga skupa. Svaki simbol kodne abecede u prosjeku donositi maksimalan sadržaj informacije. On je jednak informacijskome kapacitetu abecede $\mathrm{ld}\, L$, ako su svi simboli međusobno nezavisni i jednako vjerojatni. Srednjim brojem takvih simbola $\bar{n}$, može izraziti maksimalan sadržaj informacije jednak $\bar{n}\ \mathrm{ld}\, L$. Prosječan broj kodnih simbola u kodnoj riječi, a koja predstavlja jednu elementarnu vijest, ne smije biti manji od odnosa entropije skûpa vijesti i kapaciteta kodne abecede. Matematičkom obradom prethodnih jednadžbi dobije se:

$$\frac{H(U)}{\mathrm{ld}\, L} \le \bar{n} < \frac{H(U)}{\mathrm{ld}\, L} + 1, \qquad (1.39)$$

a to je granična vrijednosti za $\bar{n}$. Prosječan broj kodnih elemenata potrebnih za kodiranje neke elementarne vijesti je cio broj između vrijednosti $\frac{H(U)}{\mathrm{ld}\, L}$ i $\frac{H(U)}{\mathrm{ld}\, L} + 1$ ukoliko veličina $\frac{n(U)}{\mathrm{ld}\, L}$ već nije cio broj. Time su izraženi uvjeti optimalnoga (ekonomičnoga) kodiranja elementarnih vijesti.

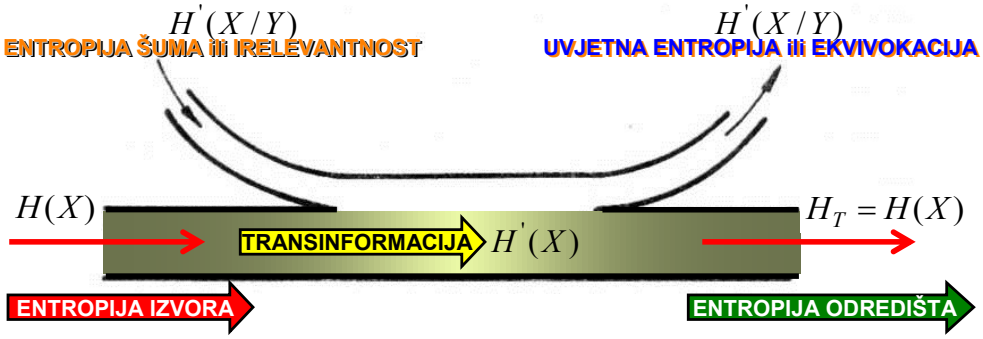
---

[26] *ekvivokacija* … (lat. *aequivocatio*) dvosmislenost, jednak izričaj, dvostruka (višestruka, neodređena) značenja, dvosmislica.

[27] *irelevantan* … (lat. *irrelevans*) neznatan, beznačajan, sićušan, mali, koji je bez značaja, nevažan

## 2. *Auditorna vježba 2: Entropija*

Napomena: Cjelovit opis nalazi se na sustavu za podršku nastavi!



## 1.4. 1.4. Digitalne modulacije

1.4.1. M-struka signalizacija, 1.4.2. Kodiranje valnoga oblika, 1.4.3. Modulacija, 1.4.4. Modulacija impulsnih signala

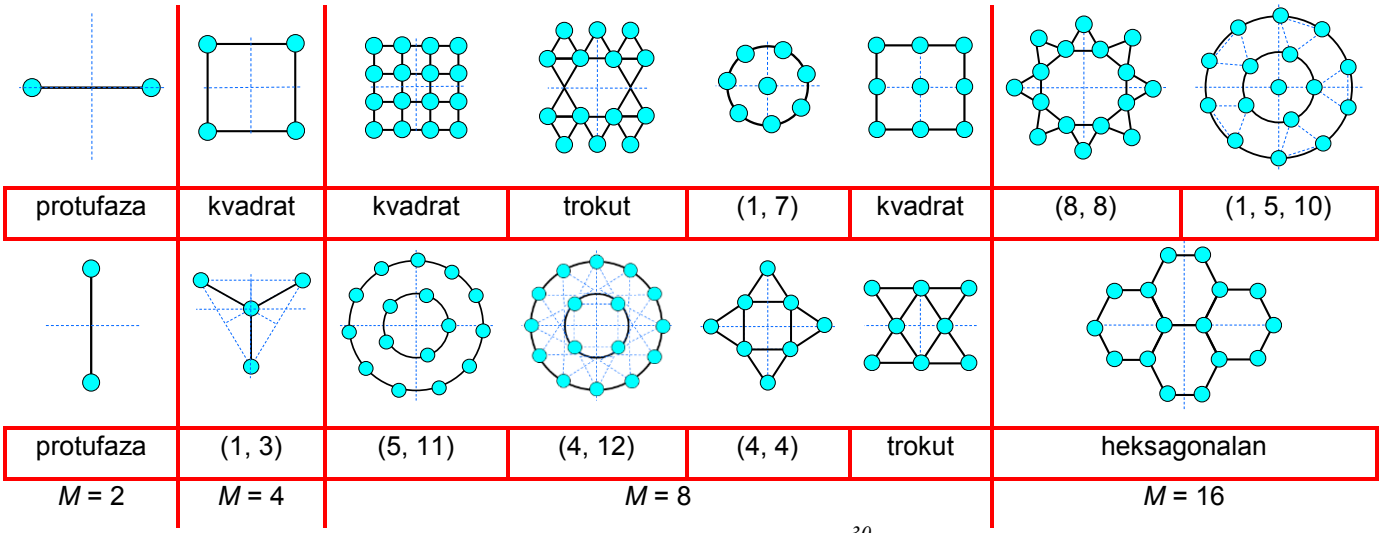
Digitalne modulacije, nadopuna su zaštitnome kodiranju, a služe za prijenos povećane količine informacijskih bitova, kanalom ograničenoga propusnoga pojasa. Ako se blok-kodovi podvrgnu modulacijskome postupku, to se prepoznaje kao *modulacija kodiranoga bloka* BCM (*block coded modulation*). U koliko se isto napravi nad konvolucijskim kodovima, postupak se prepoznaje kao *modulacija kodirane rešetke* TCM (*trellis coded modulation*).

### 1.4.1. 1.4.1. M-STRUKA SIGNALIZACIJA

U *M-strukoj* signalizaciji, procesor prihvaća $k$ podatkovnih bitova u isto vrijeme. Onda ih upućuje modulatoru za proizvodnju jednoga od $M = 2^k$ valnih oblika. Binarna signalizacija poseban je slučaj u kojemu je $k = 1$. Za $k > 1$, sama *M-struka* signalizacija može se smatrati *postupkom kodiranja valnoga oblika*. Za okomitu signalizaciju (npr., MFSK)[28], povećanjem $k$ unaprjeđuju se svojstva pogrešaka ili se smanjuje potreban odnos $E_b/N_0$, na štetu propusnosti. Ne-okomita signalizacija (npr., MPSK)[29] iskazuje bolju iskoristivost pojasne širine, na štetu narušavanja svojstava pogrešaka ili povećanja obveznoga odnosa $E_b/N_0$. Uz odgovarajući izbor valnih oblika signala, može se ostvariti ustupak između vjerojatnosti omjera pogrešaka, $E_b/N_0$ i učinkovite propusnosti.

### 1.4.2. 1.4.2. KODIRANJE VALNOGA OBLIKA

Postupci kodiranja valnoga oblika pretvaraju skup valnih oblika (koji predstavljaju skup poruka) u poboljšan skup valnih oblika. Poboljšan skup valnih oblika onda se može iskoristiti da se osigura veća *vjerojatnost pogreške bita* $P_B$ u odnosu na izvorni skup. Najpopularniji takvi *kodovi valnih oblika* nazivaju se *okomiti* i *dvo-okomiti kodovi.* Postupak kodiranja nastoji da se svaki od valova u kodiranome signalu postavi nasuprot (pomaknut za 180°) ako je to moguće. Cilj je, između svih parova signala napraviti *koeficijent križne korelacije* $z_{ij}$, što je moguće manjim. Najmanja moguća vrijednost koeficijenta križne korelacije postiže se ako signali imaju suprotne faze ($z_{ij} = -1$). To se može postići samo za 2 simbola u skupu ($M = 2$) i samo ti simboli su *protufazni* (slika 1.14).




*Slika 1.14: Primjeri M-strukih konstelacija[30] amplitude i faze*

U načelu, moguće je da su svi koeficijenti križne korelacije nula. Za skup se onda kaže da je *okomit.* Skupovi protufaznih signala su optimalni ako je svaki signal maksimalno udaljen od ostalih signala u skupu. Udaljenost $d$ između signalnih vektora je $d = 2\sqrt{E}$ (vidi dodatak: 10.5. *Protufazni i okomiti signali*), gdje $E$ predstavlja energiju signala za vrijeme trajanja simbola $T$. Usporedbom s protufaznim

[28] *multi-level frequency shift keying* … više-razinska modulacija frekvencije
[29] *multi-level phase shift keying* … više-razinska modulacija faze
[30] *konstelacija* (lat. *constellatio*) astr. zvjezdano jato, zviježđe; položaj zvijezda i njihov tobožnji utjecaj na ljudsku sudbinu; neko određeno svrstavanje; fil. psihička konstelacija opće prethodno stanje svijesti; pren. stjecaj prilika

signalima, svojstva udaljenosti okomitih skupova signala mogu se shvatiti kao "prilično dobra" za određenu razinu energije valnoga oblika.

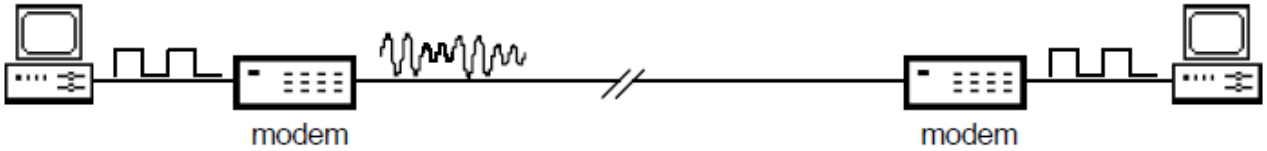
*Križna korelacija* između dvaju signala, mjera je *udaljenosti* među *signalizacijskim* vektorima. Što je križna korelacija manja, vektori su udaljeniji jedan od drugoga. Onda protufazne signale predstavljaju vektori ($z_{ij} = -1$), najudaljeniji jedan od drugoga. Okomite signale ($z_{ij} = 0$) predstavljaju vektori najbliži jedan drugome. Za $z_{ij} = 1$, udaljenost između dvaju istih valnih oblika jednaka je nuli.

Uvjet okomitosti predstavlja se valnim oblicima $s_i(t)$ i $s_j(t)$, gdje su: $i, j = 1, \ldots, M$, a $M$ je veličina skupa valnih oblika. Svaki valni oblik u skupu $\{s_i(t)\}$, može se sastojati od niza impulsa, pri čemu je svaki impuls označen razinom +1 ili –1 i te razine predstavljaju binarne znamenke 1 odnosno 0. Ako se skup $\{s_i(t)\}$ izražava na ovaj način, jednadžba iskazuje da se on sastoji od okomitih signala onda i samo onda ako je

$$z_{i,j} = \frac{\text{broj podudarnih znamenaka} - \text{broj nepodudarnih znamenaka}}{\text{ukupan broj znamenaka}} = \begin{cases} 1 & \text{za : } i = j \\ 0 & \text{za : } i \neq j \end{cases} \qquad (1.41)$$

### 1.4.3. 1.4.3. MODULACIJA

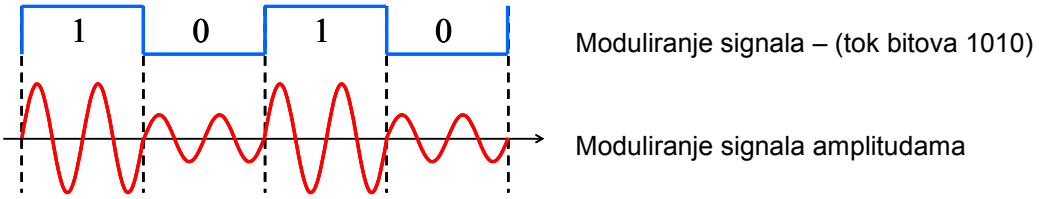
Prijenos digitalnih podataka preko analogne veze ostvaruje se *modulacijom*, gdje se digitalni tok bitova modulira analognim signalom *nositeljem*. Uređaj koji najčešće koristi ovu tehniku zove se **modem** (**mo**dulator-**dem**odulator). Prije napuštanja uređaja, digitalni tokovi bitova pretvaraju se u analogni signal, a na prijemnoj strani, analogni ulazni signal pretvara se u digitalan tok bitova (slika 1.15).




*Slika 1.15: Uloga modema.*

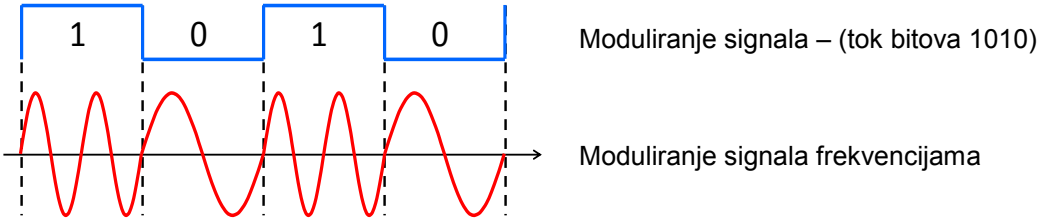
Moguće su tri osnovne vrste modulacija:

1. Modulacija amplitude AM (*Amplitude Modulation*). Za modulaciju amplitude, amplituda signala nositelja mijenja se ovisno o vrijednosti bitova modulacijskoga digitalnoga signala. Na primjer, za predstaviti vrijednosti bitova 0 odnosno 1, mogu se koristiti dvije veličine amplitude (jedna manja, a druga veća). Glavna slabost modulacije amplitude, njezina je osjetljivost na izobličenja (slika 1.16).



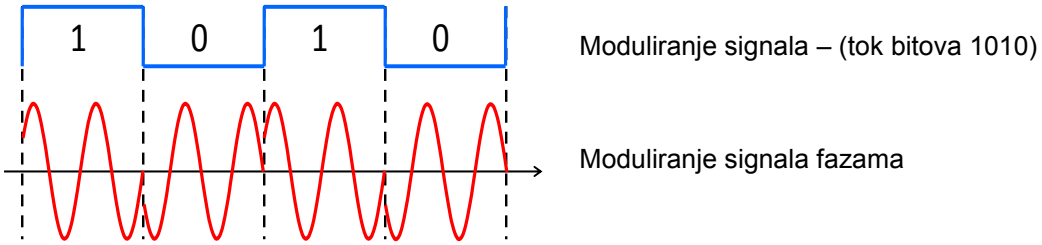

*Slika 1.16: Modulacija bitova dvjema različitim amplitudama*

2. Modulacija frekvencije FM (*Frequency Modulation*). Za modulaciju frekvencijom, mijenja se frekvencija signala nositelja sukladno vrijednostima bitova modulacijskoga digitalnoga signala. Na primjer, za predstaviti vrijednosti bitova 0 odnosno 1 mogu se koristiti dvije različite vrijednosti frekvencije (jedna manja, a druga veća). Modulacija frekvencije otpornija je na izobličenja od modulacije amplitude (slika 1.17).



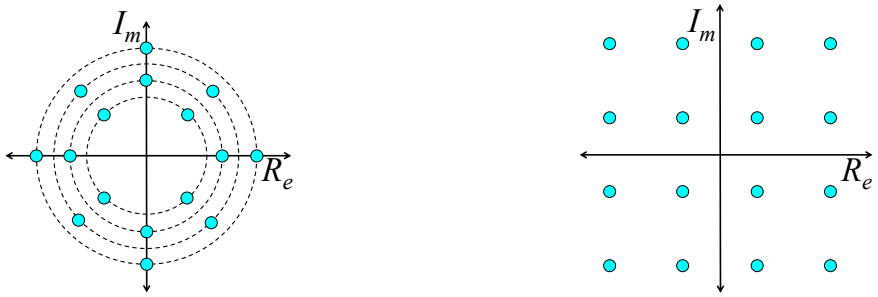

*Slika 1.17: Modulacija bitova dvjema frekvencijama*

3. Modulacija faze PM (*Phase Modulation*). Za modulaciju faze, faza signala nositelja mijenja se ovisno o vrijednosti bitova modulacijskoga digitalnoga signala. Promjena faze signala nositelja, označava promjenu vrijednosti bitova modulacijskoga digitalnoga signala (promjena 0 u 1 ili promjena 1 u 0), slika 1.18.




*Slika 1.18: Modulacija bitova dvjema različitim fazama*

Svako stanje moduliranoga signala određuju četiri binarna simbola. Grayevim kodom, obostrano se pridružuju binarni simboli i diskretna stanja moduliranoga signala. Modulacijski postupak prema slici 1.19 naziva se PSK[31] postupak s četiri razine amplitude i četiri različite faze.

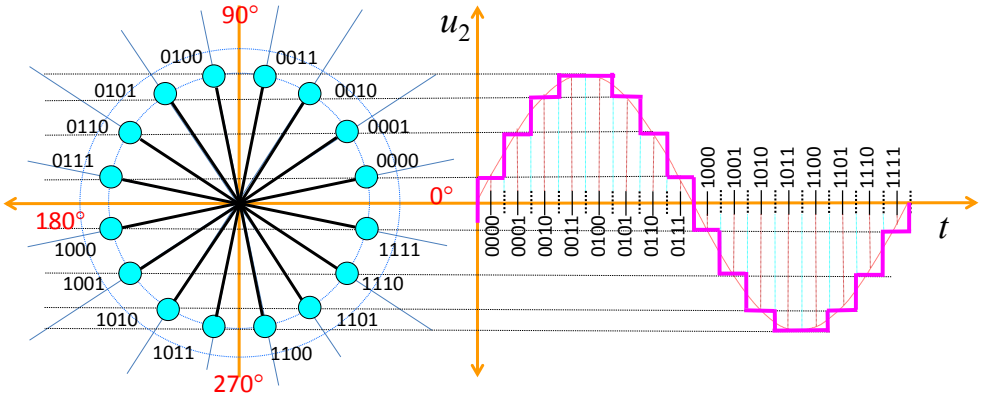



a) *PSK-postupak s 4 stanja amplitude odnosno faze* b) *QASK-postupka sa 16 mogućih stanja*

*Slika 1.19: Mogući položaji vrha jediničnoga usmjerenoga rotacijskoga vektora (versor) moduliranoga signala*

Amplituda predstavlja iznos vektora, a fazni kut određuje smjer vektora. Da bi se naznačila razlika, koriste se zamjenski izrazi kao: *fazor*, *sinor*, *versor* ili *kompleksor*, koji potvrđuju da se sinusne funkcije mogu predstaviti i predstavljaju se kao vektori. Vektorski prikaz i uporaba odgovarajućih vektorskih dijagrama, omogućuju jasan prikaz izmjeničnih vrijednosti.

Prema preporukama udruge CCITT, PSK postupak upotrebljava se za prijenos digitalnih signala od 9600 bit/sek, a brzina rada u digitalnome prijenosu iznosi 2400 [Bd]. Postupak prema slici 1.19 naziva se *diskretnom kvadraturskom modulacijom amplitude* sa 16 diskretnih stanja ($QASK_{16}$). Modulacijski postupak $PSK_4$ odgovara diskretnoj kvadraturskoj modulaciji amplitude s četiri diskretna stanja ($QASK_4$). $PSK_4$-signal dobiva se diskretnom kvadraturskom modulacijom amplitude prijenosnoga signala (slika 1.19) . Proširi li se taj postupak na 16 diskretnih stanja moduliranoga signala, dobije se $QASK_{16}$ (slika 1.20




*Slika 1.20: Digitalna sinteza sinusnoga signala za n = 16*

[31] *PSK* (*Phase Shift Keying*) … modulacija faze

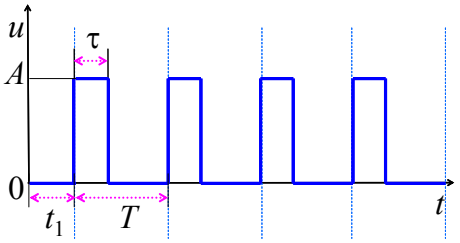
### 1.4.4. 1.4.4. MODULACIJA IMPULSNIH SIGNALA

Sinusni valni oblik prirodan je oblik signala za prijenos informacija posebno u radio-komunikacijama. Zajedničko obilježje svih tih modulacijskih postupaka jest činjenica da modulirani signali zauzimaju minimalne širine pojasa frekvencija, jer se prijenosni signal sastoji od jedne jedine frekvencijske komponente.

Upotreba bilo kojega drugoga valnoga oblika prijenosnoga signala, rezultira širokopojasnim moduliranim signalom. Razlog tomu je, što i sam prijenosni signal zauzima veću širinu pojasa od sinusnih signala. U osnovi, bilo koji periodički signal može se upotrijebiti kao prijenosni signal. Analiziraju se postupci modulacije prijenosnih signala gdje su impulsi pravokutnoga oblika. Takvi modulacijski postupci nazivaju se *impulsnim modulacijskim postupcima*.

Periodički impulsni (digitalni) signal određuju četiri parametra (slika 1.21):

1. amplituda impulsa $A$,
2. vremensko trajanje impulsa $\tau$, odnosno
3. faza $\varphi = 2\pi t_1/T$ i
4. frekvencija ili ponavljanje (*repetition*) impulsa $f = 1/T$

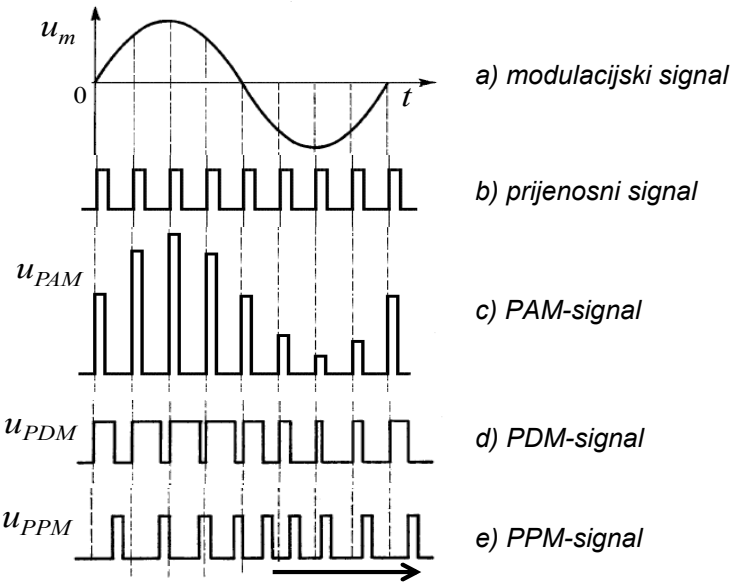



*Slika 1.21: Valni oblik i parametri periodičkoga pravokutnoga impulsnoga signala*

Ti parametri kontinuirane su veličine, tj. mogu poprimiti bilo koju vrijednost u nekim granicama. Impulsni signali moduliraju se mijenjajući jedan ili više svojih parametara ovisno o razini modulacijskoga signala. Na temelju parametra moduliranja, odgovarajući modulacijski postupci svrstavaju se u šest skupina:

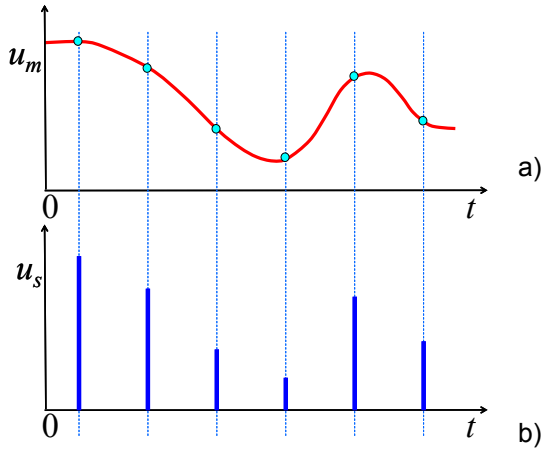
1. modulacija *amplitude* impulsa PAM (*Pulse Amplitude Modulation*);
2. modulacija *trajanja* impulsa PDM (*Pulse Duration Modulation*);
3. modulacija *širine* impulsa PWM (*Pulse Width Modulation*);
4. modulacija *položaja* impulsa PPoM (*Pulse Position Modulation*);
5. modulacija *faze impulsa* ili PPhM (*Pulse Phase Modulation*);
6. modulacija *frekvencije* impulsa PFM (*Pulse Frequency Modulation*).

Slika 1.22 prikazuje primjere valnih oblika modulacijskoga i prijenosnoga signala te tipične oblike moduliranih impulsnih signala.




*Slika 1.22: Valni oblici signala pri modulaciji impulsnih signala*

Impulsni signali diskretne su pojave u vremenu. Zbog toga se ne mogu modulirati vremenski kontinuiranim modulacijskim signalom kao što je npr. signal govora. Prije same modulacije mora se modulacijski signal diskretizirati po vremenu, uzimajući uzorke signala u vremenski trenucima jednakih razmaka (slika 1.23).




*Slika 1.23: Diskretizacija signala po vremenu uzimanjem uzoraka*

Vremenski kontinuiran signal u potpunosti određuju uzorci signala, ako je broj uzoraka u jedinici vremena veći ili jednak dvostrukoj najvišoj frekvenciji u spektru kontinuiranoga signala. Taj uvjet proizlazi iz *teorema o uzimanju uzoraka* i može se izraziti tako da je vremenski razmak $T$ između susjednih uzoraka:

$$T \leq \frac{1}{2f_{\max}} \tag{1.42}$$

Za govorni signal, širina frekvencijskoga pojasa ograničena je na područje od 300 do 3400 Hz. Prema *teoremu o uzimanju uzoraka*, frekvencija uzoraka je barem 6,8 kHz. U praksi se uzorci uzimaju frekvencijom od 8 kHz. Vremenski razmak između uzoraka iznosi 125 μs, a to je ujedno i trajanje perioda impulsnoga prijenosnoga signala koji se modulira.

## 1.5. 1.5. Osnovni pojmovi

Dva važna i srodna parametra bilo kojega digitalnoga komunikacijskoga sustava su *brzina prijenosa* informacija i *propusnost kanala*. Zato što se jedan kodiran simbol prenosi svakih $T$ sekundi, *brzina prijenos simbola* (brzina [*baud*]) je $1/T$. U kodiranome sustavu, ako je kodna brzina $r = k/n$, $k$ informacijskih bitova odgovara prijenosu $n$ simbola, a *brzina prijenosa informacije* (podataka) je $r/T$ [bit/sek] (bitova u sekundi).

- Odnos signala i šuma mjeri se na prijemnoj strani, a predstavlja mjeru *kakvoće prijenosa* (u digitalnome slučaju to je podudarnost ulaznoga i obnovljenoga signala).
- Omjer signal-šum $S/N$ (*signal-to-noise ratio*) $E_b/N_0$: Omjer je energije po informacijskome bitu i spektralne gustoće snage šuma na ulazu u prijemnik. Izražava se u decibelima [dBs].
- Svojstva kodiranoga komunikacijskoga sustava, općenito se mjere vjerojatnošću pogreške dekodiranja, kraće *vjerojatnost pogreške* (*error probability*), a *kodni dobitak* (*coding gain*) mjeri se preko jednoga nekodiranoga sustava koji prenosi informacije istom brzinom. Postoje dvije vrste pogrešaka: *vjerojatnost pogreške riječi* (ili bloka) i *vjerojatnosti pogreške bita*.
- *Brzina pogreške bita BER* (*Bit-Error-Rate*): To je središnja sposobnost upravljanja pogreškama kodiranjem. Ovu veličinu želimo zadržati što manjom, obično manjom od 10. Brzina pogreške bita koristan je pokazatelj svojstava sustava u nezavisnome kanalu s pogreškama. To ima malo značenje na praskovitost ili ovisnost o pogreškama u kanalu.
- Kôd *upravljanja pogreškama ECC* (*Error-Control Code*): To je skup kodnih riječi što ih koriste i koder i dekoder: za *otkrivanje*, za *ispravak*, ili, i za *otkrivanje* i za *ispravak* pogrešaka.
- *Čestoća pojave pogrešaka poruke MER* (*Message Error Rate*): Vjerojatnost pogreške poruke. To svojstvo operater često koristi, jer želi imati poruke bez pogrešaka i manje se brinuti o BER.

- *Čestoća neotkrivene pogrešne poruke UMER* (*Undetected Message Error Rate*): Vjerojatnost da dekoder za otkrivanje pogrešaka pogriješi pa se pogrešna poruka (kodna riječ) provuče neotkrivena. Ovo se događa ako je uzorak pogreška u kanala takav da se pri prijenosu, kodna riječ pretvori u drugu valjanu kodna riječ.
- *Slučajne pogreške* (*Random Errors*): Pogreške što se javljaju samostalno. Ova vrsta pogrešaka pojavljuje se pojedinačno u kanalima isključivo zbog toplinskoga (Gaussovoga) šuma. Kanali samostalnih pogrešaka također se zovu *kanali bez memorije*, jer spoznaja o prethodnim simbolima kanala ništa ne pridonosi našemu znanju o trenutnome simbolu u kanalu.
- *Praskovite pogreške* (*Burst Errors*) - Ovisne pogreške. Na primjer, kanali s velikim iščezavanjem signala (*fading*) iskuse pogreške što se javljaju u praskovima. Budući da iščezavanje signala češće djeluje na uzastopan niz bitova, pogreške se obično smatraju ovisnima. Za razliku od kanala s neovisnim pogreškama, kanali s praskovitim pogreškama imaju memoriju.
- *Energije po bitu* (*Energy Per Bit*): Količina energije sadržane u jednome informacijskome bitu. Ovaj parametar može se izvesti iz drugih poznatih parametara. Energija po bitu $E_b$, važan je parametar, jer se skoro sva oštećenja u kanalu mogu prevladati povećanjem energije po bitu. Energije po bitu [J] (Joule[32]) odnosi se na snagu odašiljača $P_t$ [W] i brzinu $r$ [bit/sek].
- *Kodno pojačanje* (*Coding Gain*): Razlika u [dB] između zahtijevanoga omjera signal-šum za održavanje pouzdane komunikacije, a nakon korištenja kodiranja. Signal-šum obično se daje kao $E_b/N_0$, gdje je $N_0$ spektralna gustoća šuma, a mjeri se u [W/Hz = J].
- *Brzina koda* (*Code Rate*): Koder uzima $k$ informacijskih i dodaje $r$ zalihosnih (*redundant*) bitova ili *paritetnih bitova* (*parity bits*). Brzina koda je odnos $k/n$, a kod se naziva ($n$, $k$) kod za upravljanje pogreškama. U informacijskome smislu, dodani paritetni bitovi nisu potrebni.

Prije nastavka čitanja teksta, vrlo bitno je da student spozna i usvoji sve prethodno navedene pojmove i sve pojmove (p)opisane u nastavku.

[32] Jedinica *rada, energije i količine topline,* izvedena je SI-jedinica, definira je umnožak sile jednoga njutna i jednoga metra, za koji se pomakne hvatište sile u smjeru sile, dakle Joule (džul) je njutn-metar [J = Nm].

# 2. Laboratorijska vježba 1: Kodovi, BCD, Grayev i Manchester kod

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

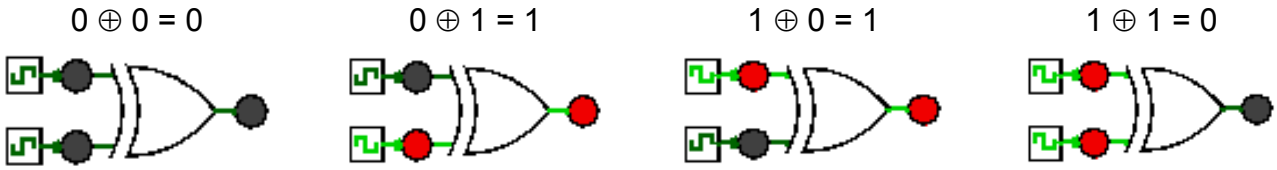
## 1.6. 1.6. Utjecaj pogrešaka na digitalni komunikacijski sustav

Ova skripta bavi se prijenosom binarnih znamenaka - *bitova*, od izvora - *odašiljača* prema odredištu - *prijemniku*. Na putu od odašiljača do prijemnika bitovi prolaze kroz medij - *kanal*. Prolaskom kroz kanal, bitovi su podložni šumovima što uvodi *pogreške* u smislu da neki primljeni bitovi, na mjestima podudarnima položajima u poslanoj poruci, imaju suprotnu vrijednost.

Obrađujemo bitove koji se prenose blokovima određene duljine. Pretpostavlja se da se tijekom prijenosa bitovi ne umeću u blok niti izostavljaju iz bloka. Također se pretpostavlja da uvijek postoji sinkronizacija bloka (tj. prijemnik uvijek zna gdje jedan blok završava i gdje drugi počinje). Sustav prikazan na slici 1.24 poslužuje blok bitova duljine 7.


*Slika 1.24: Komunikacijski sustav*

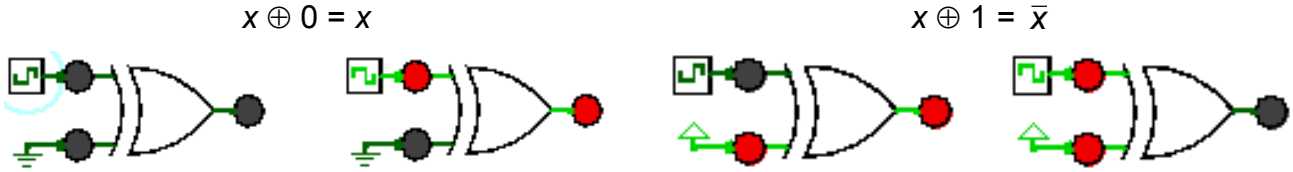
Kanalom se šalje blok 11010001, a primi se blok 10011101. Primljen blok razlikuje od onoga poslanoga bloka u bitovima što se nalazi na drugome i petome mjestu, računajući s lijeva. Ovi bitovi su 0 odnosno 1 u poslanome, a 1 odnosno 0 u primljenome bloku. Treba dobro naučiti logičku operaciju izričito ILI - EXOR (*exclusive OR*), iz navedenoga dodatka (LogiSim). Ona je dobro poznata iz *logike*. Operacija EXOR u cijeloj skripti (u pravilu) obilježava se znakom ⊕. Znakom + obično se označava ILI operacija. Dakle imamo (slika 1.25):




*Slika 1.25: Rezultati XOR zbroja dviju varijabli*

Uočite da, ako je $a \oplus b = c$, onda je $a \oplus c = b$ i $b \oplus c = a$. Ovo promatranje tvrdi da je moguće prenijeti elemente s jedne strane jednadžbe na drugu i još uvijek zadrži znak ⊕, a dokazuje se jednostavno tako da, ako je $a \oplus b = c$, a zatim se "$a$" doda objema stranama jednadžbe, imamo da je $a \oplus a \oplus b = a \oplus c$. Pošto je prema definiciji, $a \oplus a = 0$, onda je $b = a \oplus c$. Ovo svojstvo, što se u ovome tekstu strogo koristi, također znači (za binarni sustav) da ako moramo *oduzimati* (npr., tijekom operacije *dijeljenja*), to možemo napraviti *zbrajanjem*.

Operacija XOR može se sažeti tvrdnjom da operacija XOR bita $x$ s 0 daje bit vrijednosti $x$, a operacija XOR bita $x$ s 1 rezultira bitom vrijednosti $\bar{x}$, gdje je $\bar{x}$ označava komplement od $x$ (slika 1.26)

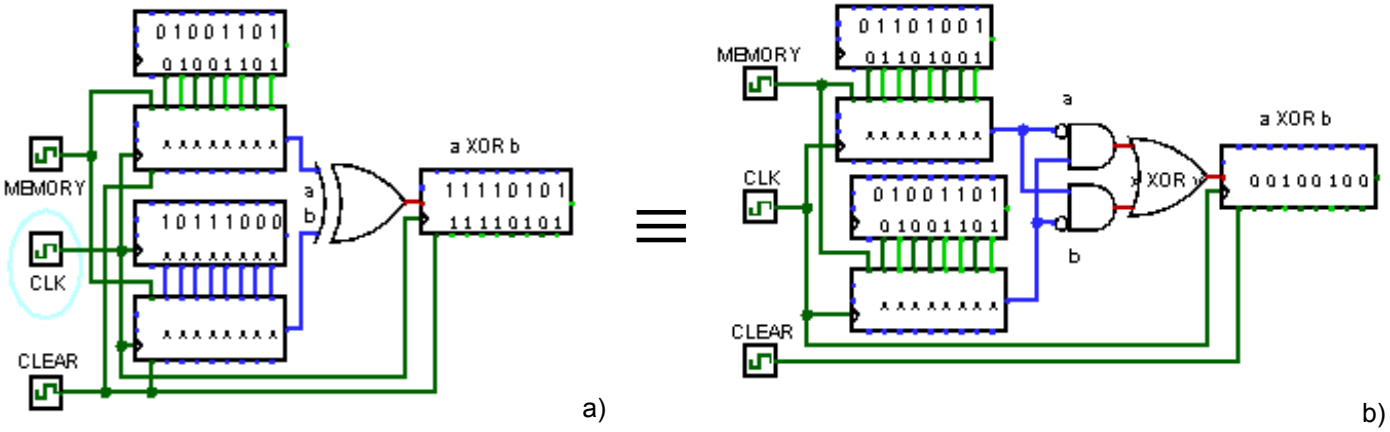



*Slika 1.26: Rezultati XOR zbroja varijable nulom i jedinicom*

Sve slike u skripti predstavljaju ujedno i simulacijske primjere za laboratorijske vježbe ovoga kolegija. Zbroj dvaju blokova bitova (iste duljine) predstavlja rezultat dobiven operacijom XOR na razini bitova dvaju blokova. Rezultat je također blok iste duljine kao i svaki pribrojnik. Njegov prvi element rezultat je dobiven operacijom XOR prvih elemenata dvaju blokova. Njegov drugi element rezultat je dobiven operacijom XOR drugoga elementa obaju blokova, itd. Rezultirajući blok onda *ima* vrijednost 1 na onim mjestima gdje se dva bloka razlikuju, a vrijednost 0 na mjestima gdje imaju istu vrijednost. Kao primjer, razmotrimo dva bloka 01001101 i 10111000. Ovi blokovi razlikuju se u 1., 2., 3., 4., 6. i 8. mjestu. Njihov zbroj je 11110101.

| | | |
|---|---|---|
| **a** = | [01001101] | |
| **b** = | [10111000] | ⊕ |
| **e** = | [11110101] | |

Sada razmotrimo poslane odnosno primljene blokove prikazane na slici 1.27.b.




*Slika 1.27: XOR sklopovi a) XOR kao jedinka, b) XOR napravljen od 2 AND i jednoga OR sklopa*

Ovi blokovi su 01101001, odnosno 01001101. Njihov zbroj je 01001000. Ima elemente vrijednosti 1 na trećemu i šestomu mjestu, a to su mjesta pogrešaka.

Ako je **a** blok što se prenosi, a **b** označava njegovu primljenu inačicu, onda je blok **e** = **a** ⊕ **b**, *blok uzorka pogreške*. Operator ⊕ označava "ili **a** ili **b**, ali ne oba". Ako je rezultat **a** ⊕ **b** = "sve 0", znači da se blok bitova primio (prenio preko kanala) bez pogreške. Blokovi bitova **a** i **b** razlikuju se na onim mjestima gdje se nalazi pogreška što ju je unio šum kanala. Njihov zbroj, ima vrijednost 1 na mjestima gdje se blokovi razlikuju pa blok koji sadrži **1-elemente**[33] ukazuje na mjesta pogrešaka. Imajte na umu da ako je **e** = **a** ⊕ **b**, onda je **b** = **a** ⊕ **e**, što znači da se primjena poruka dobije zbrojem bloka *uzorka pogreške* i primljenoga bloka (*kodna riječ*).

## *1.7. 1.7. Pojam pariteta*

*Definicije* Parnost ili *paritet* (*parity*) cijeloga broja može biti parna (*even parity*) ili neparna (*odd parity*). Mogu se označiti brojčanim vrijednostima. Paran paritet naziva se **parnost 0** pa paran broj jedinica ima paritet 0. Neparan paritet naziva se **parnost 1** pa neparan broj jedinica ima paritet 1. **Paritet određene skupine bitova** odnosi se na parnost broja elemenata vrijednosti 1 u skupini. Razlikujemo 3 skupine provjere pariteta: *okomita* VRC (*vertical redundancy checking*), *uzdužna* LRC (*longitudinal redundancy checking*) i *ciklička* (kružna) CRC *cyclic redundancy checking*)[34], a opisuju se u dodatku na kraju skripte.

Razmotrimo blok [1100 1011]. Paritet prva četiri elementa bloka je 0. Paritet zadnja četiri elementa bloka je 1. Paritet cijeloga bloka je 1. Osnovno svojstvo, koje se u nastavku koristi, je: *paritet zbroja dvaju blokova*, *jednak je zbroju njihovih pariteta*. To jest, ako su **a** i **b** dva bloka čiji je zbroj **c**, gdje su pariteti ovih blokova *a*, *b*, odnosno *c*, onda je $c = a + b$. Slova *a*, *b* i *c* su *bitovi*, a ne *blokovi*.

Kada god se bajt (ili neka skupina bitova) prenosi ili pohranjuje, uvijek postoji mogućnost da jedan ili više bitova slučajno zamijene vrijednosti (komplement). Razmotrimo ova dva binarna broja:

| 1# | 2# | 3# | 4# |
|---|---|---|---|
| 0110 | 1010 | 0011 | 1010 |
| 0110 | 1010 | 0111 | 1010 |

[33] 1-elementi ... znači niz binarnih znamenaka (u bloku) jednakih 1.
[34] CRC (*cyclic redundancy check*) … kružna (ciklička) provjera zalihosti CRC (*cyclic redundancy code*) … *ciklički zalihostan kod*

Oni se razlikuju samo u jednome bitu (pogledajte skupinu 3#). Ako za prvi broj pretpostavimo da predstavlja ono što bi trebalo biti na nekome memorijskome mjestu, a drugi broj je ono što zapravo jest na tome mjestu, onda očito postoji problem. Tablica 1.1 pokazuje nekoliko primjera koji mogu pomoći u definiranju ovoga koncepta.

*Tablica 1.1: Prikaz paritetnoga bita*

| Podatkovni bajt | Paritetan bit | Podatkovni bajt | Paritetan bit |
|---|---|---|---|
| 0000 0000 | 0 | 0000 0100 | 0 |
| 0000 0001 | 1 | 1111 1110 | 1 |
| 0000 0011 | 0 | 1111 1111 | 0 |

U svakome slučaju, paritetan bit koristi se za napraviti podatkovni bajt parnim paritetom (praznine u sredini podatkovnoga bajta, ostavljene su zbog jasnoće).

### 1.7.1. 1.7.1. ITERACIJSKA PROVJERA PARITETA

Jedan od problema pri korištenju pariteta za otkrivanje pogrešaka je, da ne postoji način kako doznati koji od bitova nije ispravan. Na primjer, zamislite da sustav od 8 bitova koristi paran paritet te prima ove podatke i paritetan bit:

1001 1110 ⟵ PARITET = 0

Može se dogoditi da je bajt točan, ali promijenio se sam bit parnosti (kriva pogreška). Bilo bi lijepo da sklop za otkrivanje paritetne pogreške ne ukazuje samo na postojanje pogreške, već i na to koji bit se promijenio pa da ga se može ispraviti.

*Iteracijska provjera pariteta*, jedna je od metoda ispravke pogrešaka. Ako se prenosi niz bajtova od 8 bitova, svaki bajt ima pridružen paritetan bit. Tu je također paritetan bajt što sadrži paritetne bitove za svaki bit u prethodnih pet bajtova. Ovo je najlakše razumjeti pomoću tablice 1.2 (koristi se paran paritet):

*Tablica 1.2: Iteracijski paritet*

| Bajt | Podatkovni bitovi | | | | | | | | Paritetan bit |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** |
| 2 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | **1** |
| 3 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | **1** |
| 4 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | **1** |
| 5 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | **1** |
| **P** | **1** | **0** | **1** | **0** | **1** | **0** | **0** | **1** | **0** |

U ovoj tablici, bajt 1 što se prenosi je 0000 0000. Budući da je sustav koristi paran paritet, a pretpostavlja se da je bajt sa svim nulama paran, onda je paritetan bit jednak 0. Svaki od pet bajtova ima paritetan bit koji je pravilno postavljen tako da svaki bajt (s paritetnim bitom) sadrži paran broj jedinica. Međutim, nakon svake skupine od pet bajtova, "paritetan bajt" prenosi se tako da svaki stupac od pet bitova ima paritet pa se u tablici 1.2 paritetan bit nalazi u retku "P". Dakle, bit parnosti na dnu prvoga stupca je 1, dok stupac sadrži tri preostale jedinice. Kao završna provjera, sam paritetan bajt ima dodan, paritetan bit.

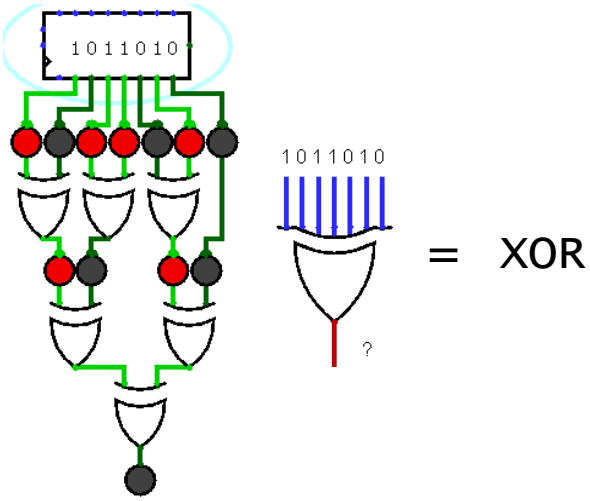
Pretpostavimo da paritet za bajt 1 nije u redu pa ni paritet za bit 0 u paritetnome bajtu nije u redu. Stoga, bit 0 u bajtu 1 treba zamijeniti. Ako paritetan bit u retku nije u redu, ali paritetan bit u stupcu je ispravan, ili paritetan bit u stupcu nije u redu, a paritetan bit u retku je ispravan, onda se promijenio sam paritetan bit. Ovo je jedan jednostavan način ne samo za otkrivanje pogrešnoga podatka, već i za ispravak te pogreške.

Slabost iteracijske provjere pariteta je u tome što je ograničena na pogrešku samo jednoga bita. Ako se u skupini promijeni paran broj bitova, sustav zakazuje, a to je opća slabost većine shema provjere pariteta.

### 1.7.2. 1.7.2. PARITET SKUPINE BITOVA

Standardnim logičkim XOR vratima s višestrukim ulazima, izračunat će se paritet određene skupine bitova. Prema definiciji, izlaz takvih vrata prikazuje paritet skupine bitova koji tvore njegove ulaze. Na

primjer, ako za XOR vrata sa sedam ulaza, točno 6 ulaza ima vrijednost 1, izlaz je 0. Ako točno 3 ulaza imaju vrijednost 1, izlaz je 1. Slika 1.28 prikazuje kako rade XOR vrata s višestrukim ulazima, izrađena od standardnih vrata s dvaju ulaza, a koja imaju izlaz kao paritet ulazne skupine.

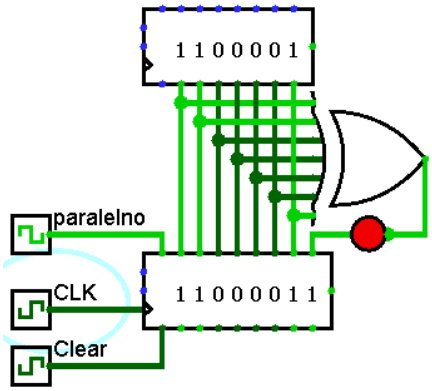



Napomena: Mnogi autori(teti) tvrde da oblikovanje ponašanja XOR vrata treba odgovarati vratima s neparnim paritetom, ali ne postoji dogovor o tome pitanju. Zadano ponašanje za XOR vrata u LogiSim simulacijskome alatu temelji se na standardu IEEE 91. Ponašanje je također u skladu s intuicijskim značenjem na kojemu počiva pojam isključivo ILI: Ako Vas konobar pita želite li prilog od pire krumpira, mrkve, graška ili kelja, on prihvaća samo jedan izbor, a ne tri, bez obzira što neki autori(teti) govorili. Ipak, treba priznati, da ova izjava nije podvrgnuta strogim provjerama. U LogiSim simulacijskome alatu, XOR i XNOR vrata možete konfigurirati koristeći postavku atributa:

Multiple-Input Behavior ⇒ When an odd number are ON.

*Slika 1.28: XOR vrata s više ulaza[35] i paritet ulazne skupine bitova*

Pretvorba bloka bitova duljine $k$ u blok duljine $k$+1 s paritetom 0 predstavlja osnovni postupak. Ako se bloku bitova duljine $k$ s neparnim brojem bitova vrijednosti 1, doda bit vrijednosti 1, dobije se blok duljine $k$+1 koji ima paran broj jedinica. Ako je broj elemenata 1 u bloku paran i doda mu se bit vrijednosti 0, dobije se blok duljine $k$+1 isto s parnim brojem jedinica. Izvoran blok od $k$ bitova, dodavanjem bitova, može se pretvoriti u blok od $k$+1 bitova s paritetom 0 čija je vrijednost jednaka paritetu ovoga bloka. Ova parnost dobiva se pomoću operacije XOR svih $k$ bitova izvornoga bloka. Slika 1.29 prikazuje opisan postupak.




Sklop za automatsko stvaranje pariteta

*Slika 1.29: Pretvorba bloka duljine k u blok duljine k+1 koji ima paritet 0*

## 1.8. 1.8. Otkrivanje pogrešaka

Svrha prijenosa *binarnoga bloka* jest prenijeti neke informacije od predajnika do prijemnika. Pretpostavlja se da primatelj nema nikakve prethodne spoznaje o sadržaju bloka koji mu se šalje. Odašiljač šalje *slučajan* uzorak od $k$ bitova pa se prenošen blok sastoji samo od tih $k$ bitova. Prijemnik ne zna je li se dogodila nekakva pogreška u bloku na putu od odašiljača do prijemnika. On očekuje $k$ nepoznatih bitova i dobije tih $k$ bitova. Da bi se omogućilo prijemniku utvrditi je li dobio blok s pogreškama, blok treba posjedovati neka svojstva. Njihovo narušavanje pokazat će promjenu (bloka podataka) na putu od predajnika do prijemnika.

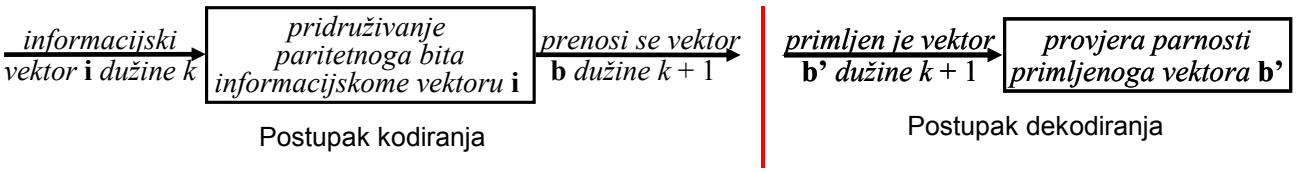
Postojanje tih svojstava znači da primatelj zna nešto o poslanoj poruci, iako ona sadrži slučajne informacijske bitove. To znanje ugrađuje predajnik u poruku prije slanja, tako da se nizu od $k$ informacijskih bitova pridruže dodatni bitovi. Tako se informacija od $k$ bitova proširuje na blok duljine $n$, gdje je $n > k$. Dodatni bitovi suvišni su u smislu informacijskoga sadržaja same poruke.

[35] Napisati vježbu!

Informacijski sadržaj poruke još uvijek je slučajan uzorak od $k$ bitova. Suvišni bitovi potrebni su prijemniku za utvrditi sadrži li dostavljen blok duljine $n$ pogreške (za $n > k$). Postupak odlučivanja, zove se *otkrivanje pogreške*.

Ne postoji način koji je *uvijek* u stanju otkriti postojanje pogrešaka u isporučenome bloku. Uvijek ostaju neki uzorci pogrešaka čije postojanje u dostavljenome bloku ne može se otkriti. Imajte na umu da otkrivanje postojanja pogrešaka ne znači kako primatelj ne zna ništa o prirodi pogrešaka (npr., koliko ih ima i gdje se one nalaze). *Otkrivanje pogrešaka jednostavno znači da primatelj otkriva činjenicu kako su u primljenome bloku moguće* ***pogreške*** pa primljen blok stoga nije jednak poslanome bloku.

Pokazat ćemo osnovni način otkrivanja postojanje neparnoga broja pogrešaka u primljenome bloku. Kao što se prije pokazalo, moguće je pretvoriti bilo koji blok duljine $k$ u blok duljine $k$+1 uz paritet 0 pridruživanjem izvornome bloku bita koji je jednak paritetu bloka. Razmotrimo sada slučaj u kojemu se prenosi informacijski blok duljine $k$. Prije stvarnoga prijenosa, odašiljač pridružuje bloku bit(ove) parnosti (*pariteta*), pretvarajući ga u blok duljine $n = k$+1 čiji je paritet 0. Prijemnik zna da svaki odaslan blok (duljine $k$+1) ima paritet 0. Nakon što dobije blok, prijemnik provjerava njegov paritet pa određuje ima li pogrešaka u primljenome bloku onda i samo onda ako je paritet 1. Slika 1.30 prikazuje opisanu shemu.




*Slika 1.30: Osnovna shema otkrivanja pogrešaka*

U odašiljačkome dijelu sustava, informacijski blok **u** pretvara se u blok **b** za prijenos sustavom, a taj postupak zove se *kodiranje* (*encoding*). Postupak koji se obavlja na prijemnome kraju sustava, gdje se primljen blok **b'** provjerava na pogreške, zove se *dekodiranje* (*decoding*). Imajte na umu da primljen blok nema zapis jednak poslanome bloku. Blok **b'** primljena je inačica **b** i ne mora biti jednaka **b** zbog učinka pogrešaka.

Sada ćemo analizirati uspješnost ove sheme otkrivanja pogrešaka (tj. želimo vidjeti koje se pogreške mogu otkriti, a koje ne mogu). Neka je **b'** = **b** + **e**, gdje je **e** blok uzorka pogreške. Paritet bloka **b** je 0. Prijemnik provjerava ima li **b'** paritet 0. Kao što se prije navelo, paritet zbroja dvaju blokova (iste duljine) jednak je zbroju njihovih pariteta. To znači da će **b'** imati paritet 1 (u slučaju otkrivanja pogreške) onda i samo onda ako **e** ima pariteta 1. Budući da svaki element vrijednosti 1 u **e** ukazuje na pogrešku, slijedi da primjenom naše sheme *možemo otkriti postojanje neparnoga broja pogrešaka.*

Možemo zaključiti da blok od $k$ informacijskih bitova, proširen na $n = k$+1 i prenesen prijemniku, ima paritet 0. Provjerom pariteta primljenoga bloka, moguće je otkriti postojanje *neparnoga* broja pogrešaka ali nije moguće otkriti postojanje *parnoga* broja pogrešaka.

## *1.9. 1.9. Pojam blok-kodova*

U teoriji kodiranja, blok-kodovi obuhvaćaju velik i važan razred kodova za ispravak pogrešaka, a podatke kodiraju u blokovima. Postoji velik broj primjera za blok-kodove, od kojih mnogi imaju širok raspon praktične primjene. Blok-kodovi su koncepcijski korisni, jer omogućuju teoretičarima kodiranja, matematičarima i računalnim znanstvenicima proučavati ograničenja svih blok-kodova na jedinstven način. Takva ograničenja često poprimaju oblik ograničenja što se odnose na različite parametre blok-koda jednih prema drugima, kao što su njihove brzine te njihova sposobnost otkrivanja i ispravaka pogrešaka.

Primjeri blok-kodova su Reed-Solomonovi kodovi, Hammingovi kodovi, Hadamardovi kodovi, proširujući kodovi (*expander codes*), Golayevi kodovi i Reed-Mullerovi kodovi. Ovi primjeri također pripadaju razredu linearnih kodova pa se zbog toga zovu *linearni blok-kodovi* (*linear block codes*).

Ovdje će blok-kod značiti zbirku binarnih blokova, koji se zovu "kodne riječi", a sve riječi iste su duljine. Pojmovi "binarni objekt", "binarni vektor" i "binarna riječ" slobodno se izmjenjuju ovisno o kontekstu u kojemu se koriste. U tablici 1.3 navedene su kodne riječi nekoga koda.

*Tablica 1.3 Primjer kodnoga prostora s paritetnim bitom*

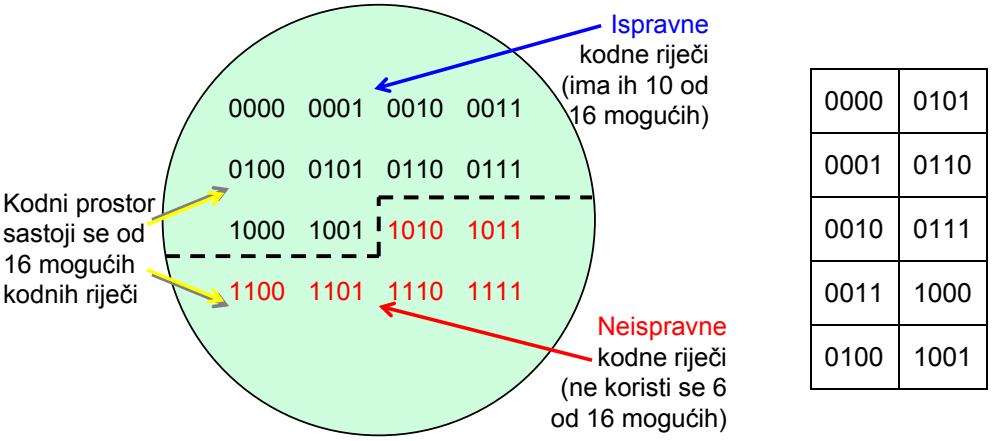
| | | | |
|---|---|---|---|
| 0000 | 1001 | 0101 | 1100 |
| 0011 | 1010 | 0110 | 1111 |

Kôd u tablici 1.3 sastoji se od zbirke svih blokova duljine 4 i ima paritet 0 (svaka riječ ima ukupno paran broj jedinica).

Svaki takav blok je kodna riječ pa u kodu postoji ukupno 8 kodnih riječi. One su napravljene izborom svih mogućih sastava blokova duljine 3, a zatim se svakome bloku pridružio bit parnosti. Kao što se prije objasnilo, ako odašiljač odašilje samo kodne riječi koje pripadaju ovome kodu, a ta činjenica poznata je prijemniku, moguće je otkriti postojanje neparnoga broja pogrešaka u primljenome bloku.

U teoriji kodiranja, *proširujući kodovi* (*expander codes*) tvore razred kodova za ispravak pogrešaka. Oni su izgrađeni od dvodijelnih proširujućih grafova. Uz Justesenove kodove, proširujući kodovi od posebnoga su zanimanja, jer imaju stalnu pozitivnu brzinu, stalnu pozitivnu relativnu udaljenost i stalnu veličinu abecede. U stvari, abeceda sadrži samo dva elementa, pa proširujući kodovi pripadaju razredu binarnih kodova. Nadalje, proširujući kodovi mogu se kodirati i dekodirati u vremenu što je sukladno duljini bloka koda. Proširujući kodovi su jedini poznati asimptotski dobri kodovi koji se mogu kodirati i dekodirati iz cjelovitoga dijela polinoma trajanja pogrešaka.

Još jedan primjer kodnoga prostora prikazuje slika 1.31:




*Slika 1.31: Prostor kodnih riječi popisuje sve kodne riječi*

Prostor kodnih riječi popisuje sve riječi koje oblikuju posebnu abecedu veličine 4 bita (kao što su riječi prikazane na slici). Ispravne kodne riječi dio su tih kodnih riječi, a preslikavaju se u stvarne podatke (npr., 10 kodnih riječi jedinstveno su dodijeljene jednoj od 10 dekadskih znamenaka). Želja nam je kodirati 10 cijelih brojeva, od 0 do 9 digitalnim nizom. Šesnaest jedinstvenih nizova može se dobiti riječima od četiri bita. Od ovih riječi, dodijelimo prvi deset, po jednu svakomu cijelomu broju. Svaki cio broj sada se prepoznaje svojim vlastitim jedinstvenim nizom bitova kao što prikazuje tablica 1.4, a preostalih šest mogućih nizova nisu se dodijelili.

*Tablica 1.4: Dodjela nizova od 4 bita brojevima od 1 do 10*

| niz | broj |
|---|---|
| 0000 | 0 |
| 0010 | 2 |
| 0100 | 4 |
| 0110 | 6 |
| 1000 | 8 |

| niz | broj |
|---|---|
| 0001 | 1 |
| 0011 | 3 |
| 0101 | 5 |
| 0111 | 7 |
| 1001 | 9 |

| niz | broj |
|---|---|
| 1010 | Nepridružen |
| | |
| 1100 | Nepridružen |
| | |
| 1110 | Nepridružen |

| niz | broj |
|---|---|
| 1011 | Nepridružen |
| | |
| 1101 | Nepridružen |
| | |
| 1111 | Nepridružen |

Prostor svih mogućih nizova zove se *kodni prostor* (*code space*). Za nizove od 4 bita, kodni prostor sastoji se od 16 nizova. Od njih smo iskoristili samo deset. To su valjane riječi. Oni nizovi što se ne koriste ili su nepridruženi, nisu ispravne kodne riječi. One se nikada neće poslati, pa ako se dobije jedna od njih, onda prijemnik (ispravno) pretpostavlja da je došlo do pogreške.

Svaka kodna riječ duljine $n$ u blok-kodu općenito je izgrađena proširenjem informacijskoga vektora duljine $k$. Proširenje se napravilo pridruživanjem paritetnoga bita informacijskome vektoru, čije su vrijednosti izračunate iz informacijskih bitova. Broj kodnih riječi u kodu određuje izbor mogućega

broja slučajnih $k$ bitova. Ovaj broj je $2^k$ (varijacije s ponavljanjem od 2 elementa $k$-toga razreda). Za kod u tablici 1.3, $k = 3$ i $n = 4$, a broj kodnih riječi je $2^3$.

*Zapis*: **(*n*, *k*) blok-kod** je kod u kojemu je svaka kodna riječ duljine $n$, od kojih su $k$ bitova informacijski bitovi. Broj paritetnih bitova, stoga je $n–k$.

Kod opisan u tablici 1.3 je (4, 3) blok-kod. Još se nismo osvrnuli na položaje informacijskih $k$ bitova unutar kodne riječi. Ovi bitovi mogu se nalaziti bilo gdje u kodnoj riječ i ne moraju se smjestiti jedan pored drugoga. Blok-kod u kojemu su informacijski bitovi okupljeni zajedno naziva se **sustavan blok-kod**. Za određen kod, informacijski bitovi nalaze se na istim mjestima u svim kodnim riječima.

## *1.10. 1.10. Pregled ostalih uvedenih pojmova*

- Binarni *blok*, binarni *vektor*, binarna *riječ*: Pojmovi se odnose na niz bitova određene dužine.
- *Zbroj dvaju binarnih blokova*: Blok dobiven operacijom XOR dvaju blokova (iste duljine) element po element.
- *Blok uzorka pogreške*: Blok koji ima 1-elemente na onim mjestima gdje i preneseni blok ali se on i njegova poslana inačica razlikuju.
- *Binarni blokovi pariteta* 0: Binarni blok ima paran broj elemenata vrijednosti 1. Ovo uključuje i slučaj gdje blok nema niti jedan element vrijednosti 1 (blok "sve 0").
- *Binarni blokovi pariteta* 1: Binarni blok ima neparan broj elementi vrijednosti 1. Važna napomena: paritet bloka, zbroj ($\oplus$) je bitova jednakih 1.
- *Otkrivanje pogreške*: Na prijemnome kraju postupak utvrđivanja ima li pogrešaka u primljenoj poruci.
- *Binarni blok-kod*: Zbirka binarnih blokova iste duljine.
- *Kodna riječ*: Binarni blok što pripada određenoj vrsti blok-koda.
- *Informacijski sadržaj kodne riječi*: Broj slučajno odabranih bitova u kodnoj riječi.
- (*n*, *k*) *blok-kod*: Blok-kod koji ima kodne riječi duljine $n$, od kojih su $k$ bitova informacijski bitovi. Informacijski bitovi čine podatke koje treba štititi od pogrešaka. Ostatak $n–k$ bitova su "*paritetni bitovi*", a računaju se kao funkcija informacijskih bitova. Njihova svrha je oblikovanje modela ispravke pogrešaka.
- *Sustavan blok-kod*: kodne riječi blok-koda u kojima su informacijski bitovi okupljeni zajedno.

# 3. *Laboratorijska vježba 2: XOR vrata i paritet*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

Sadržaj:



## Priprema za vježbu - pročitati poglavlja iz skripte:

1.6. Utjecaj pogrešaka na digitalni komunikacijski sustav

1.7. Pojam pariteta

1.7.1. Iteracijska provjera pariteta

1.7.2. Paritet skupine bitova

1.8. Otkrivanje pogrešaka

1.9. Pojam blok-kodova

10.6. Vrste provjera zalihosti

1.10. Pregled ostalih uvedenih pojmova

10.6.1. Uzdužna provjera zalihosti

10.6.1.1. Otkrivanje pogreška

10.6.2. Bit pariteta/okomita provjera zalihosti (VRC)

10.6.3. Uzdužna provjera zalihosti (LRC)

10.6.4. Ciklička provjera zalihosti (CRC)

Koristite također skripte iz: **Laboratorijska vježba 0**:

1. Kako koristiti LogiSim paket (nalazi se na MOODLE) i
2. Digitalni sklopovi simulacijskim alatom LogiSim 2.7.1. (nalazi se na MOODLE).

Sklopovi za zbrajanje binarnih brojeva, sklop za automatsko stvaranje pariteta bloka bitova.

# 2. POGLAVLJE 2 - LINEARNI KODOVI

## 2.1. 2.1. Linearni blok-kodovi

Linearni blok-kodovi, razred su linearnih kodova paritetne provjere kojima je svojstvena oznaka (*n, k*). Koder pretvara blok od *k* znamenaka poruke (vektor poruke) u duži blok od *n* znamenaka kodne riječi (kodni vektor) izrađene od zadanih elemenata abecede. Ako se abeceda sastoji od dvaju elemenata (0 i 1), radi se o *binarnome linearnome blok-kodu* koji sadrži binarne znamenke (bitove).

Poruke od *k* bitova oblikuju $2^k$ različitih nizova poruka, nazvanih *k*-torke, (nizovi od *k* znamenaka). Blokovima od *n* bitova može se oblikovati maksimalno $2^n$ različitih nizova, a prepoznajemo ih kao *n*-torke. Postupak kodiranja dodjeljuje svakoj od $2^k$ poruka (*k*-torki) jednu od $2^n$ proširenih poruka (*n*-torki). Blok-kod predstavlja pridruživanje *jedan na jedan* (bijekcija[36]), pri čemu se $2^k$ poruka *k*-torki *jedinstveno* preslikava u nov skup od $2^k$ *n*-torki kodnih riječi. Preslik se može postići pomoću *pregledne tablice*. Za *linearne-kodove,* pretvorba preslikom je *linearna*. Izričito (formalno[37]), blok-kod predstavlja injekcijski preslik: $C$: $\sum^k \rightarrow \sum^n$, gdje je $\sum$ konačan ne prazan skup, a *k* i *n* su cijeli brojevi.

## 2.2. 2.2. Osnovni pojmovi

*Definicija* **Linearan-kod** je kod, čiji zbroj bilo kojih dviju kodnih riječi, također je kodna riječ. Način zbroja dvaju binarnih blokova već se definirao u poglavlju 1.6. Ovo pokazujemo razmatranjem koda navedenoga u tablici 1.3. Zbroj bilo kojih dviju kodnih riječi od 4 bita, *izvan* podskupa od 8 riječi iskazanih binarno, pripada podskupu od 8 *odabranih* kodnih riječi. U nastavku, ova tvrdnja potkrjepljuje se simulacijskim primjerom i nekoliko C++ zadataka.

### 2.2.1. 2.2.1. SIMULACIJSKI PRIMJER

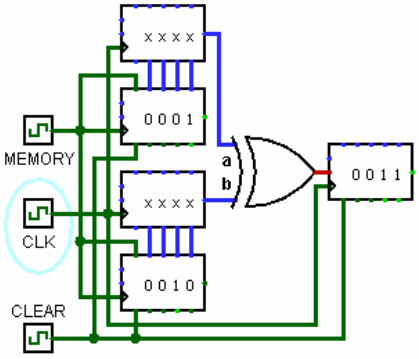
Iz skupa od 16 mogućih riječi od 4 bita, odabire se podskup od 8 riječi. Provjerava se *prethodna tvrdnja* (ponovljena tablica 1.3 dolje) da zbroj bilo kojih dviju riječi iz podskupa od 8 *ne-odabranih* riječi, daje riječ iz podskupa od 8 *odabranih* riječi: 0000, 1001, 0101, 1100, 0011, 1010, 0110, 1111. Zadane su funkcije pretvorbi: dekadski→binarno i binarno→dekadski. Imamo 2 skupa riječi od po 4 bita, a podjela se ponovno prikazuje u tablici uz sliku 2.1 (lijevo).

| Riječi u kodu | | Riječi izvan koda | |
|---|---|---|---|
| 1. | 0000 | 9. | 0001 |
| 2. | 0011 | 10. | 0010 |
| 3. | 0101 | 11. | 0100 |
| 4. | 0110 | 12. | 0111 |
| 5. | 1001 | 13. | 1000 |
| 6. | 1010 | 14. | 1011 |
| 7. | 1100 | 15. | 1101 |
| 8. | 1111 | 16. | 1110 |

Primjeri zbroja: 0001 + 0010 = 0011
0011 + 1001 = 1010

| |
|---|
| 0 + 0 = 0 |
| 0 + 1 = 1 |
| 1 + 0 = 1 |
| 1 + 1 = 0 |

≡ XOR




*Slika 2.1: Zbrajanje riječi u 4 posmika*

Lijevi stupac tablice, tvore riječi koje se koriste u kodu, a zalihosne kodne riječi su u desnome stupcu i ne koriste se za kodiranje.

*Zadatak 2.1*: Dokažite tvrdnju (vidi C++ zadatak 2. u nastavku) da zbroj bilo kojih dviju nekorištenih kodnih riječi iz desnoga stupca, daje korištenu kodnu riječ u lijevome stupcu. Za zbrajanje se koristi XOR operacija (ili jedan ili drugi ali NE oba), vidi prilaženu tablicu XOR operacija (slika 2.1 sredina). Rješenje mora sadržavati 2 funkcije: prva funkcija ispituje sve moguće zbrojeve dviju nekorištenih riječi (od 9. do 16.), a druga funkcija odmah ispituje pripada li zbroj dviju riječi, skupu riječi (od 1. do 8.). Kodne riječi pridružite elementima odgovarajućih polja u trećoj funkciji.

---

[36] Bijekcijska funkcija ili bijekcija: Neka su *A* i *B* skupovi, a *f* funkcija iz *A* u *B*. Ako je propis pridruživanja *f* između elemenata skupa *A* i skupa *B* takav da *različitim* elementima skupa *A* pridružuje *različite* elemente skupa *B*, a nema nepridruženih elemenata, funkcija *f* zove se *bijekcijska* funkcija ili *bijekcija*.

[37] *formalno* … (lat. *forma*) besadržajno; izričito, jasno izraženo, jednostavno

*1. Glavni program za prethodni zadatak (uključuje funkcije u nastavku)*

```
#include<iostream>
#include<cmath>
using namespace std;
int broj1(int broj_1);
//Funkcijom
void b2d3(int d,int b4[][4],int l,int dekod[8]);
//izračunati dekadske ekvivalente 8 kodnih riječi,
//a rezultate spremiti (npr. u numeričko polje u strukturi)
//struct ukupno{
//char kodni_vektori[8]; //
//char ne_kodni_vektori[8]; //
//char kombinacije[28];
//Hammingova težina binarnoga bloka je broj elemenata vrijednosti 1.
//Hamming_tezina[] //broj elemenata vrijednosti 1
//Hammingova udaljenost između 2 binarna bloka (iste dužine)=broj
mjesta na kojima se razlikuju.
//zbroj dvaju blokova a i b je blok c koji ima elemente vrijednosti
1 na onim mjestima gdje se dva bloka a i b razlikuju.
//(Z)broj 1-elemenata bloka c je Hammingova udaljenost između a i b.
//Ovaj broj je Hammingova težina bloka c.
//Onda imamo da je Hammingova težina zbroja dvaju binarnih blokova,
//jednaka Hammingovoj udaljenosti između ta dva bloka.
//Računa dekadski ekvivalent 8 nekodnih znamenaka
void b2d3(int d,int b4[][4],int l);
//Zbrojiti XOR svaku od 8 nekodnih riječi sa svakom preostalom
(suma=28)
//To su Kombinacije bez ponavljanja [n nad r = 8 nad 2 =
(8x7)/(1x2)=28]
//Provjera jesu li svi dobiveni zbrojevi, riječi koda.
//razlika() //može se dodati funkcija provjere jesu li sve znamenke
različite
void main(){
int d(0);
int l=4;
int dekod[8];
int kod[8][4]={{0,0,0,0},{0,0,1,1},{0,1,0,1},
{0,1,1,0},{1,0,0,1},{1,0,1,0},{1,1,0,0},{1,1,1,1}};
int da_kod[8];
cout<<"Kodni vektori:"<<endl;
b2d3(d,kod,l,dekod);
//int b4[][4]={{0,0,0,1},{0,0,1,0},{0,1,1,0},{0,1,1,1},
{1,0,0,0},{1,0,1,1},{1,1,0,1},{1,1,1,0}};
int nekod[][4]={{0,0,0,1},{0,0,1,0},{0,1,0,0},{0,1,1,1},
{1,0,0,0},{1,0,1,1},{1,1,0,1},{1,1,1,0}};
int ne_kod[8];
cout<<"NE kodni vektori:"<<endl;
b2d3(d,nekod,l);
}
void b2d3(int d,int b4[][4],int l){
int i=0,broj_1(0);
for(int i=0;i<8;i++){
d=0;
for(int j=0;j<4;j++)cout<<b4[i][j];//ispis 4 znamenke koda
```

*1. Glavni program za prethodni zadatak (uključuje funkcije u nastavku)*

```
//cout<<"=";
broj_1=0;
for(int j=0;j<4;j++){
if(b4[i][j]==1){
d+=pow(2.,3-j);
broj_1=broj1(broj_1);
}
cout<<":"<<b4[i][j]<<" d="<<d;
}
cout<<", broj \'1\'="<<broj_1<<"=Hamm tezina"<<endl;
}}
void b2d3(int d,int b4[][4],int l,int dekod[8]){
int i=0,broj_1;
for(int i=0;i<8;i++){
d=0;
for(int j=0;j<4;j++)cout<<b4[i][j]; //ispis 4 znamenke koda
//cout<<"=";
broj_1=0;
for(int j=0;j<4;j++){
if(b4[i][j]==1){
d+=pow(2.,3-j);
broj_1=broj1(broj_1);
}
cout<<":"<<b4[i][j]<<" d="<<d;
}
cout<<", broj \'1\'="<<broj_1<<"=Hamm tezina"<<endl;
}}
int broj1(int broj_1){broj_1++;return broj_1;}
```

```
C:\windows\system32\cmd.exe
Kodni vektori:
0000:0 d=0:0 d=0:0 d=0:0 d=0, broj '1'=0=Hamm tezina
0011:0 d=0:0 d=0:1 d=2:1 d=3, broj '1'=2=Hamm tezina
0101:0 d=0:1 d=4:0 d=4:1 d=5, broj '1'=2=Hamm tezina
0110:0 d=0:1 d=4:1 d=6:0 d=6, broj '1'=2=Hamm tezina
1001:1 d=8:0 d=8:0 d=8:1 d=9, broj '1'=2=Hamm tezina
1010:1 d=8:0 d=8:1 d=10:0 d=10, broj '1'=2=Hamm tezina
1100:1 d=8:1 d=12:0 d=12:0 d=12, broj '1'=2=Hamm tezina
1111:1 d=8:1 d=12:1 d=14:1 d=15, broj '1'=4=Hamm tezina
NE kodni vektori:
0001:0 d=0:0 d=0:0 d=0:1 d=1, broj '1'=1=Hamm tezina
0010:0 d=0:0 d=0:1 d=2:0 d=2, broj '1'=1=Hamm tezina
0100:0 d=0:1 d=4:0 d=4:0 d=4, broj '1'=1=Hamm tezina
0111:0 d=0:1 d=4:1 d=6:1 d=7, broj '1'=3=Hamm tezina
1000:1 d=8:0 d=8:0 d=8:0 d=8, broj '1'=1=Hamm tezina
1011:1 d=8:0 d=8:1 d=10:1 d=11, broj '1'=3=Hamm tezina
1101:1 d=8:1 d=12:0 d=12:1 d=13, broj '1'=3=Hamm tezina
1110:1 d=8:1 d=12:1 d=14:0 d=14, broj '1'=3=Hamm tezina
Press any key to continue . . .
```

Kombinacije bez ponavljanja za međusoban zbroj po dvije ne-poruke: ($n$ nad $r$) ≡ (8 nad 2) = $\binom{8}{2}$ = (8 · 7)/(1 · 2) = 28]. Primjeri:

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9. = | [0001] | | | 12. = | [0111] | | | 11. = | [0100] | | | 16. = | [1110] | | | 13. = | [1000] | | |
| 10. = | [0010] | + | | 15. = | [1101] | + | | 9. = | [0001] | + | | 14. = | [1011] | + | | 10. = | [0010] | + | |
| 2. = | [0011] | | | 6. = | [1010] | | | 3. = | [0101] | | | 3. = | [0101] | | | 6. = | [1010] | | |

Svojstvo linearnih kodova da zbroj bilo kojih dviju *kodnih* riječi daje treću kodnu riječ članicu pod-prostora, navodi se jednostavno kao: Ako su **u** i **v** kodne riječi, onda **w** = **u** + **v** također je kodna riječ. Stoga je *udaljenost između dviju kodnih riječi* jednaka *težini treće kodne riječi*, to jest,

$$d(\mathbf{u}, \mathbf{v}) = w(\mathbf{u} + \mathbf{v}) = w(\mathbf{w}).$$

Najmanja udaljenost linearnoga koda može se odrediti bez mjerenja udaljenost između svih sastava parova kodnih riječi (što upravo radi prethodni C++ zadatak). Samo treba ispitati *težinu svake kodne riječi* (isključujući kodnu riječ "sve 0") u pod prostoru minimalne težine što odgovara minimalnoj udaljenosti $d_{min}$. Isto tako, $d_{min}$ podudara se *najmanjom udaljenošću* (iz skupa svih udaljenosti) između kodne riječi "sve 0" i svih ostalih kodnih riječi.

### 2.2.2. 2.2.2. PROGRAMSKE PODRŠKE BROJEVNIM PRETVORBAMA

*2. Pretvorba: oktalno - binarno*

```
// Ugraditi provjeru ispravnosti upisanoga broja
#include <iostream>
using namespace std;
void oct2bin(int);

void main(){
int a;
cout << "Unesite 2-znamenkasti\noktalni broj: ";
cin>>a;
oct2bin(a);
// getchar();
}
void oct2bin(int oct){
long bin=0;
int A[6];
//Svaka oktalna znamenka pretvara se u 3 bita,
//2 oktalne znamenke = 6 bitova ukupno.
int a1,a2,kol,rem,x;
a2=oct/10;
a1=oct-a2*10;
for(int x=0;x<6;x++)A[x]=0;
//Skladišti se ostatak jedinica oktalne znamenke u polju.
for(x=0;x<3;x++){kol=a1/2;rem=a1%2;A[x]=rem;a1=kol;}
//Skladišti se ostatak "osmica" oktalne znamenke u polju.
for(x=3;x<6;x++){kol=a2/2;rem=a2%2;A[x]=rem;a2=kol;}
//Binarni broj dobije se iz ostataka.
for(x=x-1;x>=0;x--){bin*=10;bin+=A[x];}
cout<<"Binarni ekvivalent\noktalnoga broja "<<oct<<" je "<<bin<<
"."<<endl;
}
```

```
C:\windows\system32\cmd.exe
Unesite 2-znamenkasti
oktalni broj: 67
Binarni ekvivalent
oktalnoga broja 67 je 110111.
Press any key to continue . . .
```

Slijedi niz C++ programa koji će se obraditi na laboratorijskim vježbama u prikladnome trenutku.

1. Pretvorba dekadskoga broja u binarni (vrijedi za dekadske brojeve manje od 16).;
2. Pretvorba dekadskih brojeva (0-15) u binarne (jedan stupac).;
3. Pretvorba dekadskih brojeva (0-127) u binarne (jedan stupac).;
4. Pretvorba dekadskih brojeva (0-127) u binarne (jedan stupac-2 `for` petlje) 1. varijanta.;
5. Pretvorba dekadskih brojeva (0-127) u binarne (jedan stupac, 2 `for` petlje) 2. varijanta.;
6. Pretvorba dekadskih brojeva (0-127) u binarne (osam stupaca).;
7. Pretvorba binarnoga broja u dekadski (inačica `char`).;
8. Pretvorba binarnoga broja u dekadski (inačica `string` nastavlja se na inačicu `char`);
9. Pretvorba binarnoga broja u dekadski (inačica `int polje[]` nastavlja se na inačicu `char`).;
10. Pretvorba binarnoga broja u dekadski (inačica `int polje[][]` nastavlja se na inačicu `char`).;

Varijanta upisa binarnoga broja:

11. Pretvorba binarnoga broja u dekadski (metoda `cin.getline(b0,5)` – izbornik.

### 2.2.3. 2.2.3. OSTALE DEFINICIJE I PRETVORBE C++ PROGRAMOM

*Definicija* **Hammingova težina** binarnoga bloka je broj **elemenata** vrijednosti **1** u njemu.

*Definicija* **Hammingova udaljenost** između dvaju binarnih blokova (dva bloka *moraju* imati iste dužine) broj je mjesta u kojima se ta dva binarna bloka razlikuju.

*Primjer*: Dva bloka, [1**0**1**0**1**1**1] i [1**1**1**1**0**1**0] imaju Hammingovu *težinu* 5 (broj jedinica u riječi). Razlikuju se na drugome, četvrtome, petome i sedmome mjestu. Ukupno se razlikuju na četiri mjesta pa je njihova međusobna Hammingova *udaljenost* jednaka 4.

Blok **c** predstavlja *zbroj* dvaju blokova **a** i **b**, a ima elemente vrijednosti 1 na onim mjestima gdje se ta dva bloka (**a** i **b**) razlikuju. (Z)broj 1-elemenata bloka **c** je stoga Hammingova udaljenost između **a** i **b**. Ovaj broj, s druge strane, je *Hammingova težina* bloka **c**. Onda imamo da je Hammingova *težina* zbroja dvaju binarnih blokova, jednaka Hammingovoj *udaljenosti* između tih dvaju blokova.

### *3. Određivanje Hammingove težine skupa od 16 binarnih vektora*

| |
|---|
| Zbroj dvaju binarnih vektora **a** i **b** je vektor **c** koji ima elemente vrijednosti 1 na onim mjestima gdje se vektori **a** i **b** razlikuju (XOR). (Z)broj 1-elemenata vektora **c** je *Hammingova udaljenost* između **a** i **b**, a to je ujedno *Hammingova težina* bloka **c**. Za zadan skup binarnih vektora: {0001, 0010, 0100, 0111, 1000, 1011, 1101, 1110}, odredite *minimalnu* Hammingovu težinu. Skup vektora upišite kao `int polje[8][4]`. |
| |

*Definicija* **Minimalna Hammingova udaljenost** koda, minimalan je rezultat dobiven mjerenjem udaljenosti između svih mogućih parova kodnih riječi.

**Minimalna Hammingova težina** koda, minimalan je rezultat dobiven mjerenjem Hammingove težine svih kodnih riječi, osim kodne riječi koja sadrži "sve 0" (trebala bi postojati).

*Primjer* Minimalna Hammingova udaljenost koda od 8 kodnih riječi što ih prikazuje tablica 1.3 je 2, jer se bilo koje dvije riječi razlikuju u najmanje 2 mjesta. Minimalna Hammingova težina također je 2.

U *linearnome kodu*, *minimalna Hammingova težina jednaka je minimalnoj Hammingovoj udaljenosti*. Blok "sve 0" je kodna riječ zbroja bilo koje kodne riječi same sobom i mora pripadati kodu (na temelju definicije linearnosti) što vodi kodnoj riječi "sve 0". Ako je **a** kodna riječ koja ima minimalnu Hammingovu težinu *d*, onda je minimalna Hammingova *udaljenost* između **a** i bloka "sve 0" također *d*. Nije moguće imati dvije kodne riječi, npr. **b** i **d**, čija je udaljenost *e* manja od *d*. Ako bi to bio slučaj, onda je **c** + **d** također kodna riječ i mora imati težinu *e*. To je u suprotnosti s činjenicom da je *d* minimalna težina. Jedini blok-kodovi kojima se bavimo u ovoj skripti su *linearni kodovi*.

Slijedi popis C++ programa koji potvrđuju uvedene i navedene pojmove.

12. Određivanje Hammingove udaljenosti skupa od 8 binarnih vektora EXOR operatorom

### *4. Dekadski ekvivalenti nekodiranih ulaznih vektora i broj jedinica ('1') u njima*

Zadan je skup od 8 binarnih vektora (4 bita): {$[0001]^T$, $[0010]^T$, $[0100]^T$, $[0111]^T$, $[1000]^T$, $[1011]^T$, $[1101]^T$, $[1110]^T$}. Napišite funkciju koja računa dekadski ekvivalent svakoga vektora. Koliko svaki binarni vektor sadrži jedinica ('1')? Koliki je ukupan broj jedinica ('1') u svim vektorima?

13. Određivanje Hammingovih udaljenosti za 28 sastava binarnih vektora EXOR operatorom.;
14. Hammingova težina zbroja dvaju binarnih blokova.;
15. Minimalna Hammingova težina kodnih vektora (sva sjedinjenja EXOR zbrojeva ne-kodnih vektora i provjera dobije li se jedan od 8 kodnih vektora.;
16. Odrediti, pripada li zbroj dvaju ne-kodnih vektora (∑=28) jednome iz skupa kodnih vektora {8}.

## *2.3. 2.3. Minimalna Hammingova udaljenost i sposobnost otkrivanja/ispravke pogrešaka*

### 2.3.1. 2.3.1. OTKRIVANJE POGREŠAKA

Prenosi se vektor **a** = [10110011]. Vektor **a** podliježe pogreškama na putu do primatelja, gdje se pogreške prepoznaju kao *uzorak pogreške* **e** = [11001001]. Primljeni vektor **b** onda je **a** + **e** = [01111010]. Broj pogrešaka koje se pojave u ovome primjeru u vektoru **a**, na putu do prijemnika je 4, a to je Hammingova težina **e**. Ovaj broj je Hammingova udaljenost između **a** i njegove primljene inačice **b**. Općenito se može reći da Hammingova udaljenost između poslanoga vektora i njegove primljene inačice, predstavlja Hammingovu težinu uzorka pogreške.

Pretpostavimo da odašiljač i prijemnik imaju popis kodnih riječi specifičnoga koda i usuglašeni su da se bilo koji preneseni blok među njima, mora sastojati od ovih kodnih riječi. Nakon što dobije poruku, prijemnik odlučuje postoje li pogreške u poruci, provjerom pripada li kodna riječ kodu. Ako pripada, odlučit će ima li ili nema pogreške. Ako primljena poruka nije kodna riječ, prijemnik odlučuje sadrži li primljena poruka pogrešku.

*Definicija* **Mogućnost otkrivanja pogreške koda** maksimalan je broj pogrešaka što se mogu pojaviti u poslanoj kodnoj riječi. To prijemniku omogućuje otkriti je li primljena poruka, pogrešna.

Takvo otkrivanje podupire promatranje da primljena poruka nije kodna riječ. Sada se vidi povezanost između sposobnosti *otkrivanja pogrešaka* u kodu i njezine *minimalne* Hammingove udaljenosti. Označimo minimalnu Hammingovu udaljenost $d_{\min}$. Ovo znači da se bilo koje dvije kodne riječi razlikuju u najmanje $d_{\min}$ mjesta.

Sada pretpostavimo prijenos kodne riječi **a** koja se primila kao vektor **b**. Ako je **b** također kodna riječ, onda je došlo do pogreške pa **b** nije jednak **a**. Takva pogreška neće se otkriti, jer se otkrivanje pogrešaka temelji na provjeri je li **b** *bilo koja* kodna riječ. Bez obzira koliko se paritetnih bitova pridruži informacijskome vektoru, nemoguće je ustrojiti kod u kojemu se postojanje pogrešaka u primljenoj poruci *uvijek* otkrije (osim "koda" koji se sastoji od samo jedne kodne riječi). To je zbog činjenice da uvijek postoji mogućnost slanja kodne riječ **a** i prijema druge kodne riječ **b** pa se pogrešaka ne može otkriti.

*Zaključak* 2.1 U primljenoj poruci, pogreške se ne mogu otkriti onda i samo onda ako je vektor uzorka pogreške i sam, kodna riječ.

Promotrimo prethodno razmatrani primjer, u kojemu se kodna riječ **a** dobila kao još jedna kodna riječ **b** (to je jedini slučaj u kojemu nije moguće otkriti pogreške). Neka je **e** = **a** + **b**. Zbog linearnosti koda, uzorak pogreške **e** i sam je kodna riječ.

Zaključak 2.1 također oblikuje temelj za izračun vjerojatnosti neprepoznavanja pogrešaka. Mnogi pouzdani komunikacijski sustavi temelje samo na otkrivanju pogrešaka, u smislu da se provjeri sadrži li primljena poruka pogreške. Ako se otkriju pogreške (tj., otkrije se da primljena poruka nije kodna riječ), traži se ponavljanje prijenosa poruke (jedna od 3 glavne metode ARQ) pa će se ponovljenim slanjem poruke, ona opet provjeriti na pogreške, itd., sve dok se ne primi poruka bez otkrivene

pogreške. Važno je izračunati *vjerojatnost neprepoznavanja pogrešaka* kako bi se analizirala učinkovitost sustava. Za linearan kod, to je vjerojatnost da je vektor uzorka pogreške, kodna riječ.

Koliki je minimalan broj pogrešaka koje će se pojaviti u poslanoj kodnoj riječi **a**, u slučaju da se primi neka druga kodna riječ (u tome slučaju nemoguće je otkriti pogreške)? Kako je $d_{\min}$ minimalan broj mjesta u kojima se kodna riječ razlikuje od bilo koje druge kodne riječi, to znači da se barem $d_{\min}$ pogrešaka mora dogoditi za vrijeme prijenosa, kako bi se riječ **a** primila kao druga kodna riječ **b**. S druge strane, ako je broj pogrešaka koje se pojave u **a**, jednak $d_{\min}-1$ ili manji, uvijek će se otkriti pojava pogreške, jer broj pogrešaka nije dovoljan za pretvorbu u neku drugu kodnu riječ.

*Zaključak* 2.2 Mogućnost otkrivanja pogreške koda je $d_{\min}-1$, gdje $d_{\min}$ označava minimalnu Hammingovu udaljenost koda.

### 2.3.2. 2.3.2. ISPRAVAK POGREŠAKA

Razmotrit ćemo mogućnost *ispravke* pogrešaka u primljenoj poruci, znajući da se poslala kodna riječ. Ispravak pogrešaka u poruci znači određivanje mjesta pogrešnih bitova, a nakon što se otkrije da je poruka primljena kao pogrešna (tj., otkrije se da dobivena poruka nije kodna riječ). Ispravljena poruka treba bi biti jednaka kodnoj riječi u kodu koji je *zajednički* i odašiljaču i prijemniku.

*Definicija* **Sposobnost ispravke pogrešaka** koda, maksimalan je broj pogrešaka koje se mogu pojaviti u primljenoj kodnoj riječi, a da prijemnik još *uvijek* može odrediti izvornu poruku.

***Primjer***: Procjena sposobnosti koda za ispravak pogrešaka u slučaja da se prenosi kodna riječ **a** = [11001100], a primi se poruka **m** = [11111100]. U korištenome kodu (koriste ga i odašiljač i prijemnik) ima još jedna kodna riječ **b** = [11111111]. Hammingova udaljenost između **a** i **m** je 2, a to je također udaljenost između **b** i **m**. Pretpostavimo da je zadana **m**. Uz spoznaju da u njoj postoje dvije pogreške, u odnosu na njenu poslanu inačicu, koja je važeća kodna riječ?

Nemoguće je utvrditi je li prenesena kodna riječ **a** ili **b**, budući se **m** može dobiti uvođenjem dviju pogrešaka, ili u **a** ili u **b** (i možda u neku treću kodnu riječ, čija je Hammingova udaljenost **m** jednaka 2). Odatle slijedi da je sposobnost ispravke pogrešaka koda manja od 2. Ako je između **a** i **b** Hammingova udaljenost jednaka 4, poruka **m** nalazi se točno u sredini između **a** i **b** (njihova udaljenost je 2 za obje kodne riječi). Stoga je nemoguće odrediti izvornu poruku.

*Ispravak pogrešaka moguć je samo ako je broj pogrešaka manji od pola minimalne Hammingove udaljenosti koda*, što osigurava da je primljena pogrešna poruka bliža izvornoj kodnoj riječi nego bilo koja druga kodna riječ. Sposobnost ispravke pogrešaka koda onda je najveći broj koji je manji od $d_{\min}/2$. Taj broj je $t = \lfloor (d_{\min}-1)/2 \rfloor$, gdje zagrade "$\lfloor$" i "$\rfloor$" označavaju *cjelobrojan dio* broja u zagradama, (npr., $\lfloor 1 \rfloor = 1$ i $\lfloor 1{,}5 \rfloor = 1$).

*Zaključak 2.3* Sposobnost ispravke pogreške koda je $\lfloor (d_{\min}-1)/2 \rfloor$ (oznake za cjelobrojnu vrijednost[38]).

Obzirom na zaključke 2.2 i 2.3, sposobnost otkrivanja/ispravke pogrešaka koda, ovisi o njegovoj minimalnoj Hammingovoj udaljenosti. Što je veća minimalna Hammingova udaljenost, veća je sposobnost otkrivanja i ispravke pogrešaka. Isti rezultat također znači da ako kod omogućuje ispravak do $t$ pogrešaka u kodnoj riječi, onda je minimalna Hammingova udaljenost koda najmanje $2t+1$.

$$t = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor \cdot 2 \Rightarrow 2t = d_{\min}-1 \Rightarrow d_{\min} = 2t+1 \qquad (2.1)$$

Minimalna Hammingova udaljenost za ($k$, $k$) kod, koji se sastoji od svih $2^k$ (varijacije s ponavljanjem od dvaju elemenata $k$-toga razreda) slučajnih vektora duljine $k$, jednaka je 1. Paritetni bitovi onda se mogu pridružiti svakome od tih vektora, zbog potrebe povećanja minimalne Hammingove udaljenosti. Ovi paritetni bitovi računaju se iz slučajnih bitova. Pridruživanjem sve više i više paritetnih bitova, moguće je povećati minimalnu Hammingovu udaljenost ovoga koda, zadržavajući stalan broj kodnih riječi. Ovaj postupak prikazuje tablica 2.1.

[38] Obilježavanje $\lceil x \rceil$ znači najmanji cio broj ne manji od $x$, a $\lfloor y \rfloor$ znači najveći cio broj ne veći od $y$.

*Tablica 2.1 Povećanje minimalne Hammingove udaljenosti koda dodavanjem paritetnih bitova*

| (a) | | (b) | | (c) (7, 4) kod | | | |
|---|---|---|---|---|---|---|---|
| 16 osnovnih riječi | | Dodavanje 1 paritetnoga bita | | Dodavanje 3 paritetna bita | | | |
| 0000 | 1000 | 00000 | 10001 | 0000 | 000 | 1000 | 110 |
| 0001 | 1001 | 00011 | 10010 | 0001 | 101 | 1001 | 011 |
| 0010 | 1010 | 00101 | 10100 | 0010 | 111 | 1010 | 001 |
| 0011 | 1011 | 00110 | 10111 | 0011 | 010 | 1011 | 100 |
| 0100 | 1100 | 01001 | 11000 | 0100 | 011 | 1100 | 101 |
| 0101 | 1101 | 01010 | 11011 | 0101 | 110 | 1101 | 000 |
| 0110 | 1110 | 01100 | 11101 | 0110 | 100 | 1110 | 010 |
| 0111 | 1111 | 01111 | 11110 | 0111 | 001 | 1111 | 111 |

U stupcu (a) tablice 2.1 navedeni su svi uzorci od 4 bita (16 složaja). Hammingova minimalna udaljenost je 1. Stupac (b) dobije se pridruživanjem dodatnoga bita parnosti na kraj riječi, tvoreći kodnu riječi ukupnoga pariteta 0. Hammingova minimalna udaljenost, ovdje je 2. Stupac (c) dobije se pridruživanjem triju paritetnih bitova riječima prvoga koda [stupac (a)]. Minimalna Hammingova udaljenost ovdje je 3. Način, kako se kod izgrađuje, objasnit će se u sljedećem poglavlju.

Napomena: Ovaj kod nije izgrađen poštujući jednadžbe za: $p_0$, $p_1$, $p_2$.[39] Nemojte se zavaravati. Da bi se postigla minimalna Hammingova udaljenost od 3, početnim slučajnim $k$ bitovima, nije uvijek dovoljno pridružiti samo 3 paritetna bita.

U sljedećoj tablici NISU zadovoljene paritetne jednadžbe: $p_0 = a \oplus b \oplus d$, $p_1 = a \oplus c \oplus d$, $p_2 = b \oplus c \oplus d$, nego samo minimalna Hammingova udaljenost od 3, između SVIH riječi, međusobno. Ovakvo kodiranje NEĆE imati željeni učinak na dekodiranje u dekoderu na prijemnoj strani.

| | $p_0$ | $p_1$ | $a$ | $p_2$ | $b$ | $c$ | $d$ | | | $p_0$ | $p_1$ | $a$ | $p_2$ | $b$ | $c$ | $d$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 000 | 1000 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 110 |
| 0001 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 101 | 1001 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 011 |
| 0010 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 111 | 1010 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 001 |
| 0011 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 010 | 1011 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 100 |
| 0100 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 011 | 1100 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 101 |
| 0101 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 110 | 1101 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 000 |
| 0110 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 100 | 1110 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 010 |
| 0111 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 001 | 1111 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 111 |

## 2.4. 2.4. Hammingovi kodovi

Moguće je generirati kod s minimalnom Hammingovom udaljenosti jednakoj 3 čija je kodna riječ duljine $2^m$–1 za bilo koji $m$, gdje se svaka kodna riječ sastoji od $2^m$–$m$–1 informacijskih bitova, a $m$ su paritetni bitovi. Također je poznato da je ovaj kod najučinkovitiji pri razmatranju brojnih paritetnih bitova potrebnih zbog pridruživanja informacijskome vektoru, da bi se postigla minimalna Hammingova udaljenost od 3.

Ovaj učinkovit kod naziva se *Hammingov kod*. Moguće dimenzije Hammingovih kodova su: (7, 4), (15, 11), (31, 26)[40], itd.. Budući da Hammingovi kodovi imaju $d_{min} = 3$, njihova sposobnost otkrivanja pogrešaka je 2, a njihova sposobnost ispravke pogrešaka je 1.

Izgradnja koda znači pretvorbu informacijskih vektora duljine $k$ u kodne riječi duljine $n$. To znači pridruživanje $n$–$k$ paritetnih bitova informacijskome vektoru, a koji se računaju iz informacijskih bitova. Međusobno različito raspoređeni informacijski i paritetni bitovi u kodnoj riječi, dat će drugačije Hammingove kodove.

Na primjer, kod u stupcu (c) u tablici 2.1 je Hammingov (7, 4) kod. Tablica 2.2 prikazuje još jedan Hammingov (7, 4) kod.

[39] Napomena: Hamming je paritetne bitove $p_0$, $p_1$ odnosno $p_2$, označio kao $P_1$, $P_2$ odnosno $P_3$ (brojanje je započeo od 1, a ne od 0 i koristio je velika slova. Poslije su se oznake promijenili iz $P$ u $p$. Slovo $p$ zbog lakšega pamćenja (grč. *mnemonikos*) upućuje na "paritet"), a indeksi započinju od 0, zbog prilagodbe binarnome označavanju (potencije započinju 0). Postoji još jedno odstupanje od izvornoga označavanja, jer je Hamming brojanje započinjao s lijeva u desno (što je prikladno za razumijevanje), a potencije binarnih polinoma u pravilu rastu s desne, prema lijevoj strani.

[40] Detalji će se obraditi na laboratorijskim vježbama

*Tablica 2.2 Hammingov (7, 4) kod ($p_0 = a \oplus b \oplus d$, $p_1 = a \oplus c \oplus d$, $p_2 = b \oplus c \oplus d$)*

| *Tablica 2.2 (7, 4) kod* | | | | | | | | | | *Tablica 2.1 (7, 4) kod* (zadovoljen je SAMO uvjet minimalne udaljenosti od 3) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Kodne riječi | | Raščlanjene kodne riječi | | | | | | | | | | | |
| | | $p_0p_1$ | $a$ | $p_2$ | $bcd$ | $p_0p_1$ | $a$ | $p_2$ | $bcd$ | | | | |
| 0000000 | 1110000 | 00 | 0 | 0 | 000 | 11 | 1 | 0 | 000 | 0000 | 000 | 1000 | 110 |
| 1101001 | 0011001 | 11 | 0 | 1 | 001 | 00 | 1 | 0 | 001 | 0001 | 101 | 1001 | 011 |
| 0101010 | 1011010 | 01 | 0 | 1 | 010 | 10 | 1 | 0 | 010 | 0010 | 111 | 1010 | 001 |
| 1000011 | 0110011 | 10 | 0 | 0 | 011 | 01 | 1 | 0 | 011 | 0011 | 010 | 1011 | 100 |
| 1001100 | 0111100 | 10 | 0 | 1 | 100 | 01 | 1 | 1 | 100 | 0100 | 011 | 1100 | 101 |
| 0100101 | 1010101 | 01 | 0 | 0 | 101 | 10 | 1 | 1 | 101 | 0101 | 110 | 1101 | 000 |
| 1100110 | 0010110 | 11 | 0 | 0 | 110 | 00 | 1 | 1 | 110 | 0110 | 100 | 1110 | 010 |
| 0001111 | 1111111 | 00 | 0 | 1 | 111 | 11 | 1 | 1 | 111 | 0111 | 001 | 1111 | 111 |

Legenda: **X** … informacijski bitovi,
**X** … paritetni bitovi

Kod u tablici 2.2 izgrađen je na sljedeći način: Informacijski vektor duljine $k$=4, čiji je opći oblik ($a$, $b$, $c$, $d$), pretvara se u kodnu riječ duljine 7 dodavanjem tri paritetna bita označena s $p_0$, $p_1$, $p_2$.

Informacijski bitovi i paritetni bitovi nalazi se u odgovarajućoj kodnoj riječi kako slijedi: $p_0\ p_1\ a\ p_2\ b\ c\ d$. Paritetni bitovi računaju se iz informacijskih bitova tako da zadovolje sljedeće jednadžbe:

| $p_0$ | $p_1$ | $a$ | $p_2$ | $b$ | $c$ | $d$ |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| (1) | 4 | – | 5 | – | 6 | – | 7 | | |
| | $p_2$ | $\oplus$ | $b$ | $\oplus$ | $c$ | $\oplus$ | $d$ | = | 0 |
| (2) | 2 | – | 3 | – | 6 | – | 7 | | |
| | $p_1$ | $\oplus$ | $a$ | $\oplus$ | $c$ | $\oplus$ | $d$ | = | 0 |
| (3) | 1 | – | 3 | – | 5 | – | 7 | | |
| | $p_0$ | $\oplus$ | $a$ | $\oplus$ | $b$ | $\oplus$ | $d$ | = | 0 |

(2.2)

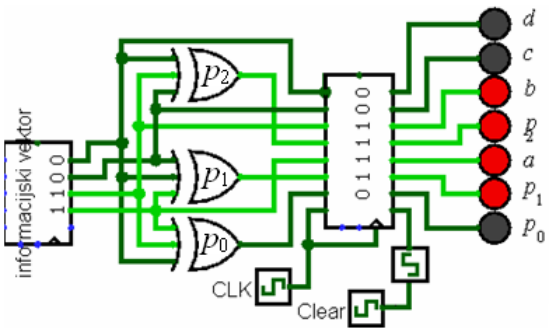
Simbol $\oplus$ označava izričito ILI zbrajanje (eXclusive OR). Obzirom na ono o čemu smo već pisali (oduzimanje ≡ zbrajanje), gornje tri jednadžbe možemo pisati kao:

$$p_2 = b \oplus c \oplus d,$$
$$p_1 = a \oplus c \oplus d,$$
$$p_0 = a \oplus b \oplus d.$$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| (1) | 4 | – | 5 | – | 6 | – | 7 | | |
| | $p_2$ | $\oplus$ | $b$ | $\oplus$ | $c$ | $\oplus$ | $d$ | = | 0 |
| (2) | 2 | – | 3 | – | 6 | – | 7 | | |
| | $p_1$ | $\oplus$ | $a$ | $\oplus$ | $c$ | $\oplus$ | $d$ | = | 0 |
| (3) | 1 | – | 3 | – | 5 | – | 7 | | |
| | $p_0$ | $\oplus$ | $a$ | $\oplus$ | $b$ | $\oplus$ | $d$ | = | 0 |

Popis 16 kodnih riječi u tablici 2.2 izradio se odabirom svih 16 mogućih informacijskih vektora duljine 4 (sve moguće permutacije $a$, $b$, $c$, $d$). Svakome takvome vektoru pridružila su se 3 paritetna bita, zadovoljavajući gore navedene tri jednadžbe. Položaj bitova u svakoj kodnoj riječi, prema ovome postupku je $p_0\ p_1\ a\ p_2\ b\ c\ d$. Sklop koji pretvara informacijski vektor u kodnu riječ, prikazuje slika 2.2.[41]

| *Tablica 2.2* | | | |
|---|---|---|---|
| ulaz | izlaz | ulaz | izlaz |
| 0000 | 0000000 | 1000 | 1110000 |
| 0001 | 1101001 | 1001 | 0011001 |
| 0010 | 0101010 | 1010 | 1011010 |
| 0011 | 1000011 | 1011 | 0110011 |
| 0100 | 1001100 | 1100 | 0111100 |
| 0101 | 0100101 | 1101 | 1010101 |
| 0110 | 1100110 | 1110 | 0010110 |
| 0111 | 0001111 | 1111 | 1111111 |

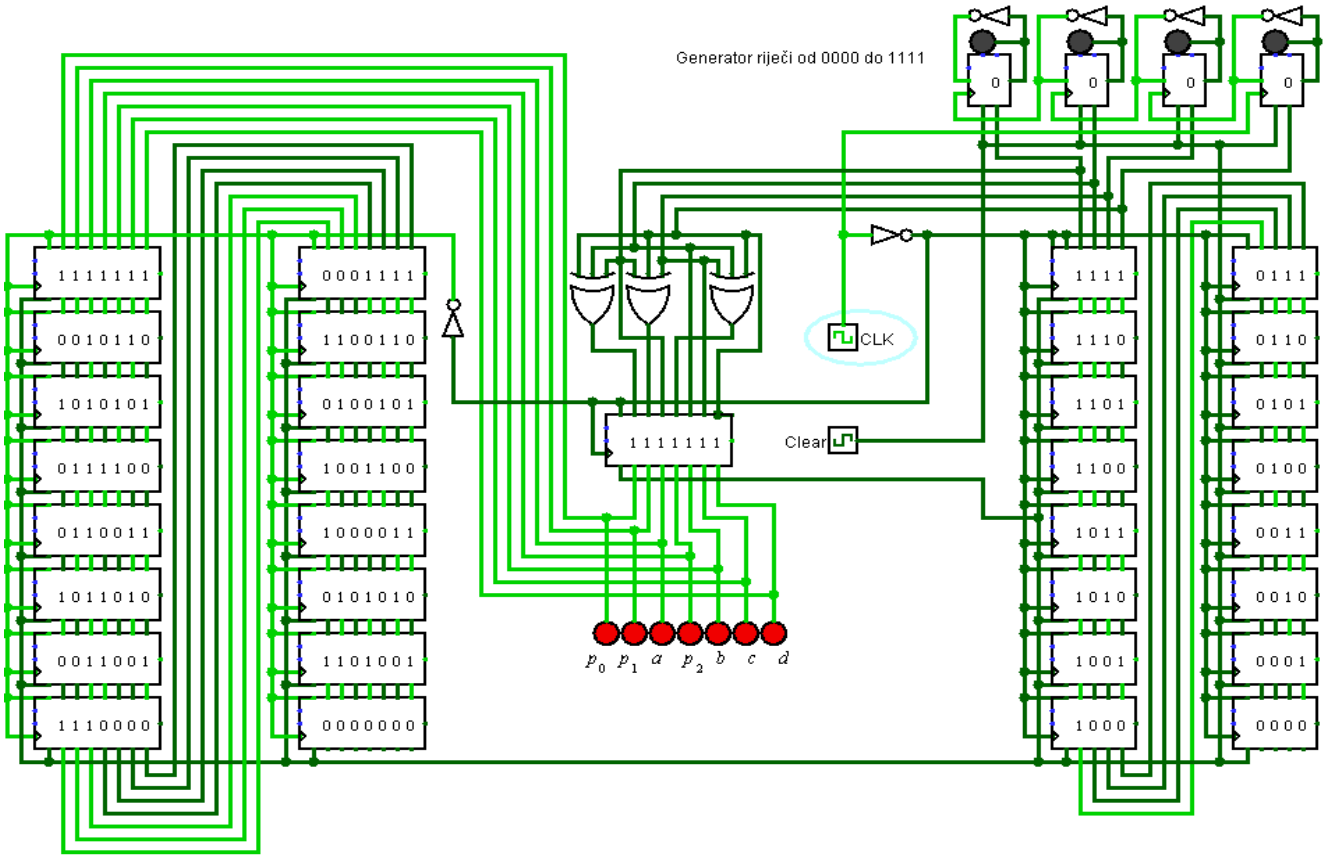



$p_0 = a \oplus b \oplus d,$
$p_1 = a \oplus c \oplus d,$
$p_2 = b \oplus c \oplus d.$

*Slika 2.2: Stvaranje kodnih riječi iz informacijskih vektora (Tablica 2.2)*[43]

[41] Napisati vježbu!
[43] Napraviti vježbu! Vježba uključuje i pripadni C++ program

Sklop na slici 2.2 automatski (korak po korak) generira sve kodne riječi Hammingova koda. Pridruženi paritetni bitovi dodani su informacijskim vektorima s desne strane (za tablicu 2.1). Plavo obojani informacijski vektori (stupci 1 i 3), predstavljaju $2^4 = 16$ varijacija s ponavljanjem od 2 elementa četvrtoga razreda. U tablici 2.2, redoslijed informacijskih i paritetnih bitova je obrnut (lijevo-desno).




*Slika 2.3: Sklop za generiranje tablice 2.2.*

[43] Napraviti vježbu! Vježba uključuje i pripadni C++ program

# 4. *Laboratorijska vježba 3: Koncept kodiranja i dekodiranja blok-kodovima. Hammingov (7, 4) kod - 1. dio*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*



Popis simulacijskih krugova:

Popis simulacijskih krugova:

Slika 2.1.circ, Slika 2.1_1.circ, Zbroji 2. riječi od 4 bita.circ, Slika matrica H.circ, Slika Count Down.circ, Hammingov sustavan (7, 4) kod.circ, Slika 3_4.circ, Slika 3.4.circ, Slika 3.4a.circ, Slika4.1ponovljena.circ, Hamming i Pretvorba BIN2DEK.circ,

Zbirke zadataka:

Konstrukcija, kodiranje i dekodiranje Hammingovoga (7, 4) koda.doc
Chapter 06 Channel Coding 1-ZADACI.doc
P08_BlockCodes_HR.doc

### 2.5.1. 2.5.1. OPTIMALNO KODIRANJE

Prema izrazu (1.39) u poglavlju o *uvjetima optimalnoga kodiranja*, izraz

$$\frac{H(U)}{\text{ld } L} \le \overline{n} < \frac{H(U)}{\text{ld } L} + \frac{1}{\nu} \tag{2.3}$$

jasno pokazuje da se prosječnim brojem simbola $\overline{n}$ možemo po volji približiti iznosu $\frac{H(U)}{\text{ld } L}$ samo ako su *blokovi bitova* dovoljno dugi. Prema izrazu, možemo postaviti osnovnu tvrdnju o kodiranju vijesti:

*Tvrdnja*: Ako se primijeni kodiranje dovoljno dugih blokova vijesti, onda se minimalan srednji broj kodnih (binarnih) simbola potrebnih za kodiranje jednoga simbola vijesti, može po volji približiti iznosu $H(U)/\text{ld } L$ (tj. odnosu prosječnoga iznosa informacije po jednome simbolu vijesti i kapacitetu abecede).

Iz gornje tvrdnje proizlazi mogućnost procjene ekonomičnosti nekoga koda, a zatim je na temelju nje moguće (pr)ocijeniti koliko se neki konkretan kod, svojom ekonomičnošću, približio optimalnome kodu. Kao najvažnije, *teorem o kodiranju* daje pravi fizikalan smisao *entropiji* elementarnih simbola vijesti, a to je minimalan broj binarnih simbola što ih se pri postupku kodiranja, u prosjeku pridružuje jednome od elementarnih simbola vijesti. Izraz će poprimit naročito jednostavan oblik za binarne sustave (kakvi se danas upotrebljavaju u praksi) tj. za $L = 2$:

$$H(U) \le \overline{n} < H(U) + \frac{1}{\nu} \tag{2.4}$$

Sasvim je jasno, da s motrišta tehničke realizacije sklopova za kodiranje, kodiranje blokova nije uvijek najprikladnije. Moramo uzeti u obzir činjenicu, da će koderi i dekoderi koji se primjenjuju za kodiranje blokova postati složeniji što je dužina bloka veća. Osim toga, na prijemnoj strani sustava, primjenom takvoga načina kodiranja, informacija nužno *kasni*, jer dekoder mora primiti čitav blok, da bi mogao dekodirati pojedine simbole u njemu.

Očito je da se mora tražiti ustupak između onoga što daju teorijske postavke i ekonomičnosti tehničke izvedbe uređaja. Teorem o kodiranju pokazuje dosljednu mogućnost najprikladnijega načina kodiranja i određuje graničnu ekonomičnost koda te metodiku sinteze najprikladnijega koda.

### 2.5.2. 2.5.2. METODE OPTIMALNOGA KODIRANJA

U sljedeća dva primjera pokazat ćemo dvije praktične metode za sintezu optimalnih kodova, Shannon-Fano metoda i Huffmanov algoritam kodiranja izvora.

#### 2.5.2.1. 2.5.2.1. Optimalna Shannon-Fano metoda kodiranja odijeljenih vijesti.

**Primjer 2.1.** Pretpostavimo da vijestima $u_i$ što ih treba kodirati, pripadaju apriorne vjerojatnosti $p(u_i)$. U tablici 2.3 vijesti su poredane po padajućim vjerojatnostima.

*Tablica 2.3. Optimalno kodiranje (Shannon-Fano metoda)*

| vijesti | | binarni znakovi | | | | | kodna riječ | | | | | broj znakova u kodnoj riječi | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $u_i$ | $p(u_i)$ | 1. | 2. | 3. | 4. | 5. | A | B | C | D | E | $n_i$ | $p(u_i)$ | | | | |
| $u_1$ | 1/2 | 0 | - | - | - | | 0 | | | | | 1 | 1/2 | | | | |
| $u_2$ | 1/4 | 1 | 0 | - | - | - | 1 | 0 | | | | 2 | | 1/4 | | | |
| $u_3$ | 1/8 | 1 | 1 | 0 | - | - | 1 | 1 | 0 | | | 3 | | | 1/8 | | |
| $u_4$ | 1/16 | 1 | 1 | 1 | 0 | - | 1 | 1 | 1 | 0 | | 4 | 1/2 | 1/4 | | 1/16 | |
| $u_5$ | 1/32 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 5 | | | 1/8 | | 1/32 |
| $u_6$ | 1/32 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 5 | | | | 1/16 | 1/32 |

Odredite binaran kod za kodiranje odijeljenih vijesti koji je najbliži optimalnome.

***Rješenje:*** Način rješavanja je sljedeći (Shannon-Fano metoda):

- Vijesti se poredaju po padajućim vjerojatnostima (vidi tablicu 2.3).

- Takav niz razdijeli se u dvije skupine tako da su *zbrojevi vjerojatnosti pojedinih vijesti* u obje skupine što je moguće bliži 1/2.
- Svim vijestima koje su u gornjoj skupini (stupac A) pridružimo simbol 0, što označuje *prvi znak* binarnoga koda.
- Svim vijestima koje su u drugoj skupini (stupac A) pridružimo simbol 1, što također označuje *prvi znak* binarnoga koda.
- Nadalje, svaku od ostalih skupina (od A do E) opet podijelimo na podskupine, tako da je zbroj vjerojatnosti u svim podskupinama približno jednak, pri tome se:
  - *gornjim podskupinama* u svakome stupcu pridružuje simbol 0, što označuje *drugi znak* binarnoga koda, a
  - *donjim podskupinama* pridružuje se simbol 1 što također ima značenje drugoga znaka.

Ovakva dioba nastavlja se tako dugo dok u svakoj podskupini ne ostane samo po jedna vijest pa im se pridružuje "0" (gornja vijest) odnosno "1" (donja vijest).

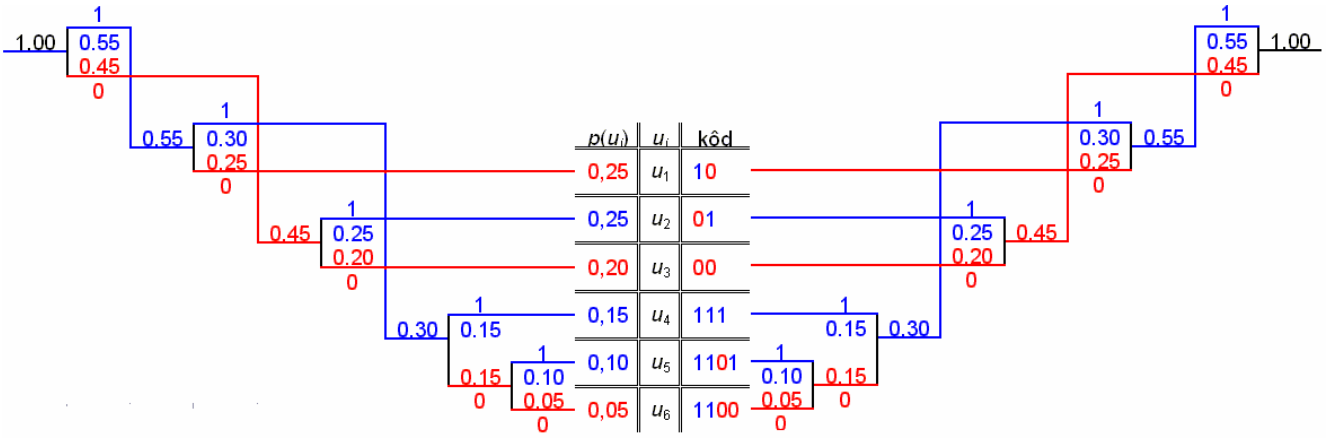
*2.5.2.2.* *2.5.2.2. Huffmanova metoda optimalnoga kodiranja odijeljenih vijesti.*

**Primjer 2.2.** Zadan je skup vijesti $u_i$ i njihove vjerojatnosti $p(u_i)$:

| vijest | $u_1$ | $u_2$ | $u_3$ | $u_4$ | $u_5$ | $u_6$ |
|---|---|---|---|---|---|---|
| vjerojatnost | 0,25 | 0,25 | 0,20 | 0,15 | 0,10 | 0,05 |

- Vijesti su poredane po padajućim vjerojatnostima. Treba pronaći optimalan kod za kodiranje odijeljenih vijesti iz zadanoga skupa.

***Rješenje:*** Primijenimo Huffmanovu metodu optimalnoga kodiranja (slika 2.4).



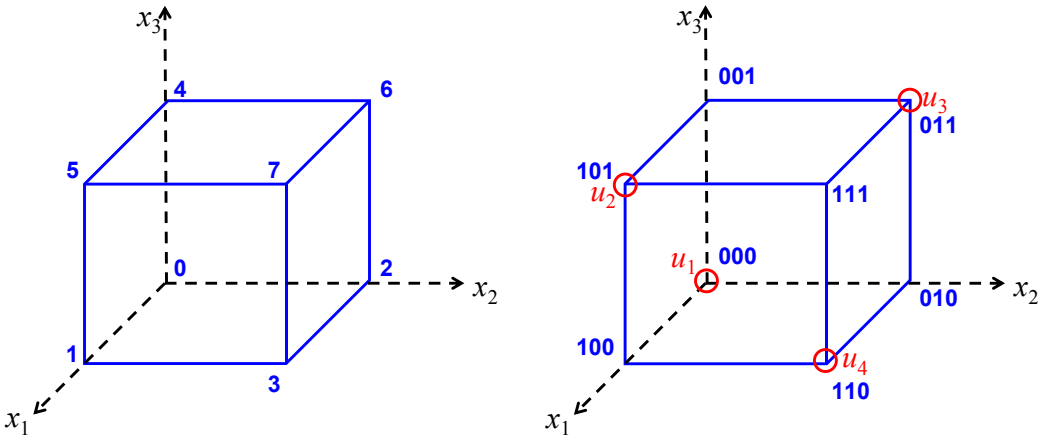

*Slika 2.4: Optimalno kodiranje Huffmanovom metodom*

Desna slika, predstavlja zrcalnu sliku lijevoga dijela dijagrama optimalnoga kodiranja. Dakle, svejedno je gdje smjestili tablicu (lijevo ili desno) i odakle započeli združivanje vjerojatnosti (s desna ili s lijeva).

Vijesti se poredaju po padajućim vjerojatnostima. Posljednje dvije (one s najmanjim vjerojatnostima) združe se u jednu skupinu, a njihove vjerojatnosti se zbroje. Nadalje, tako dobivena skupina promatra se kao jedna vijest time da se prema iznosu vjerojatnosti stavi na odgovarajuće mjesto. U tako dobivenome skupu opet se združuju dvije vijesti s najmanjim vjerojatnostima i dobivena nova vijest stavlja se na određeno mjesto po veličini. Čitav postupak ponavlja se dok zbroj vjerojatnosti ne bude jednak 1. Na takav način dobila se shema kao na slici 2.4.

Da bi došli do odgovarajućega koda za pojedine vijesti krenimo od gornjega lijevoga kuta, tako da uvijek gornju granu označimo "1" a donju "0". Broj grananja (–|) umanjen za 1 dat će nam broj binarnih simbola u kodnoj riječi za pojedine vijesti, a oznake grana bit će ujedno i oznaka odgovarajućega binarnoga simbola. Na slici 2.4 označene su kodne riječi za pojedine vijesti.

Zbog lakše predodžbe primijenit ćemo geometrijski prikaz kodova. U binarnome sustavu ($k = 2$) kodna riječ koja se sastoji od $k_u$ simbola (bitova), može se prikazati koordinatama $d_v$ (koje mogu imati vrijednosti 0 ili 1) u $k_u$ dimenzijskom prostoru $E_{K_u}$. Ako je $k_u = 3$, prostor se može sasvim pregledno prikazati u tri dimenzije. Ako s $x_1$, $x_2$ i $x_3$ označimo koordinate prostora $E_3$, onda se vrhovi kocke mogu označiti brojevima 0, 1, 2, …, 7. Ako svakome od tih brojeva pridijelimo normalan binarni ekvivalent, onda su npr. koordinate vrha 7: $x_1 = 1$, $x_2 = 1$ i $x_3 = 1$ (slika 2.5).




a) Geometrijski prikaz koda b) Sigurnosni kod s udaljenošću 2

*Slika 2.5: Geometrijski prikaz koda i sigurnosne udaljenosti*

Želimo li pomoću koda s $k_u = 3$ (bita) kodirati četiri vijesti, $N = 4 = \{u_1, u_2, u_3 \text{ i } u_4\}$, onda će u kodu postojati zalihost:

$$r = \frac{k_u - \text{ld } N}{\text{ld } N} = \frac{k_u - k}{k} \Rightarrow r = \frac{3-2}{2} = 0{,}5 = 50\% \qquad (2.5)$$

Naš cilj je iskoristiti tu zalihost od 50% za sigurniji prijenos informacija.

Gledano geometrijski, potrebno je na pogodan način odabrati one vrhove kocke, u koje ćemo smjestiti naše četiri vijesti. U tu svrhu uvedimo pojam *udaljenosti dvaju vrhova*, kao *najmanji broj bridova* koji ih dijele. Npr. udaljenost vrhova 1 i 7 na slici 2.5.a jednaka je 2. Udaljenost između dvaju vrhova označavat ćemo s $d_{ij}$. Ako sada pridijelimo informacijama one vrhove kocke čija udaljenost je 2, možemo načiniti kod u kojemu je moguće otkriti jednostruku pogrešku. Sasvim je jasno da se može načiniti više takvih kodova zavisno od toga koji vrh odaberemo kao početni. Jedna mogućnost prikazuje slika 2.5.b. U tablici pridružen je odgovarajući kod.

| vijest | $u_1$ | $u_2$ | $u_3$ | $u_4$ |
|---|---|---|---|---|
| kod | 000 | 101 | 011 | 110 |

Svojstvo $d_{ij} = 2$ u ovome slučaju znači, da se odabrane kodne kompleksije razlikuju u dvije znamenke. Drugim riječima, nastupi li prilikom prijenosa jednostruka pogreška u kodnoj riječi, time nastaje *nova kodna riječ*, koja nije pridodana ni jednoj drugoj vijesti (zalihosna riječ).

Npr. ako pri prijenosu vijesti $u_3$ nastupi pogreška, te se kodna kompleksija primi kao 001, prijemnik će otkriti pogrešku, ali neće moći ustanoviti koja je vijest ispravna, jer je primljena kompleksija 001 mogla nastati ili od 101 ili od 011 ili od 000 promjenom samo jednoga bita. Jasno, dvostruka pogreška u jednoj riječi pretvoriti će je u drugu, a ona također pripada jednoj od vijesti. Prema tomu takav kod omogućuje *otkrivanje* ali ne i *ispravak* jednostruke pogreške.

Razmotrimo slučaj, ako istim kodom želimo kodirati samo dvije vijesti. Onda je $r = 100\%$. U tomu slučaju možemo odabrati $d_{ij} = 3$. Sada će jedna pogreška u kodnoj riječi dati riječ koja je od izvorne udaljena za $d = 1$, a od svih drugih, koje su primijenjene za kodiranje (u ovomu slučaju to je samo jedna) za $d = 2$. Prema tomu, takav kod imat će mogućnost:

1. Ispraviti jednu pogrešku.

   Npr. odaberimo kod prema tablici:

| vijest | $u_1$ | $u_2$ |
|---|---|---|
| kod | 000 | 111 |

Ako je primljena riječ 101, očito je pravilna informacija 111, uz *pretpostavku da je nastupila samo jedna pogreška*, jer kompleksije 001 i 100 u odabranome kodu nemaju značenje vijesti.

2. Otkriti dvije pogreške, bez mogućnosti ispravke.
   Na primjer riječ 100 mogla je nastati od riječi 111 uz dvostruku pogrešku ili od riječi 000 uz jednostruku pogrešku. Prema tomu možemo samo otkriti pogreške, ali ne možemo ustanoviti ispravnu informaciju.

Svojstva kodova razmatrana uz $k_u = 3$ možemo primijeniti na opći slučaj. Dakako da je onda geometrijski prikaz složeniji i zahtijeva uvođenje više-dimenzijskih prostora. Općenito, ako je u $k_u$ dimenzijskome prostoru, $E_{K_u}$ udaljenost između dviju točaka koje predstavljaju niz od $N$ simbola 0, 1, …, $N$–1 poredanih u prirodnomu poretku, jednaka $d$, onda kodne kompleksije koje se koriste za kodiranje simbola, čine *pod-prostor* u kojemu je udaljenost između ma kojih dviju točaka jednaka $d$.

Za bilo koje dvije kodne riječi $M_i$ i $M_j$ vrijedi, da je $d_{ij} \geq d$. Ako u kodnoj riječi nastupi jedna pogreška, to će dovesti do pogrešne riječi $M_i^{'}$, udaljene od pravilne za 1, a od svih ostalih riječi za najmanje $d$–1. Očito je moguće otkriti jednostruku pogrešku čiji nastup dovodi do pojave točke koja je za $\alpha$ udaljena od riječi $M_i$, ako je ispunjen uvjet:

$$d - \alpha \geq 1. \tag{2.6}$$

Na taj način vidimo da je maksimalan broj pogrešaka koje možemo otkriti jednak $d$–1. Takav **kod** može dakle otkriti prisutnost $d$–1 pogrešaka, ali pri tomu može ispraviti samo $\frac{d}{2}-1$ pogrešku ako je $d$ paran, ili $\frac{d-1}{2}$ ako je $d$ neparan. Za neke vrijednosti udaljenosti, tablica pokazuje koliko je moguće pogrešaka otkriti i ispraviti.

| Minimalna vrijednost udaljenosti $d$ | Mogućnosti | |
|---|---|---|
| | otkrivanje | ispravak |
| 1 | 0 | 0 |
| 2 | 1 | 0 |
| 3 | 2 | 1 |
| 4 | 3 | 1 |
| 5 | 4 | 2 |

Iz gornjih razmatranja proizlazi, da je uvijek moguće binarno kodirati informacije takvim kodom, koji u sebi sadrži svojstva koja omogućuju otkrivanje i ispravak pogrešaka.

Postoji i niz drugih metoda za sintezu sigurnosnih kodova od kojih je najpoznatija Hammingova. Uvid u tu metodu najlakše se dobije preko primjera, zato će se ona primijeniti u primjerima u poglavlju 2.5.3.

### 2.5.3. 2.5.3. SINTEZA SIGURNOSNOGA HAMMINGOVOGA KODA

Sinteze dviju vrsta sigurnosnoga koda pokazuju sljedeći primjeri.

#### *2.5.3.1. 2.5.3.1. Otkrivanje i ispravak jednostruke pogreške.*

**Primjer 2.3.** Optimalno kodiranje vijesti iz primjera 2.2, dodavanjem odgovarajućih zalihosnih bitova za otkrivanje i ispravak jednostruke pogreške.

***Rješenje:*** Izvor informacija serijski generira moguće vijesti $u_1$ … $u_6$. Prema *veličini vjerojatnosti pojave* pojedine vijesti, koder informacija (slika 2.6) pridodaje svakoj vijesti *duži* ili *kraći* kod, koji se dobije, npr. prije pokazanim postupcima *optimalnoga* kodiranja.


*Slika 2.6: Uz primjer 2.3.*

Takav kod prikazuje tablica 2.4.

*Tablica 2.4. Vjerojatnosti pojave optimalno kodiranih vijesti*

| Moguće vijesti | Optimalan kod | Vjerojatnosti pojavljivanja |
|---|---|---|
| $u_1$ | 10 | 0,5 |
| $u_2$ | 01 | 0,25 |
| $u_3$ | 00 | 0,125 |
| $u_4$ | 111 | 0,0625 |
| $u_5$ | 1101 | 0,03125 |
| $u_6$ | 1100 | 0,03125 |
| Suma: $\Sigma$ = 1,00000 | | |

Promatramo li postupak prijenosa vijesti, onda za određen niz vijesti $u_i$ na ulazu kodera informacija, na njegovome izlazu pojavljuju se serijski "1" i "0" u odgovarajućemu redoslijedu. Na primjer:

| ulaz | $u_5$ | | | | $u_3$ | | $u_1$ | | $u_2$ | | $u_1$ | | $u_3$ | | $u_2$ | | $u_4$ | | | itd. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | |
| izlaz | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | itd. |

Iz ovakvoga niza "jedinica" i "nula" uvijek je moguće dekodirati primarne vijesti, budući je kod reverzibilan. Potrebno je jedino jednoznačno poznavati početak niza vijesti.

Znamo da su informacije pri prolazu kroz komunikacijski sustav podložne utjecaju smetnji naročito u prijenosnome sustavu. Smetnje mogu postati takve, da se pojedine jedinice pretvaraju u nule i nule pretvaraju se u jedinice. Pretpostavimo, da je u gornjemu primjeru umjesto predanoga niza, primljen sljedeći niz:

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | Redni broj bita | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 2. | Nazivi optimalno kodiranih informacija | $u_5$ | | | | $u_3$ | | $u_1$ | | $u_2$ | | $u_1$ | | $u_3$ | | $u_2$ | | $u_4$ | | | ? |
| 3. | Niz poslan na **ulaz kodera** | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | … |
| 4. | Prijenosni sustav | | | | | | | | | | | | | | | | | | | | |
| 5. | Niz primljen na **ulaz dekodera** | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | … |
| 6. | Nazivi optimalno kodiranih informacija | $u_5$ | | | | $u_2$ | | $u_1$ | | $u_2$ | | $u_1$ | | $u_3$ | | $u_2$ | | $u_1$ | | 1 | ? |

Obrnutim postupkom kodiranja (dekodiranje) prema tablici 2.4. dobili bi niz vijesti prikazan u petome (5.) retku. Vidimo, da se zbog toga što je nastala pogreška na dva mjesta (6, 18), primljen niz vijesti razlikuje od predanoga. Prema tomu postojeći kod nema nikakve mogućnosti suprotstaviti se smetnjama. Zbog toga, prije predaje prijenosnome sustavu, potrebno je prikladno osigurati informacije dodavanjem kodu novih elemenata.

### 2.5.3.2. 2.5.3.2. Udaljenost Hammingova koda

Richard Hamming, radeći u Bell Labs, 1947., pomoću preplitanja paritetnih bitova u kodnoj riječi, osmislio je način ne samo kako otkriti da se promijenila prenesena riječ, već koji se točno bit te riječi promijenio. Definirao je "udaljenost" između bilo kojih dviju riječi kao broj bitova u kojima se razlikuju te dvije riječi. U čast njegova rada, ovaj koncept nazvao se *Hammingova udaljenost*.

### 2.5.3.3. 2.5.3.3. Hammingova metoda paritetnoga ispitivanja koda

Ovdje ćemo primijeniti metodu tzv. paritetnoga ispitivanja koda (metoda po Hammingu). Pretpostavljamo, da se pogreške ne događaju suviše često. Neka je razmak između dviju pogrešaka barem sedam binarnih znakova. Onda se izvorni niz binarnih znakova podijeli u skupine (blokove) po četiri *bez obzira na granice pojedinih vijesti*:

| 1. | 2. | 3. | 4. | | |
|---|---|---|---|---|---|
| 1 1 0 1 | 0 0 1 0 | 0 1 1 0 | 0 0 0 1 | 1 1 1 1 | …… |

Tako se u svakome bloku nalaze po *četiri* informacijska bita $m_1$, $m_2$, $m_3$ i $m_4$. Dodavanjem kontrolnih bitova svakome bloku, proširit ćemo broj mjesta za $k$, na kojima također može nastupiti pogreška pri prijenosu. Zato je potrebno kontrolirati i informacijske i kontrolne bitove pomoću kontrolnih bitova. Prema tomu, kompleksije kontrolnih bitova moraju svojim dekadskim ekvivalentom moći označiti sva $m + k$ mjesta u jednome bloku. Zbog toga je broj kontrolnih bitova:

$$k \geq \mathrm{ld}\,(m + k + 1)\,. \qquad (2.7)$$

Kompleksija kontrolnih bitova sastavljena od samih "nula", označavat će da pogreška nije nastupila. Za naš slučaj:

$$k \geq \mathrm{ld}\,(4+k+1) \rightarrow k = 3$$

Prema tomu s 3 bita možemo obuhvatiti sva informacijska i kontrolna mjesta ($2^3 = 8$):

| | informacijska i kontrolna mjesta | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Kontrolni bitovi | 0. | 1. | 2. | 3. | 4. | 5. | 6. | 7. |
| $p_0$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| $p_1$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $p_2$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

U svrhu kontrole primijenit ćemo svojstvo parnosti zbroja binarnih znamenaka "1" (jedinica) na određenim mjestima. Zbog toga, potrebno je na predajnoj strani kanala prilagoditi kod tako, da *zbroj po modulu* 2 između znamenaka na odabranim mjestima bude *jednak nuli*. Kontrolni bit $p_0$ može nam odrediti pogreške na mjestima 1, 3, 5, 7, $p_1$ na mjestima 2, 3, 6, 7 i $p_2$ na mjestima 4, 5, 6, 7.

Tomu treba prilagoditi i primaran kod tako, da se pri kodiranju, mjestima $p_0$, $p_1$ i $p_2$ na *predajnoj strani*, dodaju kontrolni bitovi pa je ***zbroj po modulu 2 uvijek jednaka nuli***. Zbog toga, potrebno je na odgovarajući način razmjestiti informacijske i kontrolne bitove, da se pri svakoj kontroli, među informacijskim bitovima nađe *samo jedan kontrolni bit*. Za naš primjer, razmještaj bitova prikazuje sljedeća tablica:

| Blokovi od 4 bita | 1. $p_0$ | 2. $p_1$ | 3. $m_1$ | 4. $p_2$ | 5. $m_2$ | 6. $m_3$ | 7. $m_4$ | |
|---|---|---|---|---|---|---|---|---|
| 1. blok | | | 1 | | 1 | 0 | 1 | |
| Kontrola 1-3-5-7 | 1 | | 1 | | 1 | | 1 | |
| Kontrola 2-3-6-7 | | 0 | 1 | | | 0 | 1 | ∑po modulu 2 |
| Kontrola 4-5-6-7 | | | | 0 | 1 | 0 | 1 | |
| Novi kod za 1. blok | 1 | 0 | 1 | 0 | 1 | 0 | 1 | |
| | | | | | | | | |
| 2. blok | | | 0 | | 0 | 1 | 0 | |
| Kontrola 1-3-5-7 | 0 | | 0 | | 0 | | 0 | |
| Kontrola 2-3-6-7 | | 1 | 0 | | | 1 | 0 | ∑po modulu 2 |
| Kontrola 4-5-6-7 | | | | 1 | 0 | 1 | 0 | |
| Novi kod za 2. blok | 0 | 1 | 0 | 1 | 0 | 1 | 0 | |
| | | | | | | | | |
| 3. blok | | | 0 | | 1 | 1 | 0 | |
| Kontrola 1-3-5-7 | 1 | | 0 | | 1 | | 0 | |
| Kontrola 2-3-6-7 | | 1 | 0 | | | 1 | 0 | ∑po modulu 2 |
| Kontrola 4-5-6-7 | | | | 0 | 1 | 1 | 0 | |
| Novi kod za 3. blok | 1 | 1 | 0 | 0 | 1 | 1 | 0 | |
| | | | | | | | | |
| 4. blok | | | 0 | | 0 | 0 | 1 | |
| Kontrola 1-3-5-7 | 1 | | 0 | | 0 | | 1 | |
| Kontrola 2-3-6-7 | | 1 | 0 | | | 0 | 1 | ∑po modulu 2 |
| Kontrola 4-5-6-7 | | | | 1 | 0 | 0 | 1 | |
| Novi kod za 4. blok | 1 | 1 | 0 | 1 | 0 | 0 | 1 | |

Na mjesto svakoga od 3 kontrolna bita, pri kodiranju stavi se znamenka ("0" ili "1"). Ona osigurava da zbroj znamenaka po modulu 2 za svaki od kontrolnih bitova ($p_0$, $p_1$, $p_2$), daje nulu (prema prije navedenim shemama). Sve tri kontrole daju vrijednosti kontrolnih bitova pa se nov (proširen) kod za svaki blok sada sastoji od 7 elemenata. Prema tomu, umjesto prvobitnoga niza od 4 bita, za koji smo vidjeli da je neotporan na smetnje, na prijenosni sustav predajemo nov niz u blokovima od po 7 bitova:

| Blokovi od 7 bitova | | | | |
|---|---|---|---|---|
| 1. | 2. | 3. | 4. | |
| 1010101 | 0101010 | 1100110 | 1101001 | …… |

Dobiven niz znatno je duži od prvobitnoga, ali zato on u sebi sadrži mogućnost otkrivanja i ispravke *jednostruke* pogreške.

Kako se otkrivaju i ispravljaju pogreške? Neka je npr. pod utjecajem smetnji pri prolazu kroz prijenosni sustav, nastupila pogreška na 5. mjestu 2. bloka, te je na prijemnu stranu umjesto predanoga niza stigao niz:

1010101 0101110 1100110 1101001 ….

Prijemnik dakle, ne može unaprijed znati da je nastupila pogreška pa provjerava tako da se prijemni niz razbije na blokove od po 7 znakova i u svakome bloku provodi se provjera pariteta. Provjerava se sadrže li sva tri pariteta, paran broj jedinica računajući sva odabrana kontrolna mjesta:

| funkcija | 1. | 2. | 3. | 4. | 5. | 6. | 7. | Rezultat kontrole |
|---|---|---|---|---|---|---|---|---|
| Primljen 1. blok | 1 | 0 | 1 | 0 | 1 | 0 | 1 | |
| kontrola 1-3-5-7 | 1 | | 1 | | 1 | | 1 | 0 |
| kontrola 2-3-6-7 | | 0 | 1 | | | 0 | 1 | 0 |
| kontrola 4-5-6-7 | | | | 0 | 1 | 0 | 1 | 0 |
| Informacija sadržana u 1. bloku | | | 1 | | 1 | 0 | 1 | Rezultat kontrole je: 000 |

Dakle, prvi blok ispravno je primljen. Kontrolni bitovi mogu se sada odbaciti i informacija sadržana u prvome bloku glasi: 1 1 0 1. Sada se pristupa kontroli 2. bloka.

| funkcija | mjesto | 1. | 2. | 3. | 4. | 5. | 6. | 7. | Rezultat kontrole | |
|---|---|---|---|---|---|---|---|---|---|---|
| Primljen 2. blok | | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | |
| kontrola 1-3-5-7 | | 0 | | 0 | | 1 | | 0 | 1 | ↓ ≡ $1 \times 2^0$ |
| kontrola 2-3-6-7 | | | 1 | 0 | | | 1 | 0 | 0 | ↓ ≡ $0 \times 2^1$ |
| kontrola 4-5-6-7 | | | | | 1 | 1 | 1 | 0 | 1 | ↓ ≡ $1 \times 2^2$ |
| Primljen 2. blok | | 0 | 1 | 0 | 1 | 1 | 1 | 0 | Računa se odozgo prema dolje kao: $1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 = 1 + 0 + 4 = 5$ Pogreška je na 5. mjestu! | |
| Ispravljen 2. blok | | 0 | 1 | 0 | 1 | 0 | 1 | 0 | | |
| Ispravna korisna informacija | | | | 0 | | 0 | 1 | 0 | | |
| Informacija sadržana u 2. bloku | | 0 | 0 | 1 | 0 | | | | | |

Kao što se vidi, za 2. blok ne zadovoljavaju kontrole pariteta 1-3-5-7 i 4-5-6-7, te kao rezultat za te kontrole bilježimo 1. Rezultat kontrole čitan *odozgo* prema *dolje*, daje redni broj mjesta na kojemu je nastupila pogreška. Budući da je sustav binaran, čim saznamo mjesto gdje je nastupila pogreška, možemo ju odmah i ispraviti. Znamenku koja se nalazi na tomu mjestu zamijenimo njezinim komplementom. U našemu primjeru, ispravno je da na 5. mjestu bude 0, a ne 1. Kontrola prema istomu postupku provodi se za svaki blok.



**Primjer 2.4.** Neka je informacija pisanoga jezika sa zalihosti izvora prikazana tekstom:

VIJEST_TREBA_PRENOSITI_BEZ_POGREŠKE_

- Najprije treba optimalno kodirati taj niz, a zatim primjenom Hammingova koda osigurati točan prijenos vijesti.

***Rješenje:*** Stvaran sadržaj te informacije dobit će se korištenjem optimalnoga koda za hrvatski jezik, gdje je zalihost sukladna *frekvencijama* znakova dobivenih analizama tekstovima hrvatskih pisaca (tablica 2.5).

*Tablica 2.5. Optimalan kod za hrvatski jezik.*

| | 1 | 2 | 3 | 4 | 5 | | | 1 | 2 | 3 | 4 | 5 | 6 | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Rk[44] | 0 | 1 | 0 | | | | T | 0 | 0 | 1 | 0 | 1 | | | B | 0 | 1 | 1 | 1 | 0 | 1 | | |
| A | 1 | 0 | 1 | | | | U | 0 | 0 | 0 | 1 | 0 | | | Z | 1 | 1 | 0 | 1 | 0 | 1 | | |
| E | 1 | 1 | 1 | | | | M | 1 | 0 | 0 | 1 | 0 | | | Š | 0 | 0 | 0 | 1 | 1 | 1 | 0 | |
| O | 0 | 0 | 0 | 0 | | | D | 0 | 1 | 1 | 1 | 1 | | | Č | 0 | 0 | 0 | 1 | 1 | 1 | 1 | |
| I | 0 | 1 | 1 | 0 | | | V | 1 | 0 | 0 | 1 | 1 | | | C | 0 | 0 | 0 | 1 | 1 | 0 | 0 | |
| N | 1 | 1 | 0 | 0 | | | L | 1 | 0 | 0 | 0 | 0 | | | H | 0 | 0 | 0 | 1 | 1 | 0 | 1 | |
| J | 0 | 0 | 1 | 1 | 0 | | K | 1 | 0 | 0 | 0 | 1 | | | Ž | 1 | 1 | 0 | 1 | 0 | 0 | 0 | |
| S | 0 | 0 | 1 | 1 | 1 | | P | 1 | 1 | 0 | 1 | 1 | | | Ć | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| R | 0 | 0 | 1 | 0 | 0 | | G | 0 | 1 | 1 | 1 | 0 | 0 | | F | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

Zadan tekst, kodiran ovim optimalnim kodom, izgleda ovako:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| V | | | | | I | | | | J | | | | | E | | | | S | | | | | T | | | | | razmak | | |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| T | | | | | R | | | | E | | | | | B | | | | | A | | | razmak | | | | P | | | |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| R | | | | | E | | | | N | | | | | O | | | | S | | | | | I | | | | | T | | |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| I | | | | razmak | | | | | B | | | | | E | | | | Z | | | | | razmak | | | | P | | | |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| O | | | | | G | | | | | R | | | | | E | | | | | Š | | | | | K | | | | |
| 1 | 1 | 1 | 0 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | |
| E | | | | razmak | | | | | | | | | | | | | | | | | | | | | | | | | |

Ako napišemo ovako skraćen stvaran sadržaj kodirane informacije na izvoru prema međunarodnome ravnomjernome kodu broj 2 (tablica 2.6), onda optimalno kodiran tekst ima ovaj niz znakova:

BIPWWRH $R_k$Q brojke R brojke razmak slova komb.32 OFHIBXQR brojke komb.32 U $R_k$UC $R_k$ZQ

*Tablica 2.6. Međunarodni ravnomjeran kod broj 2*

| ZNAK | 12345 kod | | ZNAK | 12345 kod | | ZNAK | 12345 kod | | ZNAK | 12345 kod |
|---|---|---|---|---|---|---|---|---|---|---|
| A | 11000 | | I | 01100 | | Q | 11101 | | Y | 10101 |
| B | 10011 | | J | 11010 | | R | 01010 | | Z | 10001 |
| C | 01110 | | K | 11110 | | S | 10100 | | | 00010 |
| D | 10010 | | L | 01001 | | T | 00001 | | | 01000 |
| E | 10000 | | M | 00111 | | U | 11100 | | Slova | 11111 |
| F | 10110 | | N | 00110 | | V | 01111 | | Brojke | 11011 |
| G | 01011 | | O | 00011 | | W | 11001 | | Razmak | 00100 |
| H | 00101 | | P | 01101 | | X | 10111 | | Komb.32 | 00000 |

U svrhu prilagodbe informacije kanalu odnosno prijenosnome mediju, informaciju ćemo kodirati kodom koji je sposoban otkriti i ispraviti pogreške prema Hammingu. Napravimo to za kod od 4 elementa (tablica 2.7).

*Tablica 2.7. Hammingov kod za blokove od četiri elementa.*

| | 1. | 2. | 3. | 4. | | 1. | 2. | 3. | 4. | 5. | 6. | 7. | | 1. | 2. | 3. | 4. | | 1. | 2. | 3. | 4. | 5. | 6. | 7. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | 0 | 0 | 0 | 0 | → | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9. | 1 | 1 | 1 | 1 | → | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2. | 0 | 0 | 0 | 1 | → | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 10. | 1 | 1 | 1 | 0 | → | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 3. | 0 | 0 | 1 | 0 | → | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 11. | 1 | 1 | 0 | 1 | → | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 4. | 0 | 0 | 1 | 1 | → | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 12. | 1 | 1 | 0 | 0 | → | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 5. | 0 | 1 | 0 | 0 | → | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 13. | 1 | 0 | 1 | 1 | → | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 6. | 0 | 1 | 0 | 1 | → | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 14. | 1 | 0 | 1 | 0 | → | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 7. | 0 | 1 | 1 | 0 | → | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 15. | 1 | 0 | 0 | 1 | → | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 8. | 0 | 1 | 1 | 1 | → | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 16. | 1 | 0 | 0 | 0 | → | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

[44] Rk … razmak između riječi

Koriste se sljedeća pravila:

Informacijski elementi dolaze na 3., 5., 6. i 7. mjesto Hammingova koda, a 1., 2. i 4. mjesto ispunit ćemo korisnom zalihosti, tj. kontrolnim bitovima ($p_0$, $p_1$ i $p_2$). Pri sastavljanju novoga koda, primijenit ćemo istu metodu kao u primjeru 2.3.

Na mjesto 4. bita (kontrolni bit) upisujemo 1 ili 0 tako da je:

- zbroj znamenaka na mjestima 4, 5, 6 i 7, paran.

Zatim prelazimo na 2. mjesto, tj.

- zbroj znamenaka na mjestima 2, 3, 6 i 7 također je paran.

I konačno izračun za 1. mjesto daje

- zbroj znamenaka na mjestima 1, 3, 5 i 7 isto tako je paran.

**Hammingov kod za blokove od četiri elementa iz tablice 2.6. i gornjega algoritma**

Pseudo kod:
Od 4 bita ukupno se može napraviti 16 riječi
Najprije treba generirati tih 16 riječi
U for() petlji svakoj od riječi treba pridružiti paritetne bitove prema algoritmu:
$p_0 = \Sigma(1 + 3 + 5 + 7) = 0$
$p_1 = \Sigma(2 + 3 + 6 + 7) = 0$
$p_2 = \Sigma(4 + 5 + 6 + 7) = 0$
Nakon izračunatoga pariteta, treba ga pridružiti odgovarajućemu mjestu te proširiti kodnu riječ

Na potpuno jednak način kontrolira se primljen niz elemenata u prijemniku. Trostruka kontrola toga niza od sedam elemenata, daje potreban podatak o mjestu pogreške pa je tako omogućen ispravak. Sada kodirajmo tekst:

VIJEST_TREBA_PRENOSITI_BEZ_POGREŠKE_

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | | | | | | | | | | | | |

i razdjelom ga u skupine od četiri elementa.

| blok | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 1. | 2. | 3. | 4. | 5. | 6. | 7. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | 1 | 0 | 0 | 1 | | | | 1 | 0 | 1 | 1 | | | | 0 | 0 | 0 | 1 | | | | 1 | 0 | 1 | 1 | | | |
| | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| | | 1 | | | | | | | | | | | 0 | | | | | | | | | | | | | 1 | | |
| 5. | 1 | 0 | 0 | 1 | | | | 1 | 1 | 0 | 0 | | | | 1 | 0 | 1 | 0 | | | | 1 | 0 | 0 | 0 | | | |
| | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| | | | | | | | | | | 0 | | | | | | | | | | 0 | | | | | | | | |
| 9. | 1 | 0 | 1 | 0 | | | | 0 | 1 | 0 | 0 | | | | 1 | 1 | 1 | 0 | | | | 1 | 1 | 1 | 0 | | | |
| | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| | | | | | 1 | | | | 1 | | | | | | | | 0 | | | | | | | | | | | |
| 13. | 1 | 1 | 0 | 1 | | | | 0 | 1 | 0 | 1 | | | | 1 | 0 | 1 | 1 | | | | 0 | 0 | 1 | 0 | | | |
| | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| | | | | 1 | | | | | | | | | | | | | | | | 0 | | | | | | | | |
| 17. | 0 | 1 | 1 | 1 | | | | 1 | 1 | 0 | 0 | | | | 0 | 0 | 0 | 0 | | | | 0 | 0 | 1 | 1 | | | |
| | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| | | | | | | | 0 | | | | | | | | | | | 1 | | | | | | | | 1 | | |
| 21. | 1 | 0 | 1 | 1 | | | | 0 | 0 | 0 | 1 | | | | 0 | 1 | 0 | 1 | | | | 1 | 0 | 0 | 1 | | | |
| | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| | | | | | 1 | | | | | | | | | | | | | | 0 | | | | | | | | | |
| 25. | 0 | 0 | 1 | 1 | | | | 1 | 0 | 1 | 1 | | | | 1 | 1 | 1 | 1 | | | | 0 | 1 | 0 | 1 | | | |
| | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| | | | | 1 | | | | | | | | 1 | | | | | | | | 0 | | | | | | | | |
| 29. | 0 | 1 | 0 | 1 | | | | 1 | 0 | 1 | 1 | | | | 0 | 0 | 0 | 0 | | | | 0 | 1 | 1 | 1 | | | |
| | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| | | | 1 | | | | | | | | | 1 | | | | | | | | | | | | | | | 0 | |
| 33. | 0 | 0 | 0 | 0 | | | | 1 | 0 | 0 | 1 | | | | 1 | 1 | 0 | 0 | | | | 0 | 1 | 1 | 1 | | | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

| blok | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 1. | 2. | 3. | 4. | 5. | 6. | 7. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 37. | 0 | 1 | 0 | 0 | | | | 0 | 1 | 1 | 1 | | | | 1 | 0 | 1 | 0 | | | | | | | | | | |
| | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | | | | | | | |

| | 1. | 2. | 3. | 4. | 5. | 6. | 7. | Rezultat kontrole |
|---|---|---|---|---|---|---|---|---|
| Primljen 29. blok | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| kontrola 1-3-5-7 | 0 | | 1 | | 1 | | 1 | 1 ↓ ≡ $1 \times 2^0$ |
| kontrola 2-3-6-7 | | 1 | 1 | | | 0 | 1 | 1 ↓ ≡ $1 \times 2^1$ |
| kontrola 4-5-6-7 | | | | 0 | 1 | 0 | 1 | 0 ↓ ≡ $0 \times 2^2$ |
| Primljen 29. blok | 0 | 1 | X | 0 | 1 | 0 | 1 | Računa se odozgo prema dolje kao: |
| Ispravljen kod u 29. bloku | 0 | 1 | 0 | 0 | 1 | 0 | 1 | $1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 = 1 + 2 + 0 = 3$ |
| Ispravna korisna informacija | | | 0 | | 1 | 0 | 1 | |
| Informacija sadržana u 29. bloku | | | 0 | | 1 | 0 | 1 | Pogreška je bila na 3. mjestu! |

U ovome nizu, pri prijenosu podataka, nastale su 24 pogreške. Prikazan niz prema međunarodnome kodu broj 2 sadrži ove znakove:

N G M S F I W V H Y A G R I H

| Z | P | R | S | Y | B | R | E | K | K | Kombinacija Σ 32 |
|---|---|---|---|---|---|---|---|---|---|---|
| O | I | K | D | D | B | N | O | I | slova K D W | |
| E | F | J | I | M | A | | | brojke | E slova | |

Nastupe li pogreške označene prekriženim elementima u tablici od 39 blokova i to samo na jednome od sedam elemenata u bloku, ovaj niz u prijemniku, izmijenio bi se u sljedeće znakove:

| C | G | O | S | F | U | W | G | H | Z | A | G | brojke | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| I | T | Z | P | C | S | Y | Z | R | E | U | K | T | M |
| P | K | D | E | B | N | G | P | | | J | D | J | E |
| D | J | I | O | W | | | B | E | Q | | | | |

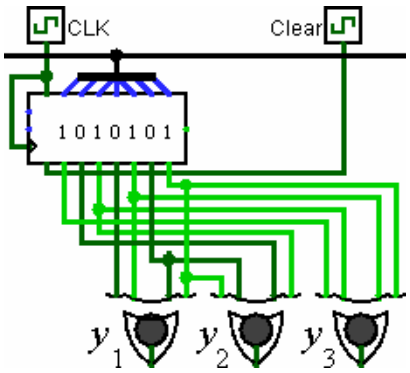
Izbrojimo li pogreške u tekstu od 49 znakova u gornjoj tablici, bez ispravke dolazimo do poraznoga rezultata od 23 pogreške. Ako međutim ispravljamo Hammingovom metodom onda će nestati sve pogreške, a tekst je, bez korisne zalihosti, jednak stvarnome sadržaju informacije izvora. Nakon odbacivanja suvišnih elemenata konačno pretvaramo informaciju u izvorni tekst tj.:

VIJEST_TREBA_PRENOSITI_BEZ_POGREŠKE_

*Definicija* **Kodiranje** je postupak koji se provodi u predajniku, gdje se računaju paritetni bitovi i vezuju se uz informacijski vektor (pa tako oblikuju kodnu riječ za slanje). Krug što kodira, zove se **koder.**

Neka je **a** kodna riječ koja se prenosi i neka je **m** njena primljena inačica. Prijemnik će pokušati otkriti je li došlo do pogreške tijekom prijenosa provjeravajući vrijede li za **m**, jednadžbe 2.2 [(1), (2) i (3)]. Ako vrijede, onda je **m** kodna riječ. Sada ćemo pokazati kako ispraviti 1 pogrešku koja se dogodila u **a** (tj., **m** se razlikuje od **a** u jednome bitu). Bitovi u **a** $p_0\ p_1\ a\ p_2\ b\ c\ d$, zadovoljavaju jednadžbe 2.2 [(1), (2) i (3)].

Neka su odgovarajući bitovi primljene poruke **m** [$p$, $q$, $r$, $s$, $t$, $u$, $v$]. Jedan od bitova u **m** je komplement odgovarajućega bita u **a**, a ostatak bitova je isti kao bitovi u **a**. Svaka od sedam mogućnosti "posjedovanja" jedne pogrešku u **m**, ima drugačiji utjecaj na vrijednosti $y_1$, $y_2$, $y_3$ (na slici 2.7).




*Slika 2.7: Utjecaj jedne pogrešku u **m** na vrijednosti $y_1$, $y_2$, $y_3$*

Tablica 2.8 matematički opisuje sliku 2.7:

*Tablica 2.8 Učinak jedne pogreške u* **m** *na vrijednosti $y_1$, $y_2$, $y_3$*[45]

| Pogrešni bitovi u **m** | Odgovarajuće vrijednosti za $y_1\ y_2\ y_3$ | | |
|---|---|---|---|
| *p* | 001 | $2^0$ | 1 |
| *q* | 010 | $2^1$ | 2 |
| *r* | 011 | $2^1 \oplus 2^0$ | 3 |
| *s* | 100 | $2^2$ | 4 |
| *t* | 101 | $2^2 \oplus 2^0$ | 5 |
| *u* | 110 | $2^2 \oplus 2^1$ | 6 |
| *v* | 111 | $2^2 \oplus 2^1 \oplus 2^0$ | 7 |

To znači da je na temelju $y_1$, $y_2$, $y_3$, moguće saznati koji je pogrešan bit u slučaju da je došlo do pojave jedne pogreške. Slučaj bez pogreške je označen kao $[y_1\ y_2\ y_3] = [000]$.

Slijedi, da je sposobnost ispravke pogreške koda najmanje 1. Tablica 2.8 objedinjuje učinak jedne pogreške u **m**, ispisom vrijednosti $y_1$, $y_2$, $y_3$ generirane "oštećenim" vektorom **m**.

### 2.5.4. 2.5.4. ISPITIVANJE PRIPADNOSTI BINARNOGA VEKTORA SKUPU KODNIH RIJEČI

Pokazat ćemo da sposobnost ispravke pogreške koda nije veća od 1. Konačan cilj rasprave u ovome dijelu poglavlja jest pronaći $d_{min}$. Razmotrimo npr., slučaj gdje su pogrešni bitovi *p* i *q* u **m**. Unosom sadržaja vektora **m** u krug na slici uz tablicu 2.8, dobivamo $y_1 = 0$, $y_2 = 1$, $y_3 = 1$.

Iz tablice 2.8 vidi se da će jedna pogreška u *r* dovesti do iste vrijednosti $y_1$, $y_2$, $y_3$. To znači da je prijemniku nemoguće otkriti je li se u **m** pojavila jednostruka ili dvostruka pogreška. Sposobnost ispravke pogreške koda je, dakle, 1, a zbog već spomenutoga, $d_{min}$ je ili 3 ili 4.

Pokažimo sada da je $d_{min}$ jednako 3, a ne 4, razmatranjem sposobnosti *otkrivanja* pogreške koda. Uzmimo slučaj gdje su bitovi *p*, *q* i *r* u **m** pogrešni, a ostali nisu. U ovome slučaju može se potvrdili da vrijede jednadžbe 2.2 [(1), (2) i (3)], (tj., $y_1 = y_2 = y_3 = 0$.), tako da nema otkrivanja pogrešaka. To također znači da je **m** valjana kodna riječ.

Budući smo pokazali da se pojava tri pogreške ne može uvijek otkriti, sposobnost otkrivanja pogreške koda manja je od 3. Imamo da je $d_{min}$, čija vrijednost je za 1 veća od sposobnosti otkrivanja pogrešaka, manja od 4. Ako se iz sposobnosti *ispravke* pogreške koda pokazalo da je $d_{min}$ ili 3 ili 4, onda slijedi da je $d_{min} = 3$.

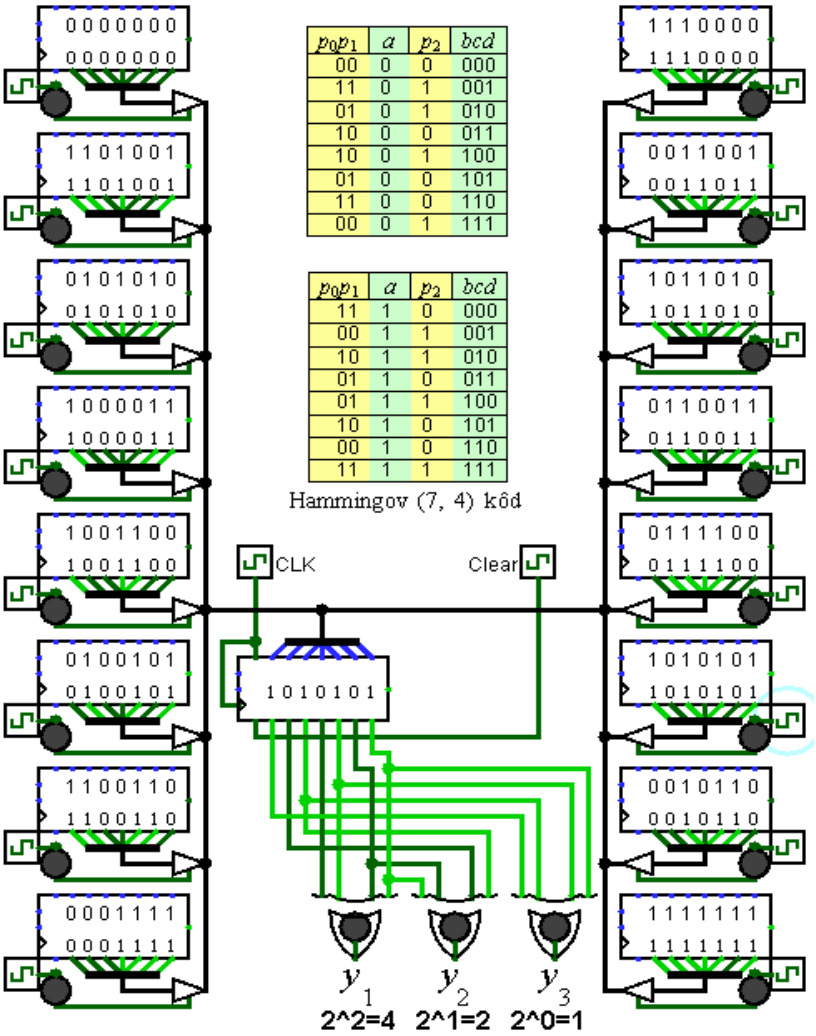
Ovdje je važno razjasniti povezanost kodova za otkrivanje/ispravak pogrešaka. Obzirom na kod u tablici 2.2 i zapise uz tablicu 2.8, već se pokazalo da jedna pogreška u *r* i dvostruka pogreška u *p* i *q* daje iste vrijednosti $y_1$, $y_2$, $y_3$. Postoje brojna druga ponašanja pogrešaka koja će dati iste $y_1$, $y_2$, $y_3$. Ispravno se zapitati: Kako prijemnik zna da se pojavila samo jedna pogreška, prije primjene postupka ispravka jedne pogreške? Odgovor je da *prijemnik doista ne zna prirodu pogrešaka.*

Odluka o tomu koju bi vrstu postupka ispravke pogrešaka trebalo primijeniti, temelji se na prijašnjim mjerenjima ponašanja pogrešaka u komunikacijskome kanalu.

[45] Ovu tablicu napraviti i sklopovski i C++!

Pretpostavka da se javljaju jednostruke pogreške opravdana je tek nakon što se opsežnim statističkim mjerenjima utvrdi ta činjenica. Ako se ta pretpostavka naruši, "ispravak" pogrešaka uvest će dodatnu pogrešku.

Može se pokazati da su kodne riječi u tablici 2.2 jedine kodne riječi koje zadovoljavaju jednadžbe 2.2 [(1), (2) i (3)].[46] Ova činjenica vrlo je važna. Na primjer, binarni vektor duljine 7 može se provjeriti pripada li kodnoj riječi ovoga koda jednostavnom provjerom zadovoljava li jednadžbe 2.2 [(1), (2) i (3)]. Drugim riječima, vektor **d** = [$p$, $q$, $r$, $s$, $t$, $u$, $v$], kodna je riječ koda iz tablice 2.2, onda i samo onda ako je $s \oplus t \oplus u \oplus v = 0$, $q \oplus r \oplus u \oplus v = 0$ i $p \oplus r \oplus t \oplus v = 0$. Slika 2.8 prikazuje krug za utvrđivanje je li vektor **d** kodna riječ našega koda.




| $p_0p_1$ | $a$ | $p_2$ | $bcd$ |
|---|---|---|---|
| 00 | 0 | 0 | 000 |
| 11 | 0 | 1 | 001 |
| 01 | 0 | 1 | 010 |
| 10 | 0 | 0 | 011 |
| 10 | 0 | 1 | 100 |
| 01 | 0 | 0 | 101 |
| 11 | 0 | 0 | 110 |
| 00 | 0 | 1 | 111 |

| $p_0p_1$ | $a$ | $p_2$ | $bcd$ |
|---|---|---|---|
| 11 | 1 | 0 | 000 |
| 00 | 1 | 1 | 001 |
| 10 | 1 | 1 | 010 |
| 01 | 1 | 0 | 011 |
| 01 | 1 | 1 | 100 |
| 10 | 1 | 0 | 101 |
| 00 | 1 | 0 | 110 |
| 11 | 1 | 1 | 111 |

*Tablica 2.1 (7, 4) kod*

| | | | |
|---|---|---|---|
| 0000 | 000 | 1000 | 110 |
| 0001 | 101 | 1001 | 011 |
| 0010 | 111 | 1010 | 001 |
| 0011 | 010 | 1011 | 100 |
| 0100 | 011 | 1100 | 101 |
| 0101 | 110 | 1101 | 000 |
| 0110 | 100 | 1110 | 010 |
| 0111 | 001 | 1111 | 111 |

*Slika 2.8: Provjera je li vektor iz tablice 2.2 kodna riječ koda*

Izlazi $y_1$, $y_2$, $y_3$ sklopa prikazanoga na slici 2.7 su "sve 0" onda i samo onda ako je vektor, kodna riječ koda iz tablice 2.2. Do sada još nismo pokazali da je minimalna Hammingova udaljenost koda iz tablice 2.2 jednaka 3. Ovo se naravno, može provjeriti mjerenjem Hammingove udaljenosti između svih mogući parova kodnih riječi u popisu kodnih riječi ovoga koda.[47] Međutim, za svrhe udžbenika naš cilj je dokazati ovo svojstvo obzirom na sposobnost otkrivanja/ispravke pogrešaka koda.

[46] Napraviti C++ program kojime se to dokazuje!
[47] Napraviti C++ program!

## 5. *Laboratorijska vježba 4: Metode optimalnoga i ravnomjernoga kodiranja (Shannon-Fano kod, Huffmanov kod, Hammingov kod)*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

Cjelovit primjer optimalnoga kodiranja u 18 slika nalazi se na MOODLE



**Definicije:**

***Optimalno kodiranje*** - postupak kodiranja koji ostvaruje minimalno vrijeme prijenosa poruka.

***Ravnomjerno kodiranje*** - postupak kodiranja kod kojega su sve kodne riječi jednake dužine.

Shannon Fano (sequence S)

### 8. *Shannon-Fano (pseudo kod)*

```
//#include <conio.h> // getch();
```

*Order the set of symbols according to the frequency of occurrence*;
Shannon Fano (sequence S)
if S *has two elements*
    *attach* 0 *to the codeword of one element and* 1 *to the codeword of another*;
else if S *has more than one element*
    *divide* S *into two subsequences*, $S_1$ *and* $S_2$,*with the minimal*
        *difference between probabilities of each subsequence*;
    *extend the code word for each symbol in* $S_1$ *by attaching* 0, *and attaching*
        *l to each code word for symbols in* $S_2$;
    ShannonFano ($S_1$);
    ShannonFano ($S_2$);

C:\windows\system32\cmd.exe

### 9. *Huffmanov algoritam (pseudo kod)*

```
//#include <conio.h> // getch();
```

Huffman()
    for(za svaki simbol napravite korijenski čvor iz čega slijedi čitavo stablo,)
        a prema vjerojatnosti pojave simbola
    while(sve dok ne preostane samo jedan čvor)
        Uzmite 2 stabla $t_1$, $t_2$ *s najmanjim vjerojatnostima* $p_1$, $p_2$ ($p_1 \leq p_2$)
        pa napravite stablo s $t_1$ i $t_2$ kao njihovim nasljednicima uz
        vjerojatnost u novome korijenu jednaku $p_1 + p_2$;
    pridružite 0 svakoj lijevoj grani, a 1 svakoj preostaloj desnoj grani;
    napravite jedinstvenu kodnu riječ za svaki simbol prolazeći kroz stablo počevši od korijena
        prema listu koji sadrži pripadnu vjerojatnost
        simbola združujući sve nule i jedinice u jednu cjelinu (riječ);

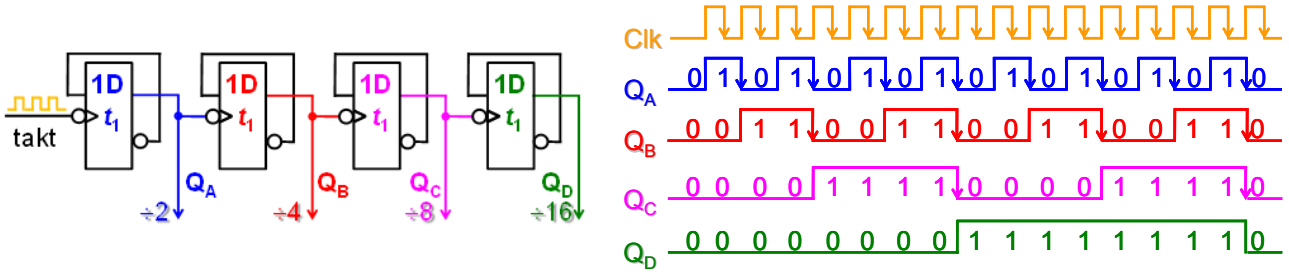
C:\windows\system32\cmd.exe

### 2.6.1. 2.6.1. OSNOVNI GENERATORI PARITETNE MATRICE

Jednostavan posmični registar SR (*shift register*) može se napraviti korištenjem samo flip-flop sklopa D vrste, po jedan flip-flop (*bistabil*) za svaki podatkovni bit. Izlaz iz svakoga sklopa bistabila, spaja se na D ulaz sklopa bistabila njemu s desne (ili lijeve) strane. Posmični registri čuvaju podatke u svojoj memoriji. Svaki impuls takta istovremeno pomiče ili "posmiče" (*shift*) sadržaje registara za jedan bit. Podatkovni bitovi mogu se unositi serijskim ulazom bit po bit ili paralelno (svi bitovi istovremeno). Podaci se mogu ukloniti iz registra bit po bit na serijskome izlazu ili istovremeno na paralelnim izlazima. Jedna od primjena posmičnih registara pretvorba je između serijskih i paralelnih podataka.

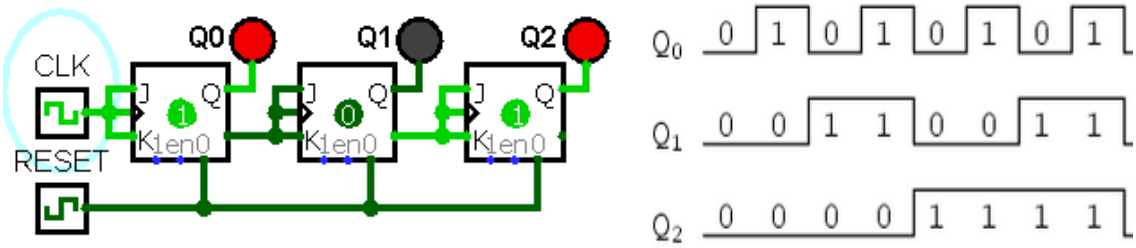
Posmični registri prepoznaju se prema načinu prihvata i isporuke binarnih podataka kao SIPO (*Serial Input, Parallel Output*), SISO (*Serial Input, Serial Output*), PISO (*Parallel Input, Serial Output*) i PIPO (*Parallel Input, Parallel Output*) vrste te kao opći posmični registri. Dodatkom povratne veze opći registri postaju *posmični registri s povratnom vezom* - LFSR (*Linear Feedback Shift Registers*).

Osnovna struktura LFSR sklopova sastoji se od D bistabila (slika 2.9).




*Slika 2.9: Serijski spojena 4 D-bistabila*

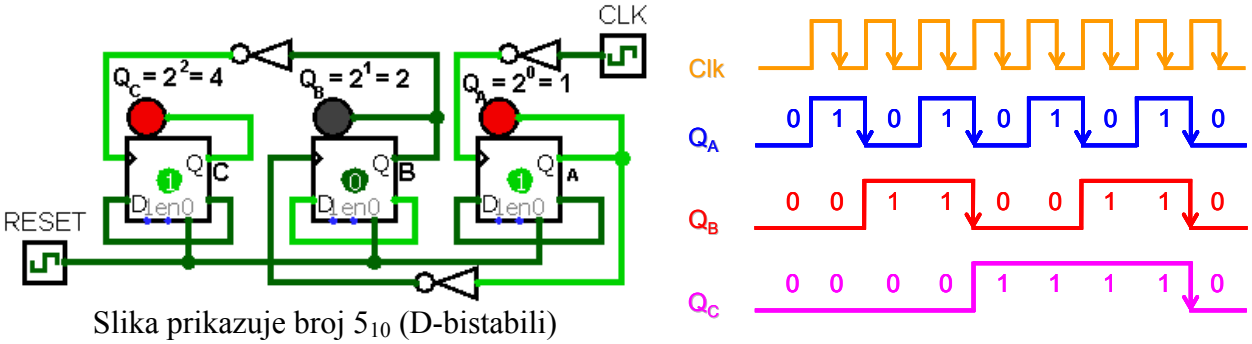
LFSR je skup serijski spojenih D bistabila s povratnim vezama. Sličan učinak stvaranja paritetne **H** matrice ostvaruje sklop LFSR napravljen JK bistabilima. On pokazuje kako sklopovski dobiti upravo traženu **H** matricu (slika 2.10).




*Slika 2.10: Generiranje **H** matrice JK-bistabilima*

I jedni i drugi sklopovi mogu se koristiti kao sastavni dijelovi posmičnoga registra i mogu generirati $\mathbf{H}^T$ matricu (slika 2.11).

Slika matrica H.circ
(posmični registar i D bistabili)




Slika prikazuje broj $5_{10}$ (D-bistabili)

*Slika 2.11: Prikaz broja $5_{10}$ D-bistabilima*

### 2.6.2. 2.6.2. PARITETNA MATRICA I PARITETNE JEDNADŽBE

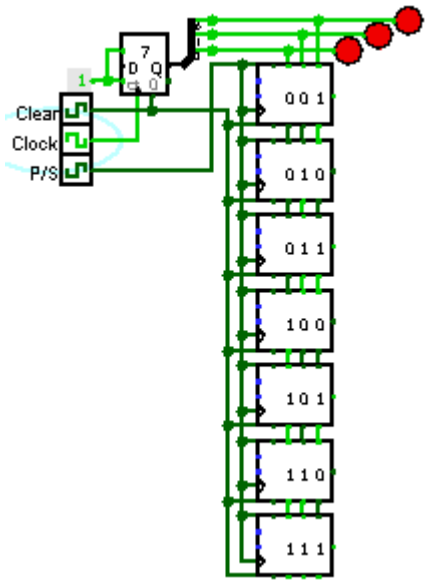
*Oznaka* Kroz ostatak ovoga poglavlja, matrice obilježene $\mathbf{H}^T$ jednake su ovoj (binarni brojevi, u rastućemu redoslijedu, poredani su odozgo prema dolje). Ovdje je također sadržano objašnjenje strukture paritetnih jednadžbi $p_0$, $p_1$, $a$, $p_2$, $b$, $c$, $d$ (stupci matrice i retci samo s jednom jedinicom):

$$\mathbf{H}^T = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccc} 2^2 & 2^1 & 2^0 \end{array} \\ \left[\begin{array}{ccc} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{l} \rightarrow 1 \rightarrow 2^0 = 1 \cong 1 \\ \rightarrow 1 \rightarrow 2^1 = 2 \cong 2 \\ 2^0 + 2^1 \quad \cong 3 \\ \rightarrow 1 \rightarrow 2^2 = 4 \cong 4 \\ 2^2 + 2^0 \quad \cong 5 \\ 2^2 + 2^1 \quad \cong 6 \\ 2^2 + 2^1 + 2^0 \cong 7 \end{array}$$

$$p_0 = a \oplus b \oplus d \quad p_0 = \sum(1 \oplus 3 \oplus 5 \oplus 7) = 0$$

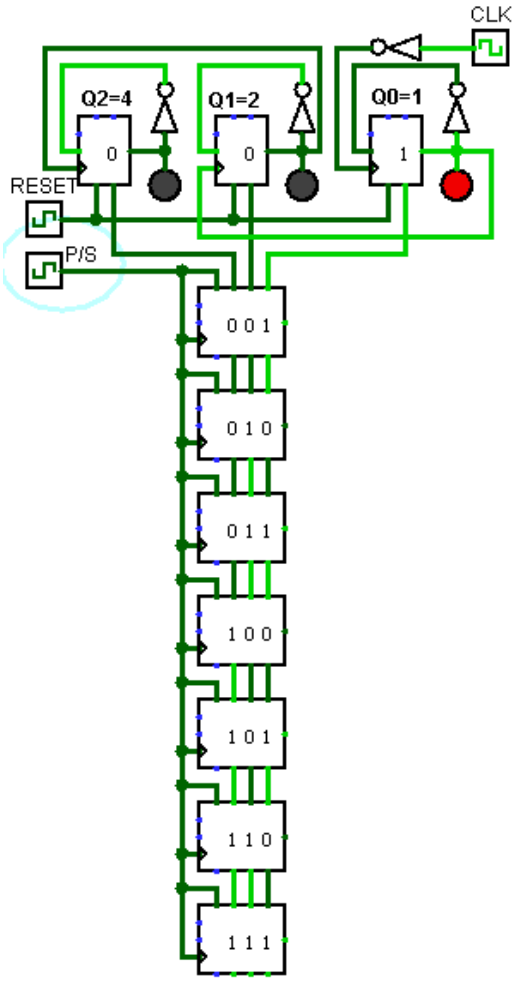
$$p_1 = a \oplus c \oplus d \quad p_1 = \sum(2 \oplus 3 \oplus 6 \oplus 7) = 0$$

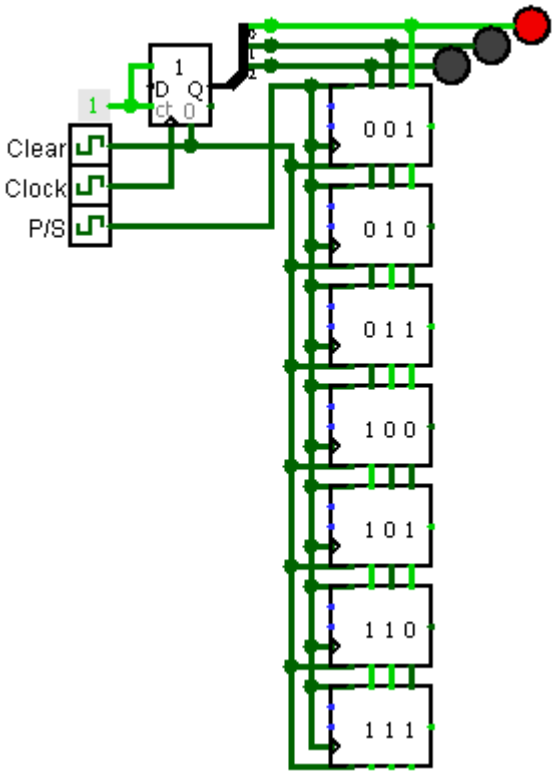
$$p_2 = b \oplus c \oplus d \quad p_2 = \sum(4 \oplus 5 \oplus 6 \oplus 7) = 0$$




LFSR?

*Slika 2.12: Simulacijski krug za stvaranje $\mathbf{H}^T$ matrice*




Cjelovit krug za punjenje paritetne matrice $\mathbf{H}^T$:

*Slika 2.13: Cjelovit krug za punjenje paritetne matrice $\boldsymbol{H}^T$*

Neka je **x** vektor od sedam bitova, **x** = [*a, b, c, d, e, f, g*]. Onda imamo:

$$\mathbf{x} \cdot \mathbf{H}^T = [d \oplus e \oplus f \oplus g, b \oplus c \oplus f \oplus g, a \oplus c \oplus e \oplus g]$$

Oznaka ⊕ označava XOR operaciju pa je stoga, $\mathbf{x} \cdot \mathbf{H}^T$ binarni vektor. Pogledajmo sada kod naveden u tablici 2.2. Kodna riječ $\mathbf{z} = [p, q, r, s, t, u, v]$ zadovoljava slijedeće jednadžbe:

| $p_0$ | $p_1$ | $a$ | $p_2$ | $b$ | $c$ | $d$ |
|---|---|---|---|---|---|---|
| $p$ | $q$ | $r$ | $s$ | $t$ | $u$ | $v$ |

(paritetne jednadžbe s izvornim oznakama i paralele prema novima)

(1) $s \oplus t \oplus u \oplus v = 0,$

(2) $q \oplus r \oplus u \oplus v = 0,$

(3) $p \oplus r \oplus t \oplus v = 0$

Onda možemo napisati da je $\mathbf{z} \cdot \mathbf{H}^T = [0, 0, 0]$. Isto tako bilo koji vektor $\mathbf{y}$ koji zadovoljava uvjet $\mathbf{y} \cdot \mathbf{H}^T = [0, 0, 0]$, kodna je riječ ovoga koda. Stupci matrice $\mathbf{H}^T$ tvore koeficijente jednadžbi, što ih kodne riječi ovoga Hammingova koda trebaju zadovoljiti. Prvi stupac znači da je zbroj posljednja četiri elementa kodne riječi jednak 0, itd. Zbog općenitosti ($n$, $k$) linearnoga koda, postoji $n–k$ nezavisnih jednadžbi, na temelju kojih se računa $n–k$ paritetnih bitova. Iz toga možemo oblikovati bilo koju matricu općega koda čiji stupci su koeficijenti tih nezavisnih jednadžbi.

*Definicija* **Matrica pariteta** linearnoga ($n$, $k$) koda ima $n$ redaka i $n–k$ stupaca. Svaki *stupac* sastoji se od koeficijenta jedne, od $n–k$ nezavisnih jednadžbi što ih kodne riječi trebaju zadovoljiti. Napomene: $n = 7$, ukupan je broj bitova kodne riječi, a $k = 4$, duljina je informacijskoga vektora koji se kodira.

### 2.6.3. 2.6.3. POLOŽAJ PARITETNIH BITOVA U KODNOJ RIJEČI OGLEDA SE U STRUKTURI PARITETNE MATRICE

Ako se svaki paritetan bit u kodnoj riječi koda računa zasebno, znači da svaka od jednadžbi iz kojih se dobivaju paritetni bitovi, sadrži jednu i samo jednu parnost. Budući da je broj tih jednadžbi ujedno i broj paritetnih bitova, posljednja tvrdnja također znači da se svaki paritetan bit pojavljuje jednom u potpunome skupu jednadžbi. Ovaj argument objašnjava se ponovnim razmatranjem primjerice, triju jednadžbi, čija su se 3 paritetna bita Hammingova (7, 4) koda, opisala u prethodnome poglavlju, a računaju se kao:

(1) $p_2 \oplus b \oplus c \oplus d = 0$

(2) $p_1 \oplus a \oplus c \oplus d = 0$

(3) $p_0 \oplus a \oplus b \oplus d = 0$

$\Big\} \sum \bmod 2$

Promatramo kako se u matrici pariteta odražava činjenica da se svaki paritetan bit pojavljuje samo jedanput. Kao što smo već najavili, svaki stupac matrice pariteta $\mathbf{H}^T$, predstavlja koeficijente svake od jednadžbi provjere pariteta. Ako se svaki paritetan bit pojavljuje samo u jednoj jednadžbi (a pripada jediničnoj matrici, koja se napiše tako da tvori sustavan oblik), onda znači da ima samo jedan stupac u matrici pariteta, koji ima 1 na mjestu podudarnome paritetnome bitu. Ovo se objašnjava promatranjem matrice pariteta $\mathbf{H}^T$ našega koda. Kodna riječ koja odgovara pripadnoj matrici, ima opću strukturu $p_0\ p_1\ a\ p_2\ b\ c\ d$. Stupci $\mathbf{H}^T$ matrice predstavljaju koeficijente prethodno napisanih (i ovdje ponovljenih) jednadžbi 2.2 [(1), (2) i (3)].

Položaji paritetnih bitova u kodnoj riječi su prvi, drugi i četvrti (računajući s lijeva). Obratite pozornost da *prvi*, *drugi* i *četvrti* redak matrice $\mathbf{H}^T$ sadrže samo jedan element vrijednosti 1. Da zaključimo, ako se svaki paritetan bit računa neovisno preko vrijednosti samo nekih informacijskih bitova, onda retci matrice pariteta, čiji je položaj jednak paritetnome bitu u kodnoj riječi, imaju samo jedan element vrijednosti 1.

Redovi takve matrice pariteta poredani su prema dekadskome prikazu niza uzastopnih binarnih brojeva u rastućemu redoslijedu (vidi sliku 2.11). Započnemo binarnim prikazom 1 (kao što je slučaj s našom matricom $\mathbf{H}^T$). Položaj redaka s jednom 1, ujedno je i položaj paritetnih bitova u kodnoj riječi, potencija je broja 2 (vidi paritetnu matricu $\mathbf{H}^T$ otprije). Binarni prikaz potencije broja 2 ima jednu 1. Ovo objašnjava zašto su paritetni bitovi u kodnoj riječi našega Hammingova koda posebno smješteni na mjestima #1, #2 i #4.

### 2.6.4. 2.6.4. ODRAZ ISPRAVKE POGREŠAKA HAMMINGOVA KODA NA PARITETNU MATRICU

Matrica $\mathbf{H}^T$ je matrica pariteta koda iz tablice 2.2. Ovaj kod je Hammingov kod koji omogućuje ispravak jednostruke i otkrivanje dvostrukih pogrešaka. To svojstvo slijedi iz razmatranja matrice pariteta. Neka je **x** kodna riječ iz tablice 2.2, gdje se šalje **x**, a prima se **y**. Onda, vektor **z** = **x** + **y** predstavlja *uzorak pogreške*.

Također možemo pisati **y** = **x** + **z**. S ciljem otkrivanja/ispravke pogrešaka, prijemnik koristi množenje vektora **y** i matrice $\mathbf{H}^T$. Imamo $\mathbf{y} \cdot \mathbf{H}^T = (\mathbf{x} + \mathbf{z}) \cdot \mathbf{H}^T = \mathbf{x} \cdot \mathbf{H}^T + \mathbf{z} \cdot \mathbf{H}^T$. Sve dok je **x**, kodna riječ, vektor $\mathbf{x} \cdot \mathbf{H}^T$ sadrži "sve 0" pa je stoga $\mathbf{y} \cdot \mathbf{H}^T = \mathbf{z} \cdot \mathbf{H}^T$.

*Definicija* Neka je **x** kodna riječ koje se prenosi i neka je **y** njena primljena inačica. Vektor dobiven množenjem vektora **y** matricom pariteta koda $\mathbf{H}^T$, zove se ***sindrom pogreške***.

Budući da je $\mathbf{y} \cdot \mathbf{H}^T = \mathbf{z} \cdot \mathbf{H}^T$, slijedi da je *sindrom pogreške jednak vektoru dobivenome množenjem vektora uzorka pogreške* **b** *matricom pariteta* $\mathbf{H}^T$. Za otkriti/ispraviti pogrešku, prvi korak je izračun sindroma pogreške, a izvodi se u prijemniku.

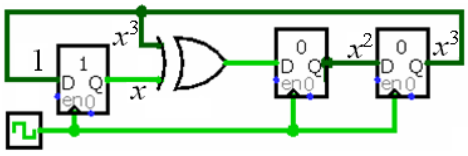
*Definicija* **Dekodiranje** je postupak koji se provodi u prijemniku. Iz primljene poruke najprije se računa *sindrom pogreške*, a na temelju njega, pogrešaka se otkriva/ispravlja. Krug za izvođenje ove operacije zove se *dekoder*.

Promatrajmo što će se sada dogoditi ako se u vektoru **x**, na njegovome putu prema prijemniku, pojavi samo jedna pogreška. Ovdje smatramo da **z** ima jedan element vrijednosti 1 čiji je položaj pogrešan bit, dok su svi ostali njegovi elementi jednaki 0. Sindrom pogreške $\mathbf{y} \cdot \mathbf{H}^T$, što ga računa prije prikazan prijemnik, jednak je $\mathbf{z} \cdot \mathbf{H}^T$, a to je jednako pojedinome retku u $\mathbf{H}^T$ čiji položaj (računajući od vrha) potječe od jednoga bita iz **z**.

Budući da je to položaj pogrešnoga bita, možemo zaključiti da je sindrom pogreške što ga računa prijemnik, $\mathbf{y} \cdot \mathbf{H}^T$, jednak tome retku u $\mathbf{H}^T$ čiji položaj predstavlja položaj pogrešnoga bita. Budući da su svi redovi od $\mathbf{H}^T$ različiti, dobiveni sindrom pogreške onda nam jedinstveno omogućuje prepoznati mjesto pojedine pogreške.

Imajte na umu da je u posebnome slučaju matrice $\mathbf{H}^T$, iz sindroma $\mathbf{y} \cdot \mathbf{H}^T$ veoma lako odrediti mjesto jedne pogreške u **y**, jer su redovi $\mathbf{H}^T$ prirodan binaran prikaz dekadskih brojeva 1, 2, 3, 4, 5, 6, 7, raspoređeni u rastućemu redoslijedu. Slijedi da je *za kod čija je paritetna matrica* $\mathbf{H}^T$, *sindrom jednak binarnome prikazu indeksa mjesta pogreške.*

*Primjer* Neka se prenosi kodna riječ **x** = [0111001]. Na prijenosnome putu riječ **x** doživi pogrešku na bitu #5 (s lijeve strane) i dobije se vektor **y** = [0111101]. Paritetna matrica $\mathbf{H}^T$ automatski se dobila sklopom kojega prikazuje slika 2.14




*Slika 2.14: Automatsko stvaranje redaka matrice* $\mathbf{H}^T$ *polinomom* $R(x) = 1 + x + x^3$

$$\mathbf{H}^T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{ccc} p_0 & p_1 & p_2 \end{array} \\ \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \\ a \\ b \\ c \\ d \end{array}$$

$\mathbf{H}^T$ matrica u sustavnome obliku daje paritetne jednadžbe:

$$p_0 = a \oplus c \oplus d, p_1 = a \oplus b \oplus c, p_2 = b \oplus c \oplus d$$

Iz matrice pariteta $\mathbf{H}^T$ dobije se tzv. **H** matrica, a iz nje generator matrica **G**. Prikazom matrice **G** u sustavnome obliku, prepoznaje se njezin sastav od dviju međusobno "zalijepljenih" pod-matrica, paritetne pod-matrice **P** i jedinične pod-matrice **I**.

$$\mathbf{H}^T = \begin{bmatrix} \mathbf{I}_{n-k} \\ \hline \mathbf{P} \end{bmatrix} \Rightarrow \mathbf{H} = [\mathbf{I}_{n-k} | \mathbf{P}^T] \Rightarrow \mathbf{G} = [\mathbf{P} \mid \mathbf{I}_k] \qquad (2.8)$$

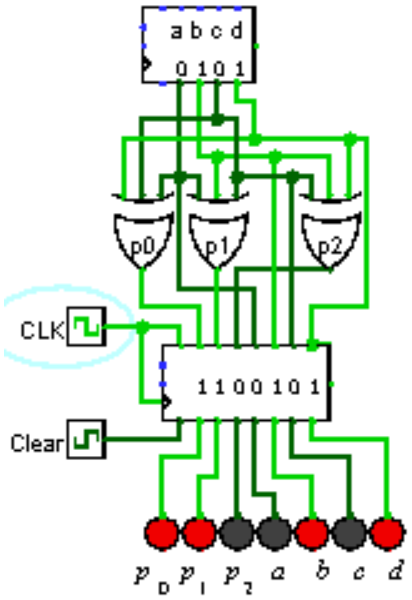
$$\mathbf{H}^T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{matrix} p_0 & p_1 & p_2 \end{matrix} \\ \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hline 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} p_0 \\ p_1 \\ p_2 \\ \hline a \\ b \\ c \\ d \end{array} \Rightarrow \quad \mathbf{H} = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \Rightarrow$$

LIJEVO je paritetna matrica **P**

$$\mathbf{G} = \left[\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

Jedinična matrica **I** je DESNO

Na temelju paritetnih jednadžbi, ustrojava se *koder*. On skup riječi od 4 bita (16 složaja), pretvara u kodne riječi kojima se automatski pridružuju paritetni bitovi na mjesta #1, #2 i #3, na početku informacijskih riječi od 4 bita, slika 2.15.




$$\mathbf{G} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{matrix} p_0 & p_1 & p_2 & a & b & c & d \end{matrix} \\ \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}$$

Pripadne paritetne jednadžbe sklopa su:

$$p_0 = a \oplus c \oplus d,$$

$$p_1 = a \oplus b \oplus c,$$

$$p_2 = b \oplus c \oplus d$$

a paritetni bitovi zauzimaju prva 3 mjesta s lijeve strane informacijskoga vektora:

$$p_0\, p_1\, p_2\, a\, b\, c\, d$$

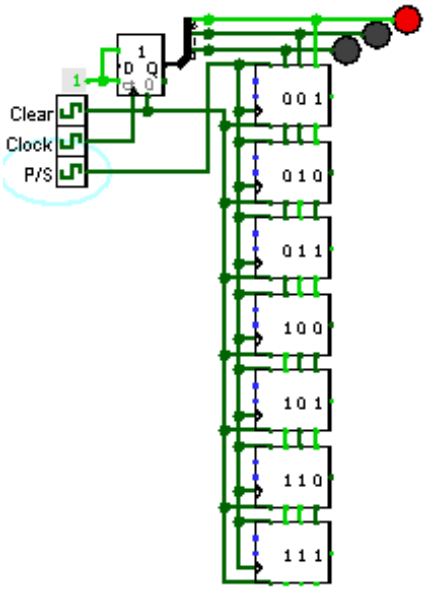
Kodne riječi Hammingova koda su:

| y | y |
|---|---|
| 0000000 | 1101000 |
| 1010001 | 0111001 |
| 1110010 | 0011010 |
| 0100011 | 1000011 |
| 0110100 | 1011100 |
| 1100101 | 0000101 |
| 1000110 | 0100110 |
| 0010111 | 1111111 |

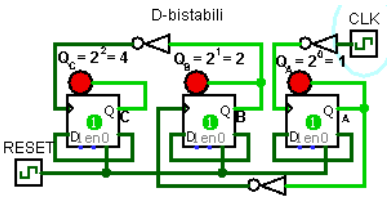
*Slika 2.15: Postupak kodiranja informacijskoga vektora od 4 bita*

$$\mathbf{H}^t = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{matrix} 2^2 & 2^1 & 2^0 \end{matrix} \\ \left[\begin{array}{ccc} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{array}\right] \end{array} \begin{array}{l} p_0 \to 1 \to 2^0 = 1 \cong 1 \\ p_1 \to 1 \to 2^1 = 2 \cong 2 \\ a \quad 2^0 \oplus 2^1 \cong 3 \\ p_2 \to 1 \to 2^2 = 4 \cong 4 \\ b \quad 2^2 \oplus 2^0 \cong 5 \\ c \quad 2^2 \oplus 2^1 \cong 6 \\ d \quad 2^2 \oplus 2^1 \oplus 2^0 \cong 7 \end{array}$$

Napomena: Ovakvo označavanje za $p_0$, $p_1$, $a$, $p_2$, $b$, $c$, $d$ izvorno je i logično, jer je povezano težinskim kodovima. Iz njega se trebalo izvesti prethodno označavanje za $\mathbf{H}^T$ ali nije.




**LFSR????**




Simulacijski krug (Pitanje: Kako izgleda LFSR za generiranje ovakve matrice?)

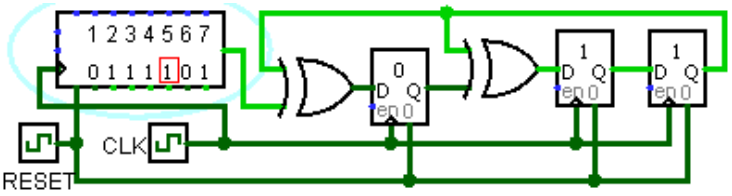
*Slika 2.16: Simulacijski krug*

Hammingov (7, 4) kod što ga na predajnoj strani generira sklop prikazan na slici 2.16 (matrica nije u sustavnome obliku)

| y | y |
|---|---|
| 0000000 | 1101000 |
| 1010001 | 0111001 |
| 1110010 | 0011010 |
| 0100011 | 1001011 |
| 0110100 | 1011100 |
| 1100101 | 0001101 |
| 1000110 | 0101110 |
| 0010111 | 1111111 |

Ako je **z** = [0000100], onda je primljeni vektor **y** jednak [0111101] pa se lako izračuna da je $\mathbf{y} \cdot \mathbf{H}^T = \mathbf{z} \cdot \mathbf{H}^T = [011]$.

$$\mathbf{z} \cdot \mathbf{H}^T = 0000100 \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = [011]$$




Nakon 7 posmika vektora **y** (ili **z**), sadržaji registara su [011], a to je sindrom pogreške. Sada se koristi tablica dekodiranja (standardno polje) u kojoj su združene ispravne kodne riječi, sindromi i riječi s jednostrukom pogreškom pa se iz te tablice očita ispravna kodna riječ.

*Slika 2.17: Automatsko generiranje redaka matrice* $\mathbf{H}^T$

Posmik kroz LFSR svih čelnika koskupova zasebno, daje sve sindrome. To je isto kao posmik kodiranoga informacijskoga vektora "sve 0" kroz LFSR, svaki puta s jednom pogreškom. Isti postupak (i rezultat) može se dobiti matematički, množenjem vektora **y** i matrice $\mathbf{H}^T$, odnosno $(\mathbf{y} + \mathbf{z}) \cdot \mathbf{H}^T$.

$$\mathbf{y} \cdot \mathbf{H}^T = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}^T \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}^T$$

Jednostruka pogreška

Množenje vektora **z** i matrice $\mathbf{H}^T$ (lijevo), daje rezultat [011], a to je sindrom pogreške.

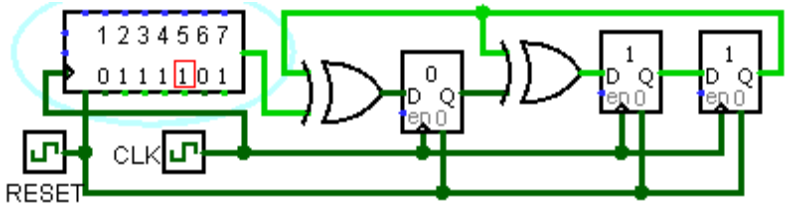
Množenje isprav(lje)noga vektora **y** + **z** = [0111001] matricom **H** (desno), potvrđuje ispravnost, riječi, jer je sada sindrom = [000].

$$[\mathbf{y}+\mathbf{z}] \cdot \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}^T \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}^T$$

Kod(ira)na riječ bez pogreške

### 2.6.4.1. 2.6.4.1. Standardno polje

Objašnjenje standardnoga polja dano je kao laboratorijska vježba 5: *Kodiranje, dekodiranje (otkrivanje i ispravak pogrešaka) za linearne blok-kodove* i/ili laboratorijska vježba 6: *Standardno polje za (8, 2) kôd*. Sljedećim sklopom (slika 2.18) provjerava se ispravnost rada sindroma.




*Slika 2.18: Provjerava ispravnosti sindroma*

| sindrom | čelnik | Kodne riječi | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 000 | 0000000 | 0000000 | 1010001 | 1110010 | 0100011 | 0110100 | 1100101 | 1000110 | 0010111 |
| 100 | 1000000 | 1000000 | 0010001 | 0110010 | 1100011 | 1110100 | 0100101 | 0000110 | 1010111 |
| 010 | 0100000 | 0100000 | 1110001 | 1010010 | 0000011 | 0010100 | 1000101 | 1100110 | 0110111 |
| 001 | 0010000 | 0010000 | 1000001 | 1100010 | 0110011 | 0100100 | 1110101 | 1010110 | 0000111 |
| 110 | 0001000 | 0001000 | 1011001 | 1111010 | 0101011 | 0111100 | 1101101 | 1001110 | 0011111 |
| 011 | 0000100 | 0000100 | 1010101 | 1110110 | 0100111 | 0110000 | 1100001 | 1000010 | 0010011 |
| 111 | 0000010 | 0000010 | 1010011 | 1110000 | 0100001 | 0110110 | 1100111 | 1000100 | 0010101 |
| 101 | 0000001 | 0000001 | 1010000 | 1110011 | 0100010 | 0110101 | 1100100 | 1000111 | 0010110 |

| sindrom | čelnik | Kodne riječi | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 000 | 0000000 | 1101000 | 0111001 | 0011010 | 1001011 | 1011100 | 0001101 | 0101110 | 1111111 |
| 100 | 1000000 | 0101000 | 1111001 | 1011010 | 0001011 | 0011100 | 1001101 | 1101110 | 0111111 |
| 010 | 0100000 | 1001000 | 0011001 | 0111010 | 1101011 | 1111100 | 0101101 | 0001110 | 1011111 |
| 001 | 0010000 | 1111000 | 0101001 | 0001010 | 1011011 | 1001100 | 0011101 | 0111110 | 1101111 |
| 110 | 0001000 | 1100000 | 0110001 | 0010010 | 1000011 | 1010100 | 0000101 | 0100110 | 1110111 |
| 011 | 0000100 | 1101100 | 0111101 | 0011110 | 1001111 | 1011000 | 0001001 | 0101010 | 1111011 |
| 111 | 0000010 | 1101010 | 0111011 | 0011000 | 1001001 | 1011110 | 0001111 | 0101100 | 1111101 |
| 101 | 0000001 | 1101001 | 0111000 | 0011011 | 1001010 | 1011101 | 0001100 | 0101111 | 1111110 |

### 2.6.5. 2.6.5. POSTUPAK OTKRIVANJA POGREŠKE HAMMINGOVA KODA ŠTO SE ODRAŽAVA NA NJEGOVU PARITETNU MATRICU

Pogledajmo sada kako iskoristiti svojstva $\mathbf{H}^T$ za pokazati da je nemoguće ispraviti dvije pogreške koje se dogode u prenesenoj kodnoj riječi **x**. U tomu slučaju, vektor uzorka pogreške **z** ima dva elementa vrijednosti 1, a ostatak su 0. Prijemnik računa sindrom $\mathbf{y} \cdot \mathbf{H}^T = \mathbf{z} \cdot \mathbf{H}^T$, a zatim izjednačuje zbroj dvaju redaka matrice $\mathbf{H}^T$ čija su mjesta ona koja sadrže pogreške. Da bi se dvije pogreške mogle ispraviti, moramo imati način za iskazati koja su to dva retka, čiji zbroj daje određen sindrom. Međutim, obzirom na bilo koji sindrom, postoji nekoliko parova redaka čiji zbroj daje ovaj sindrom.

To znači da je iz određenoga sindroma nemoguće odrediti, koja se dva specifična retka $\mathbf{H}^T$ zbrajaju operacijom $\mathbf{z} \cdot \mathbf{H}^T$, a kojom se dobije taj sindrom pa je ispravak pogrešaka nemoguć. Također, zbroj dvaju redaka **b** i **c** u $\mathbf{H}^T$ uvijek daje vektor **d**, a on je također redak u $\mathbf{H}^T$. Stoga je nemoguće iz određenoga sindroma zaključiti je li došlo do jednostruke (u kojemu slučaju sindrom pokazuje indeks mjesta pogreške) ili dvostruke pogrešku (u tome slučaju sindrom je jednak zbroju dvaju redaka u $\mathbf{H}^T$). Ovo opet prikazuje ono što smo naveli, naime, ispravak pogrešaka moguć je samo ako prijemnik radi uz pretpostavku da broj pogrešaka u primljenoj poruci ne prelazi sposobnost ispravke pogreške koda.

*Primjer* Neka je $\mathbf{y}\cdot\mathbf{H}^T = \mathbf{z}\cdot\mathbf{H}^T = [100]$. Sljedeći parovi redaka $\mathbf{H}^T$ imaju isti zbroj [100]: (prvi, peti), (drugi, šesti) i (treći, sedmi).

To znači da za sve sljedeće **z**, operacija $\mathbf{z} \cdot \mathbf{H}^T$ kao rezultat daje [100]: **z** = [1000100], **z** = [0100010] i **z** = [0010001]. Također, jedna pogreška na četvrtome mjestu u **y** daje isti sindrom.

Pokazali smo kako se sposobnost otkrivanja pogrešaka koda iz tablice 2.2, sada određuje analizom svojstava $\mathbf{H}^T$. Za primljenu poruku **y** utvrdilo se da ima pogreške onda i samo onda ako sindrom $\mathbf{y} \cdot \mathbf{H}^T$ nisu "sve 0" (što pokazuje da **y** nije kodna riječ). Za **y** koji sadrži jednu pogrešku, pokazali smo da je $\mathbf{y} \cdot \mathbf{H}^T$ jednako određenome retku u $\mathbf{H}^T$. Ako **y** sadrži dvije pogreške, pokazali smo da je $\mathbf{y} \cdot \mathbf{H}^T$ jednako zbroju dvaju redaka u $\mathbf{H}^T$. Tvrdimo da takav iznos nikada ne može dati vektor "sve 0", što osigurava otkrivanje. To proizlazi iz činjenice da su svi redovi $\mathbf{H}^T$ različiti pa je nemoguće da zbroj dvaju različitih redaka daje vektor "sve 0". Ako **y** sadrži tri pogreške onda je $\mathbf{y} \cdot \mathbf{H}^T$ jednako zbroj triju redaka u $\mathbf{H}^T$. Budući da je moguće odabrati tri retka iz $\mathbf{H}^T$ čiji je zbroj 0, ovdje je otkrivanje pogreške nemoguće. Možemo zaključiti da je sposobnost otkrivanja pogreške koda iz tablici 2.2 jednaka 2.

Sposobnost otkrivanja/ispravke pogrešaka koda može se odrediti iz paritetnoga bita matrice **P** kao što slijedi. Sposobnost ispravke pogrešaka koda je *t* ako svi različiti zbrojevi *t* redaka ili manje (ali najmanje 1 redak) iz pod-matrice **P** daju različite rezultate pa ima više od jednoga skupa *t*+1 redaka iz **P** čiji zbroj daje isti rezultat. Sposobnost otkrivanja pogreške koda je *r*, ako je nemoguće naći *r* ili manje redaka u **P** čiji je zbroj 0, gdje postoje neki *r*+1 retci u **P** čiji je zbroj 0.

Sada imamo dva različita objašnjenja o mogućnosti provedbe otkrivanja/ispravke pogrešaka kodova. Jedno od objašnjenja, dano prije, koristi neposredno stečenu spoznaju, razmatranje $d_{\min}$ (minimalna Hammingova udaljenost). S druge strane, kao što se upravo pokazalo, na ovu mogućnost gleda se kroz paritetnu pod-matricu **P**. Sada ćemo pokazati zašto su ta dva motrišta jednaka, dokazujući vrlo jednostavno sljedeće:

a) Minimalan broj redaka što se biraju iz **P** tako da njihov zbroj daje 0, je $d_{\min}$.

b) Svi mogući zbrojevi od $\lfloor (d_{\min}-1)/2 \rfloor$ ili manje redaka iz **P** vode različitim rezultatima, gdje je $\lfloor (d_{\min}-1)/2 \rfloor$ najveća vrijednost uz koju vrijedi ovo svojstvo.

*Dokaz* a)*:* Ako je $d_{\min}$ minimalna Hammingova udaljenost linearnoga koda, onda smo pokazali da je $d_{\min}$ minimalna Hammingova težina koda. To jest, svaka kodna riječ $\mathbf{v}_i$ koda osim riječi "sve 0", ima Hammingovu težinu $w_i$ od barem $d_{\min}$. Samo kodna riječ $\mathbf{v}_i$ ima svojstvo da je $\mathbf{v}_i \cdot \mathbf{P} = 0$. Imajte na umu da sada množenje $\mathbf{v}_i$ i **P** znači zbrajanje $w_i$ redaka od **P**. Da zaključimo, nemoguće je zbrojiti manje od $d_{\min}$ redaka **P** tako da njihov zbroj bude "sve 0". S druge strane, uzima se kodna riječ **v** čija je težina $d_{\min}$. Sve dok je $\mathbf{v} \cdot \mathbf{P} = \mathbf{0}$, u mogućnosti smo pronaći $d_{\min}$ redaka iz **P** čiji je zbroj 0.

*Dokaz* b): Pretpostavimo da smo u mogućnosti pronaći dva skupa redaka iz **P**, takvih da broj redaka u svakome skupu ne prelazi $\lfloor (d_{\min}-1)/2 \rfloor$ pa zbroj redaka svakoga skupa daje isti rezultat. Onda slijedi da zbrajanje redaka iz oba skupa zajedno, daje rezultat 0. Zatim možemo odabrati manje od $d_{\min}$ redaka **P** čiji je zbroj 0, što proturječi prethodnome dokazu. Slijedi da dva takva skupa ne postoje. Dokazivanje drugoga dijela b), sada je vidljivo.

## 2.7. 2.7. Izgradnja općega Hammingova koda

Do sada smo razmatrali samo Hammingov (7, 4) kod. Međutim, Hammingov kod općenito je ($2^n$–1, $2^n$–$n$–1) kod. Pogledajmo prvo kako produljiti kod čija je matrica pariteta $\mathbf{H}^T$ u (15, 11) kod (obrađeno u Laboratorijskoj vježbi 7). Poopćenje na više dimenzija bit će onda očito. Kao što smo pokazali, otkrivanje mjesta jednostruke pogreške pomoću matrice $\mathbf{H}^T$ vrlo je jednostavno, jer je sindrom pogreške binaran prikaz indeksa mjesta. To je zbog činjenice da je redak $i$ matrice $\mathbf{H}^T$ binaran prikaz $i$, za $i$ = 1, 2, 3, 4, 5, 6, 7. Ista ideja može se proširiti na višu dimenziju, gdje se redovi paritetne matrice $\mathbf{K}^T$ (15, 11) koda opet sastoje od binarnih prikaza brojeva raspoređenih u *rastućemu redoslijedu* (od 0001 do 1111):

$$\mathbf{K}^T = \begin{array}{c} \\ p_0 \\ p_1 \\ a \\ p_2 \\ b \\ c \\ d \\ p_3 \\ e \\ f \\ g \\ h \\ i \\ j \\ k \end{array} \begin{array}{c} \begin{array}{cccc} p_3 & p_2 & p_1 & p_0 \end{array} \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{array}\right. \end{array} \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \\ 13 \\ 14 \\ \left.15\right. \end{array}$$ [48]

[48] Važna napomena: Retci matrice pariteta $\mathbf{K}^T$ s jednom 1, označeni su na vrhu i sa strane kao paritetni bitovi ($p_0$, $p_1$, $p_2$, $p_3$). Ta 1 naglašava *mjesto* na koje se upisuje vrijednost paritetnoga bita ovisno o rezultatu XOR zbroja informacijskih bitova u

Kao i u bilo kojemu linearnomu kodu, kodnu riječ ovoga koda obilježava činjenica da se dobije **0** ako se pomnoži matricom pariteta $\mathbf{K}^T$, jer samo ona ima to svojstvo. Problem je u tome kako izgraditi kodnu riječ tj., kako izračunati četiri paritetna bita, na temelju jedanaest informacijskih bitova i gdje ih pronaći (smjestiti) unutar kodne riječi. U pogledu rasprave iz poglavlja 2.5.2, položaji paritetnih bitova odgovaraju položajima redaka u $\mathbf{K}^T$, gdje u retku ima *samo jedna* 1 pa su ova mjesta, potencije broja 2 tj. ($2^0$, $2^1$, $2^2$, $2^3$).

Dakle, paritetni bitovi su prvi, drugi, četvrti i osmi u kodnoj riječi. Struktura kodne riječi stoga je:

| | | 1. | | 2. | 3. | 4. | | 5. | 6. | 7. | 8. | 9. | 10. | 11. | Redni broj informacijskoga bita |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $p_0$ | $p_1$ | $a$ | $p_2$ | $b$ | $c$ | $d$ | $p_3$ | $e$ | $f$ | $g$ | $h$ | $i$ | $j$ | $k$ | Σ = 11 informacijskih bitova |
| 1. | 2. | 3. | 4. | 5. | 6. | 7. | 8. | 9. | 10. | 11. | 12. | 13. | 14. | 15. | pa se kod i označava (15, 11) |

gdje su $p_0$, $p_1$, $p_2$ i $p_3$ paritetni bitovi, a ostalo su izvorni informacijski bitovi. Paritetni bitovi računaju se iz jednadžbi čije koeficijente uvjetuju stupci od $\mathbf{K}^T$. Prvi stupac matrice $\mathbf{K}^T$ znači da je zbroj posljednjih osam bitova kodne riječi jednak 0, …. Sada su ove jednadžbe:

(1) $p_3 \oplus e \oplus f \oplus g \oplus h \oplus i \oplus j \oplus k = 0$ (počevši od $p_3$, svakih drugih 8 bitova) 1. stupac u $\mathbf{K}^T$

(2) $p_2 \oplus b \oplus c \oplus d \oplus h \oplus i \oplus j \oplus k = 0$ (počevši od $p_2$, svaka druga 4 bita) 2. stupac u $\mathbf{K}^T$

(3) $p_1 \oplus a \oplus c \oplus d \oplus f \oplus g \oplus j \oplus k = 0$ (počevši od $p_1$, svaka druga 2 bita) 3. stupac u $\mathbf{K}^T$

(4) $p_0 \oplus a \oplus b \oplus d \oplus e \oplus g \oplus i \oplus k = 0$ (počevši od $p_0$, svaki drugi bit) 4. stupac u $\mathbf{K}^T$

*Tablica 2.9: Provjera Hammingova pariteta*[49]

| položaj | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $2^0$ | $2^1$ | | $2^2$ | | | | $2^3$ | | | | | | | | $2^4$ |
| namjena | P1 | P2 | d1 | P4 | d2 | d3 | d4 | P8 | d5 | d6 | d7 | d8 | d9 | d10 | d11 | P16 |
| drugi način oznake | $p_0$ | $p_1$ | $a$ | $p_2$ | $b$ | $c$ | $d$ | $p_3$ | $e$ | $f$ | $g$ | $h$ | $i$ | $j$ | $k$ | $p_4$ |
| (1) pokriva P1 ($p_0$) | X | | X | | X | | X | | X | | X | | X | | X | |
| (2) pokriva P2 ($p_1$) | | X | X | | | X | X | | | X | X | | | X | X | |
| (3) pokriva P4 ($p_2$) | | | | X | X | X | X | | | | | X | X | X | X | |
| (4) pokriva P8 ($p_3$) | | | | | | | | X | X | X | X | X | X | X | X | |
| (5) pokriva P16 ($p_4$) | | | | | | | | | | | | | | | | X |

## 2.8. 2.8. Generator-matrica sustavnoga koda

U kontekstu ove skripte razmatrat će se oni slučajevi u kojima se svaki paritetan bit u kodnoj riječi računa zasebno samo na temelju vrijednosti pojedinih informacijskih bitova. Na temelju rasprave u poglavlju 2.6.3, retci matrice pariteta tih kodova, sadrže samo jedan bit vrijednosti 1 što se odražava i na položaj takvoga paritetnoga bita u kodnoj riječi.

### 2.8.1. 2.8.1. SUSTAVAN KOD

Sada razmotrimo slučaj *sustavnoga koda*, u kojemu se *paritetni bitovi* nalaze na *početku kodne riječi*. Za prikaz možemo razmatrati (9, 5) kod čije kodne riječi imaju strukturu $p_0\ p_1\ p_2\ p_3\ a\ b\ c\ d\ e$ ($a\ b\ c\ d\ e$ je informacijski vektor), gdje su paritetne jednadžbe:

(a) $p_0 = a \oplus b \oplus c$

(b) $p_1 = a \oplus d \oplus e$

(c) $p_2 = a \oplus b \oplus d \oplus e$

(d) $p_3 = a \oplus c \oplus d$

Matrica pariteta $\mathbf{H}^T$ ovoga koda onda je:

---

paritetnim jednadžbama za ($p_0$, $p_1$, $p_2$, $p_3$). Dakle, u matrici $\mathbf{K}^T$ (u navedenim retcima), NIJE upisana vrijednost samoga paritetnoga bita, jer se ona dinamički mijenja ovisno o vrijednostima informacijskih bitova.

[49] Za potrebe rada Hammingovim kodovima, bitovi se obično broje s lijeva u desno, počevši od 1, a ne od 0. To je različito od računanja bitova binarnih brojeva, što se broje s desna u lijevo, a počinju se brojati od 0. Ovakav način, Hammingove kodove čini razumljivijima.

$$\mathbf{L}^T = \begin{array}{c|cccc|l|l} 1. & 1 & 0 & 0 & 0 & p_0 & 1.\text{ paritetan bit} \rightarrow p_0 \\ 2. & 0 & 1 & 0 & 0 & p_1 & 2.\text{ paritetan bit} \rightarrow \rightarrow p_1 \\ 3. & 0 & 0 & 1 & 0 & p_2 & 3.\text{ paritetan bit} \rightarrow \rightarrow \rightarrow p_2 \\ 4. & 0 & 0 & 0 & 1 & p_3 & 4.\text{ paritetan bit} \rightarrow \rightarrow \rightarrow \rightarrow p_3 \\ 5. & 1 & 1 & 1 & 1 & a & a\ a\ a\ a \\ 6. & 1 & 0 & 1 & 0 & b & b\ \ \ b \\ 7. & 1 & 0 & 0 & 1 & c & c\ \ \ \ \ c \\ 8. & 0 & 1 & 1 & 1 & d & \ \ d\ d\ d \\ 9. & 0 & 1 & 1 & 0 & e & \ \ e\ e \end{array} \qquad (2.9)$$

Imajte na umu da je

$$[p_0\ p_1\ p_2\ p_3\ a\ b\ c\ d\ e] \cdot \mathbf{L}^T = \mathbf{0},$$

gdje se stupci $\mathbf{L}^T$ sastoje od koeficijenata paritetnih jednadžbi (a), (b), (c) i (d).

Može se primijetiti da retci matrice pariteta $\mathbf{L}^T$, prikladnim izborom položaja bitova u kodnoj riječi, imaju samo jedan element vrijednosti 1. Osim toga, prva četiri retka $\mathbf{L}^T$ imaju strukturu dijagonale od 1-elemenata, a ostali elementi su 0. Ovo je struktura **jedinične matrice** dimenzija $4 \times 4$.

Općenito, $(n, k)$ sustavan kod, u kojemu se $n–k$ paritetnih bitova pojavljuje na početku kodne riječi, ima matricu pariteta na čijemu je vrhu jedinična matrica dimenzija $(n–k) \times (n–k)$. Da budemo točniji, svaki od prvih $n–k$ redaka takvoga koda ima samo jednu 1. Linearnim kombinacijama redaka i/ili stupaca matrice, uvijek možemo preurediti matricu tako da se 1-ce pojave u dijagonalnoj strukturi.

### 2.8.2. 2.8.2. GENERATOR-MATRICA

Osim matrice pariteta, postoji još jedna matrica koja na jedinstven način obilježava određen linearan $(n, k)$ kod. Ova matrica, **G** (*generator-matrica koda*), posjeduje svojstvo automatskoga stvaranja kodnih riječi sukladnih informacijskome vektoru **u**, izvodeći operaciju **u** · **G** (tj., množeći informacijski vektor **u** i generator-matricu **G**).

Da bi vidjeli kako se izgrađuje matrica **G**, pogledajmo sljedeće *temeljno svojstvo* linearnoga koda. Neka $C(i)$, $C(j)$, $C(i, j)$ označavaju, redom, kodne riječi koje odgovaraju informacijskim vektorima $\mathbf{u}(i)$, $\mathbf{u}(j)$, $\mathbf{u}(i, j)$. Vektor $\mathbf{u}(i)$ ima samo jednu 1, na mjestu $i$, $\mathbf{u}(j)$ ima samo jednu 1 na mjestu $j$, a $\mathbf{u}(i, j)$ ima po jednu 1 na mjestima $i$ i $j$. Onda je:

$$C(i, j) = C(i) \oplus C(j).$$

Za prethodno opisan (9, 5) kod, imamo, npr. vektore (sastoje se od 4 znamenke), $\mathbf{u}(1) = [1000]$, $\mathbf{u}(2) = [0100]$ pa je $\mathbf{u}(1, 2) = [1100]$. Isto tako, ako je npr. (vidi **G** matricu u nastavku) $C(1) = [111110000]$, a $C(2) = [101001000]$ onda je $C(1, 2) = [010111000]$, što predstavlja zbroj $C(1) \oplus C(2)$ prema *temeljnome svojstvu* linearnoga koda.

Općenito, ako je $\{p, q, r, \ldots\}$ skup indeksa mjesta 1-elemenata u bilo kojemu *informacijskomu* vektoru **u**, onda je kodna riječ $C$ koja odgovara vektoru **u**, jednaka $C(p) + C(q) + C(r) + \ldots$ (zbroj redaka generator matrice **G**). Način kako će se izgraditi kodna riječ $C$, definira naša generator-matricu **G**. Za $(n, k)$ kod, **G** ima $k$ redaka duljine $n$, gdje je redak $i$ jednak $C(i)$, $i = 1, 2, \ldots, k$. Kodna riječ $C$ stvara se iz informacijskoga vektora **u**, operacijom

$$\mathbf{u} \cdot \mathbf{G}.$$

Operacija množenja binarnih brojeva, zapravo znači zbrajanje onih redaka (indeksa) matrice **G**, koji odgovaraju položajima 1-elemenata u **u**. To je upravo, prethodno opisana operacija $C(p) + C(q) + C(r) + \ldots$.

*Primjer* Za (9, 5) kod, koji se razmatra u ovome poglavlju, imamo:

$$\mathbf{G} = \begin{array}{c} \\ C(1) \\ C(2) \\ C(3) \\ C(4) \\ C(5) \end{array} \overbrace{\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}^{}$$

Retci matrice **G** počevši odozgo, su: $C(1)$, $C(2)$, …, $C(5)$. Odgovarajuća kodna riječ, npr., za *informacijski* vektor **u** = [11011] (u 3. stupcu **G** matrice), je **u** · **G** = [010011011]. Korištenjem prethodnih oznaka, **u** = **u**(1) + **u**(2) + **u**(4) + **u**(5) pa je umnožak **u** · **G** = $C(1)$ + $C(2)$ + $C(4)$ + $C(5)$ (uočite da nema ljubičasto obojenoga pribrojnika, jer se na 3. mjestu u *informacijskome* vektoru **u**, umjesto broja 1, nalazi broj 0). Prethodna generator-matrica može se podijeliti u dva dijela:

$$\mathbf{G} = \left[\begin{array}{cccc:ccccc} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] = [\mathbf{P} \mid \mathbf{I}_k]$$

Desni dio generator matrice **G**, jedinična je matrica **I**. Za *sustavan* kod, informacijski i paritetni bitovi razdvojeni su. Jedinična matrica **I** može se smjestiti *desno* u pripadnoj generator matrici **G** kao u ovome primjeru (ili *lijevo* - vidi *laboratorijsku vježbu* br. 3). Ova druga činjenica znači da kodna riječ **c** = **u** · **G**, koja se podudara informacijskim vektorom **u**, završava jediničnom matricom **I**, a to ponavlja prvu činjenicu drugačijim izričajem.

### 2.8.3. 2.8.3. VEZE IZMEĐU MATRICE PARITETA I GENERATOR-MATRICE

Sljedeći problem koji ćemo razmatrati, a on oblikuje temelje za neka glavna razmatranja u sljedećemu poglavlju, jest veza između *matrice pariteta* i *generator-matrice* bilo kojega sustavnoga koda. Takvi kodovi imaju paritetne bitove na *početku* (ili na *kraju*) svake kodne riječi. Dakle:

- Ako je jedinična matrica $\mathbf{I}_k$ smještena *lijevo* u generator-matrici **G**, onda je ***gornji*** dio u transponiranoj matrici pariteta $\mathbf{H}^T$, označen **P**, a ***iznad*** *jedinične matrice* $\mathbf{I}_{n-k}$, jednak dijelu generator-matrice **G**, smješten ***desno*** od jedinične matrice $\mathbf{I}_k$ u **G**. Takvi kodovi imaju paritetne bitove izdvojene na ***kraju*** (desno) u svakoj kodnoj riječi.

$$\mathbf{H} = [\mathbf{P}^T \mid \mathbf{I}_{n-k}] \Rightarrow \mathbf{H}^T = \left[\begin{array}{c} \mathbf{P} \\ \hdashline \mathbf{I}_{n-k} \end{array}\right] \Rightarrow \mathbf{G} = [\mathbf{I}_k \mid \mathbf{P}] \qquad (2.10)$$

- ***Donji*** dio u transponiranoj matrici pariteta $\mathbf{H}^T$, označen **P**, a ***ispod*** *jedinične matrice* $\mathbf{I}_{n-k}$, jednak je dijelu generator-matrice **G**, ***lijevo*** od jedinične matrice $\mathbf{I}_k$. Takvi kodovi imaju paritetne bitove izdvojene na ***početku*** (lijevo) u svakoj kodnoj riječi.

$$\mathbf{H}^T = \left[\begin{array}{c} \mathbf{I}_{n-k} \\ \hdashline \mathbf{P} \end{array}\right] \Rightarrow \mathbf{H} = [\mathbf{I}_{n-k} \mid \mathbf{P}^T] \Rightarrow \mathbf{G} = [\mathbf{P} \mid \mathbf{I}_k] \qquad (2.11)$$

Napomena: Ovisno o položaju promatrane pod-matrice **P**, dobiju se potpuno različite kodne riječi.

U slučaju našega (9, 5) koda, taj dio ima oblik:

$$\mathbf{P} = \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \begin{matrix} 5. = \mathbf{t} \\ 6. \\ 7. \\ 8. \\ 9. \end{matrix}$$

Priroda ove veze najbolje se objašnjava ponovnim razmatranjem našega (9, 5) koda. Ne zaboravite da *struktura* kodne riječi [$p_0$ $p_1$ $p_2$ $p_3$ $a$ $b$ $c$ $d$ $e$], zadovoljava paritetne jednadžbe unutar *svake* kodne riječi[50]:

*Primjer*:

a) $p_0 = a \oplus b \oplus c = 1 \oplus 0 \oplus 1 = 0$

b) $p_1 = a \oplus d \oplus e = 1 \oplus 1 \oplus 0 = 0$

c) $p_2 = a \oplus b \oplus d \oplus e = 1 \oplus 0 \oplus 1 \oplus 0 = 0$

d) $p_3 = a \oplus c \oplus d = 1 \oplus 1 \oplus 1 = 1$

informacijski vektor $\underset{a\ b\ c\ d\ e}{1\ 0\ 1\ 1\ 0}$

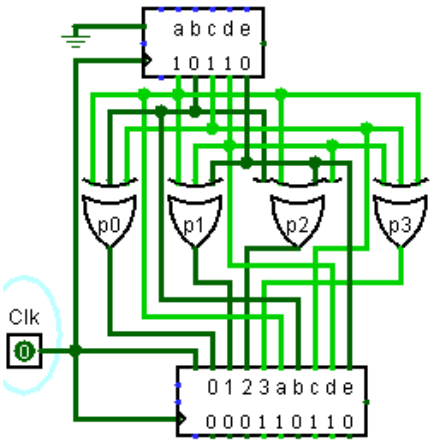
kodna riječ: $\underset{p_0\ p_1\ p_2\ p_3\ a\ b\ c\ d\ e}{0\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0}$

Neka je **t** peti redak paritetne matrice $\mathbf{L}^T$ (jednadžba 2.9), a prva četiri bita u prvoj crti generator-matrice **G** označimo s **u**. Sada pokazujemo razloge za činjenicu da je **t** = **u**. Budući da stupci $\mathbf{L}^T$ predstavljaju koeficijente paritetnih jednadžbi, bitovi vektora **t** predstavljaju one paritetne jednadžbe u kojima se pojavljuje informacijski bit "*a*" ("*a*"je peti bit u kodnoj riječi). Četiri jedinice u ovomu retku predstavlja činjenicu da se "*a*" pojavljuje u sve četiri jednadžbe. Šesti redak u $\mathbf{L}^T$ je [1010] i predstavlja činjenicu da se sljedeći informacijski bit "*b*" pojavljuje u prvoj i trećoj jednadžbi. Sada ćemo pogledati generator-matricu **G**. Njezin prvi redak je kodna riječ koja odgovara informacijskomu vektoru što ima sve "*a*" bitove jednake 1, a ostali informacijski bitovi su "sve 0".

**Primjer**:

| Informacijski vektor: | | | | | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | **1** | **0** | **1** | **1** | **0** |
| Kodna riječ: | **0** | **0** | **0** | **1** | **1** | **0** | **1** | **1** | **0** |
| | $p_0$ | $p_1$ | $p_2$ | $p_3$ | $a$ | $b$ | $c$ | $d$ | $e$ |

| |
|---|
| $p_0 = a \oplus b \oplus c$ |
| $p_1 = a \oplus d \oplus e$ |
| $p_2 = a \oplus b \oplus d \oplus e$ |
| $p_3 = a \oplus c \oplus d$ |




*Slika 2.19: Generator* (9, 5) *koda*

Kada se računaju *paritetni bitovi* što će se pridružiti *informacijskome vektoru* (da bi se oblikovala kod*na riječ*), ako se tamo pojavljuje "*a*", desna strana bilo koje od navedenih jednadžbi pariteta, jednaka je 1, u protivnome jednaka je 0. Budući da se u našemu slučaju "*a*" pojavljuje u sve četiri jednadžbe, sva četiri paritetna bita su 1. Prva četiri bita u prvomu retku paritetnoga dijela generator-matrice **G**, oblikuju vektor **u** = [1111], a to su upravo ovi paritetni bitovi, što objašnjava zašto je **t** = **u**. Da zaključimo, **t** = **u**, jer oba vektori imaju 1 na mjestu *i* onda i samo onda ako se informacijski bit "*a*" pojavljuje u *i*-toj jednadžbi pariteta.

Jednakost između šestoga retka u $\mathbf{L}^T$ i prva četiri bita u drugome retku matrice **G** može se prikazati na isti način, pokazujući zašto obje matrice imaju zajedničku pod-matricu **P**.

[50] Važna napomena: Vrijednosti paritetnih bitova u kodnoj riječi nisu zadani kao 1, već se dinamički određuju za svaku kod(ira)nu riječ prema ovim jednadžbama, ovisno o informacijskome vektoru što ga se kodira.

## 2.9. 2.9. Pregled uvedenih pojmova

- *Dekodiranje*: Postupak koji se izvodi u prijemniku, gdje se iz primljene poruke najprije računa sindrom pogreške, a na temelju njega otkriva se i/ili se ispravlja pogreška.
- *Generator-matrica linearnoga koda*: Matrica koja stvara kodnu riječ iz informacijskoga vektora, množenjem informacijskoga vektora i te matrice.
- *Hammingov kod*: To je $(2^n, 2^n–n–1)$ kod s minimalnom Hammingovom udaljenosti od 3.
- *Hammingova težina binarnoga vektora*: Broj elemenata vrijednosti 1 u vektoru.
- *Hammingova udaljenost između dvaju binarnih vektora*: Broj mjesta u kojima se vektori razlikuju.
- *Kodiranje*: Postupak što se izvodi u odašiljaču, gdje se računaju paritetni bitovi i pridružuju se informacijskome vektoru, tako da se oblikuje kodna riječ za prijenos kanalom.
- *Linearan kod*: Kod u kojemu je zbroj dviju kodnih riječi, također kodna riječ.
- *Matrica pariteta linearnoga koda*: Matrica čiji stupci predstavljaju koeficijente paritetnih jednadžbi što ih kodne riječi nekoga linearnoga koda trebaju zadovoljiti.
- *Minimalna Hammingova udaljenost koda*: Najmanja udaljenosti između svih mogućih parova kodnih riječi u kodu.
- *Minimalna Hammingova težina koda*: Najmanja težinama svih kodnih riječi u kodu, osim vektora "sve 0".
- *Mogućnost otkrivanja pogreške koda*: Najveći broj pogrešaka što ih se može pojaviti u kodnoj riječi i koje će uvijek rezultirati vektorom koji nije kodna riječ.
- *Osnovne formule*: Izraz $d_{\min}$ označava minimalnu Hammingovu udaljenost koda. Mogućnost otkrivanja pogreške koda je $d_{\min}–1$.
- *Sindrom pogreške*: Rezultat dobiven množenjem primljene poruke (čija je prenesena inačica kodna riječ) i paritetne matrice koda.
- *Sposobnost ispravke pogrešaka koda*: Najveći broj pogrešaka što se mogu pojaviti u kodnoj riječi i koje uvijek omogućuju određivanje izvorne kodne riječi iz vektora s pogreškama.
- *Sposobnost ispravke pogreške koda je*:

  $\lfloor (d_{\min}–1)/2 \rfloor$, gdje zagrade $\lfloor x \rfloor$ označavaju *cjelobrojnu* vrijednost od $x$.

# 6. *Laboratorijska vježba 5: Kodiranje, dekodiranje (otkrivanje i ispravak pogrešaka) za linearne blok-kodove*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

5. Kodiranje, dekodiranje (otkrivanje i ispravak pogrešaka) za linearne blok-kodove
5.1. Vježba: LFSR i (7, 4) kod
5.1.1. Zadatak za (7, 4) kod
5.1.1.1. Popunite priložene tablice
5.1.1.2. Pitanje:
5.1.1.3. Napišite tu matricu!
5.1.1.4. Zadatak:
5.2. Vježba: LFSR i (6, 3) kod
5.2.1. Zadatak za (6, 3) KOD
5.2.2. Zadatak
5.2.3. Primjer: Dekodiranja oštećenoga vektora za (6, 3) KOD.
5.2.4. Primjer: Linearnoga (6, 3) blok-koda
5.3. Položaji jedinične matrice
5.4. Sustavni linearni blok-kodovi
5.4.1. Matrica provjere pariteta
5.4.2. Provjera sindroma
5.4.3. Primjer: Provjera sindroma
5.4.4. Ispravak pogrešaka
5.4.5. Sindrom koskupa
5.5. Dekodiranje i ispravak pogrešaka
5.5.1.1. Pronalazak uzorka pogreške
5.5.1.2. Primjer: Ispravaka pogrešaka
5.6. Primjer: Ispravak pogrešaka za $I_k$ u **G** (desno)
5.7. Primjena dekodera

# 3. POGLAVLJE 3 - OSNOVNI STRUJNI KRUGOVI

## 3.1. 3.1 Automatsko stvaranje redaka matrice pariteta

Znamo da matrica pariteta obilježava kod u smislu da sve kodne riječi pomnožene ovom matricom dovode do sindroma **0** i nijedan drugi vektor osim kodne riječi nema ovo svojstvo. Također se pokazalo da *umnožak vektora i matrice* znači *zbrajanje redaka matrice* čiji položaji odgovaraju položajima 1-elemenata u vektoru.

Ako zamijenimo, na primjer, retke 2 i 4 u matrici pariteta, također treba zamijeniti elemente 2 i 4 u svakoj kodnoj riječi koja odgovara istome kodu. Množenje izmijenjene kodne riječi izmijenjenom matricom također će dati **0**, jer još uvijek zbrajamo iste retke kao i prije. Općenito, ako su npr. retci matrice pariteta slučajno (*randomly*) zaštitno kodirani (*scrambled*), isto zaštitno kodiranje primjenjuje se na elemente svake kodne riječi pa nov kod (čije su kodne riječi također zaštitno kodirane) ima zaštitno kodiranu (kriptiranu) matricu kao svoju matricu pariteta.

Nadalje razmatramo način dobivanja određenoga Hammingova koda primjenom postupka zaštitnoga kodiranja opisanoga za Hammingov kod iz tablice 2.2. Ako su *abcd* izvorni informacijski bitovi, struktura kodne riječi je $p_0\ p_1\ a\ p_2\ b\ c\ d$. Sada zaštitno kodirajmo svaku kodnu riječ. Nov redoslijed bio bi $p_2\ p_1\ p_0\ c\ a\ d\ b$. Paritetna matrica $\mathbf{H}^T$ našega novoga koda, prikazana dolje, dobila se primjenom istoga zaštitnoga kodiranja na retke matrice $\mathbf{H}^t$.

$$\mathbf{H}^t = \begin{array}{c|ccc|c} 1 & 0 & 0 & 1 & p_0 \\ 2 & 0 & 1 & 0 & p_1 \\ 3 & 0 & 1 & 1 & a \\ 4 & 1 & 0 & 0 & p_2 \\ 5 & 1 & 0 & 1 & b \\ 6 & 1 & 1 & 0 & c \\ 7 & 1 & 1 & 1 & d \end{array} \quad p_0\ p_1\ a\ p_2\ b\ c\ d \qquad \Rightarrow \qquad \mathbf{H}^T = \begin{array}{c|ccc|c} 4 & 1 & 0 & 0 & p_2 \\ 2 & 0 & 1 & 0 & p_1 \\ 1 & 0 & 0 & 1 & p_0 \\ 6 & 1 & 1 & 0 & c \\ 3 & 0 & 1 & 1 & a \\ 7 & 1 & 1 & 1 & d \\ 5 & 1 & 0 & 1 & b \end{array} \quad p_2\ p_1\ p_0\ c\ a\ d\ b$$

Naš nov Hammingov kod, iskazan matricom $\mathbf{H}^T$, *sustavan* je, jer su paritetni bitovi okupljeni na početku kodne riječi. To se uočava u strukturi $\mathbf{H}^T$, koja ima jediničnu matricu na svome vrhu. Temeljem rasprave na kraju prethodnoga poglavlja, naš kod ima generator-matricu i ovdje obilježenu kao **G**. Nju smo ustrojili tako, da smo "uzeli" donji dio $\mathbf{H}^T$ (dio ispod jedinične matrice) tzv. pod-matricu **P** te njoj s desna "prilijepili" odgovarajuću jediničnu matricu. Onda imamo:

$$\mathbf{G} = \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

Imajmo na umu da među osnovnim matricama **G**, **H** i $\mathbf{H}^T$ vrijede sljedeće relacije:

Jedinična matrica smještena DESNO u **G**:

$$\mathbf{G} = [\mathbf{P}_{n-k} \mid \mathbf{I}_k] \Rightarrow \mathbf{H} = [\mathbf{I}_{n-k} \mid \mathbf{P}_{n-k}^T] \Rightarrow \qquad \mathbf{H}^T = \begin{bmatrix} \mathbf{I}_{n-k} \\ \mathbf{P}_{n-k} \end{bmatrix} \tag{3.1}$$

i jedinična matrica smještena LIJEVO u **G**:

$$\mathbf{G} = [\mathbf{I}_k \mid \mathbf{P}_{n-k}] \Rightarrow \mathbf{H} = [\mathbf{P}_{n-k}^T \mid \mathbf{I}_{n-k}] \Rightarrow \qquad \mathbf{H}^T = \begin{bmatrix} \mathbf{P}_{n-k} \\ \mathbf{I}_{n-k} \end{bmatrix} \tag{3.2}$$

Pozornost treba obratiti na jediničnu pod-matricu $\mathbf{I}_k$, jer je njezin položaj (lijevo/desno) usko povezan s ostalim značajnim matricama **G**, **H**, $\mathbf{H}^T$, **P** i $\mathbf{P}^T$ (dobiju se različiti skupovi kodnih riječi).

Pri razmatranju poteškoća u pridruživanju paritetnih bitova informacijskim bitovima, sustavan kod ima veliku praktičnu prednost nad nesustavnim kodom, jer stvara kodnu riječ iz informacijskoga vektora.

Ovaj argument postaje jasniji ako se uspoređuje *struktura kodne riječi* u kodu čija je matrica pariteta $\mathbf{H}^t$ i u kodu čija je matrica pariteta $\mathbf{H}^T$. Ove strukture su:

- $p_0\ p_1\ a\ p_2\ b\ c\ d$ (za matricu $\mathbf{H}^t$), odnosno
- $p_2\ p_1\ p_0\ c\ a\ d\ b$ (za matricu $\mathbf{H}^T$).

Imajte na umu, da bi stvorili prvu kodnu riječ $p_0\ p_1\ a\ p_2\ b\ c\ d$ (za matricu $\mathbf{H}^t$), potrebno je umetnuli paritetan bit $p_2$ između dva informacijska bita. *Sklopovi potrebni* za takav zadatak *nezgodni* su i *skupi* obzirom na činjenicu da se zbog razloga praktičnosti, informacijski bitovi *kontinuirano pohranjuju u registar*. Međutim, u drugome slučaju, $p_2\ p_1\ p_0\ c\ a\ d\ b$ (za matricu $\mathbf{H}^T$), paritetni bitovi pridružuju se informacijskim bitovima pa nema potrebe razdvajati ih.

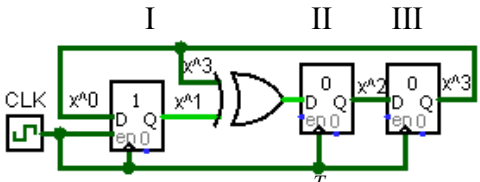
- *Kodiranje* (tj., stvaranje kodnih riječi iz informacijskih vektora) i
- *dekodiranje* (tj., stvaranje:
  - *sindroma* pogreške izvan primljene poruke te,
  - *ispravak* pogreške na temelju sindroma)

vrlo jednostavno, brzo i otmjeno izvode se korištenjem posebnoga sklopovlja.

- U linearnome kodu, *stvaranje kodne riječi* iz *informacijskoga vektora* **u**, uvijek je matematički jednako množenju vektora **u**, nekom generator-matricom **G**, bez obzira stvara li se praktički kodna riječ vještim sklopom pa se **G** izričito ne opaža (*not observed explicitly*).
- Isto se odnosi i na *postupak dekodiranja* u kojemu je stvaranje *sindroma* izvan primljene poruke jednako množenju primljenoga vektora matricom pariteta, čak i ako se ova matrica izričito ne vidi, jer je skrivena u *strukturi dekodera* (= *strukturi kodera*), a sastoji se od linearnoga posmičnoga registra s povratnom vezom LFSR[51] (*Linear Feedback Shift Register*) i vrata *izričito ILI*[52] EXOR (*EXclusive OR*). Često se ova kratica EXOR piše još kraće kao XOR.

Isto tako, ako postoji bilo kakav postupak ispravke pogrešaka kojime je moguće ispraviti do $t$ pogrešaka unutar bloka, onda minimalna Hammingova udaljenost koda nije manja od $2t+1$, čak iako se može učiniti da se pogreške ispravljaju nekim tehničkim postupkom koji u potpunosti ne uključuje razmatranje Hammingove udaljenosti. Ideja predstavljenoga sklopa, jest olakšati odašiljaču stvaranje kodnih riječi i u prijemniku brzo prepoznati pripada li primljena poruka skupu kodnih riječi.

Sada pokazujemo kako automatski generirati retke matrice $\mathbf{H}^T$. Ovo znači da nije potrebno pohraniti ovu matricu kao dio dekodera (za pomnožiti primljenu poruku ovom matricom, a time i stvoriti sindrom pogreške). Također, obzirom na izravnu povezanost između strukture $\mathbf{H}^T$ i generator-matrice **G'**, automatsko stvaranje redaka $\mathbf{H}^T$ također pomaže u postupku kodiranja (gdje se množe informacijski vektor i matrica **G'**). Krug koji automatski stvara retke matrice $\mathbf{H}^T$ prikazuje slika 3.1.




*Slika 3.1: Automatsko stvaranje redaka matrice* $\mathbf{H}^T$ *polinomom* $R(x) = 1 + x + x^3 = [1101]$

Slika 3.1 prikazuje posmični registar s tri stanja (desni posmik) s povratnom vezom. Za objasniti njegov rad, pogledajmo što će se dogoditi ako napravimo posmik, počevši početnim stanjem 100 (jer stanje "sve 0", ne daje nikakvu promjenu).

Tijekom prvoga posmika, 1 se pomiče u drugi registar (računajući s lijeva) i zatim se XOR zbraja nulom koja se vraća iz desnoga (zadnjega) registra. Ova 0 također se dovodi do prvoga stupnja (vidi Laboratorijsku vježbu 3). Sadržaj posljednjega stanja je 0, a to je prethodno bio sadržaj drugoga registra (stanja). Nov sadržaj je onda 010. Daljnji posmici vode sadržajima 001, a zatim 110, 011, 111, 101. Sljedeći posmik dovest će do sadržaja 100, što je početni sadržaj. Student treba razumjeti kako su dobiveni navedeni sadržaji.

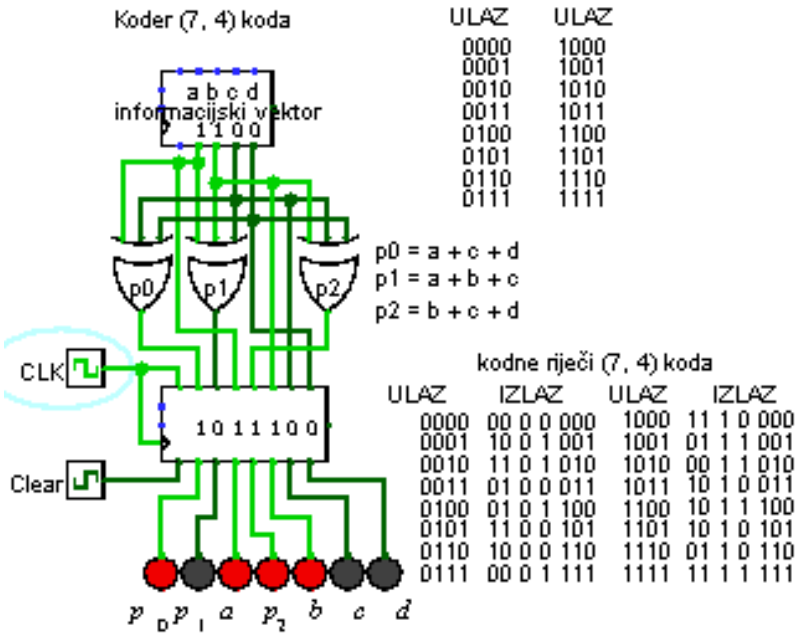
[51] LFSR je posmični registar čiji je ulazni bit, linearna funkcija njegovih prethodnih stanja.
[52] ILI jedan ILI drugi, ali NE oba!

Važna napomena: Redoslijed sadržaja matrice $\mathbf{H}^T$ određuje struktura LFSR registra [$R(x) = 1 + x + x^3$]. Iz matrice pariteta $\mathbf{H}^T$, unatrag se izvodi matrica $\mathbf{H}$, a iz nje generator-matrica $\mathbf{G}$ (ovdje je jedinična matrica $\mathbf{I}_k$ smještena desno u $\mathbf{G}$)

$$\mathbf{H}^T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{ccc} p_0 & p_1 & p_2 \end{array} \\ \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ p_0 \\ p_1 \\ p_2 \\ a \\ b \\ c \\ d \end{array} \Rightarrow \mathbf{H} = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \Rightarrow \mathbf{G} = \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

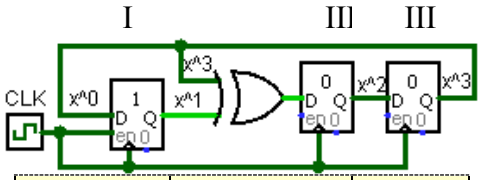
$$p_0 = a \oplus 0 \oplus c \oplus d,\ p_1 = a \oplus b \oplus c \oplus 0,\ p_2 = 0 \oplus b \oplus c \oplus d$$

Iz generator-matrice $\mathbf{G}$ ustrojava se koder za tvorbu kodnih riječi Hammingova (7, 4) koda (slika 3.2).




*Slika 3.2: Hammingov (7, 4) nesustavan koder*

Slika 3.3 prikazuje sadržaje što ih generira sklop na slici 3.1, a opisuje ga postupak ispod te slike:

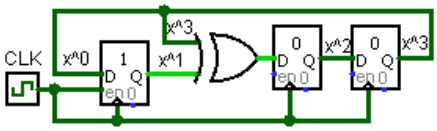



| | | |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 1 |

| | stanja registra R1 | I | II | III |
|---|---|---|---|---|
| | početno stanje | 1 | 0 | 0 |
| posmik | 1. | 0 | 1 | 0 |
| | 2. | 0 | 0 | 1 |
| | 3. | 1 | 1 | 0 |
| | 4. | 0 | 1 | 1 |
| | 5. | 1 | 1 | 1 |
| | 6. | 1 | 0 | 1 |

*Slika 3.3: Sadržaji posmičnoga registra nakon svakoga posmika*

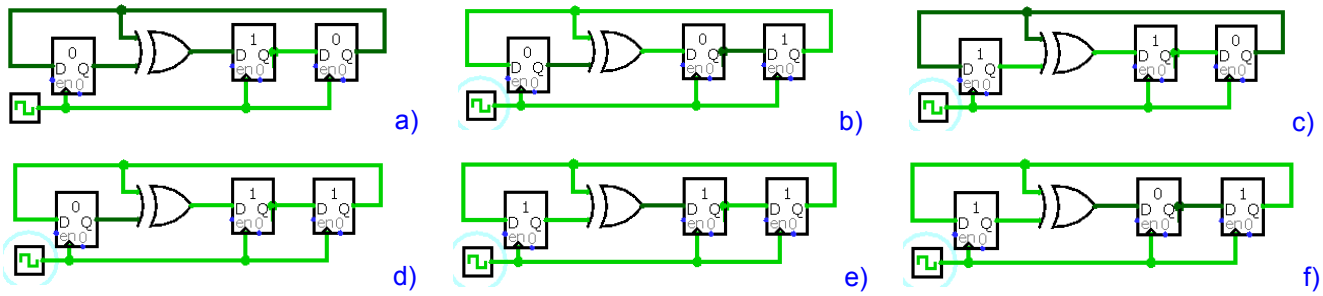
Rezultat je matrica $\mathbf{H}^T$ prikazana na slici 3.4:




$$\mathbf{H}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hline 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

*Slika 3.4: Povezanost stanja posmičnoga registra i paritetne matrice* $\boldsymbol{H}^T$

To se postiže sklopom na slici 3.4 u 7 koraka uključujući kao prvi, gornji korak, plus koraci a-f na slici 3.5.




Sljedeći posmik vraća sustav u početno stanje 100 prikazano na slici 3.4.

*Slika 3.5: Sadržaji LFSR u 6 posmika*

Kao što se jasno vidi, uzastopni sadržaji (nakon svakoga posmika) kruga na slici 3.4, retci su matrice $\mathbf{H}^T$. Treba objasniti da prilikom generiranja paritetne matrice koda, broj registara generatora, jednaka je broju paritetnih jednadžbi koda. *Dužina registara* (broj stanja) predstavlja *dužinu retka matrice* (= *broj stupaca u matrici*), a *svaki stupac* (po okomici odozgo prema dolje) predstavlja *paritetnu jednadžbu*.

*Definicija* Krug s povratnim vezama kojime upravljaju samo XOR vrata, zove se "**linearni posmični registar s povratnom vezom**" LFSR (*linear feedback shift register*).

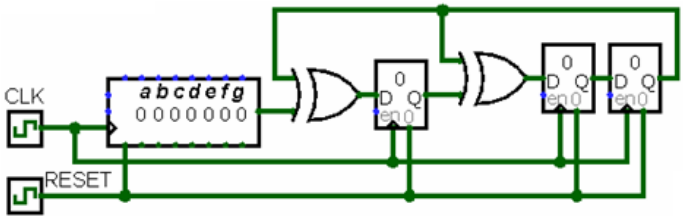
Linearnost takvoga kruga temelji se na sljedećoj očitoj činjenici: Ako su $S_1$, $S_2$, $S_3$, …, stanja dobivena posmikom u registru, počevši početnim stanjem $S_0$, a $T_1$, $T_2$, $T_3$, …, stanja dobivena posmikom u registar, počevši stanjem $T_0$, onda su $S_1+T_1$, $S_2+T_2$, $S_3+T_3$, … stanja dobivena posmikom u registar, počevši od početnoga stanja $S_0+T_0$.

## 3.2. 3.2 Množenje poruke paritetnom matricom[53] (dekodiranje i sindrom)

Množenje vektora [*abcdefg*] matricom pariteta $\mathbf{H}^T$, prikazuje se u nastavku:

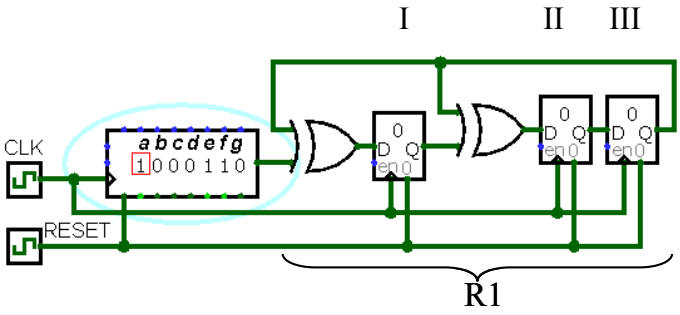
$$[abcdefg]\cdot\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \hline 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = [\underbrace{a \oplus d \oplus f \oplus g}_{\text{I}},\ \underbrace{b \oplus d \oplus e \oplus f}_{\text{II}},\ \underbrace{c \oplus e \oplus f \oplus g}_{\text{III}}]$$

[53] Napraviti vježbu!




*Slika 3.6: Množenje vektora matricom pariteta*

Sada se pokazuje kako automatski primijeniti prethodno množenje. Slika 3.7 prikazuje kako se vektor [*a b c d e f g*] množi matricom $\mathbf{H}^T$.




*Slika 3.7: Množenja vektora matricom* $\boldsymbol{H}^T$

Vektor se pohranjuje u posmični registar i posmiče se u R1 čiju osnovnu strukturu prikazuje slika 3.4, a čiji početan sadržaj jednak je [000]. Zapišimo sadržaje R1 tijekom sedam uzastopnih posmika cijeloga kruga:

| Broj posmika: | I | II | III | Sadržaj R1 za ulazni vektor 1111111 |
|---|---|---|---|---|
| početno stanje | 0 | 0 | 0 | 000 |
| 1. | $g$ | 0 | 0 | 100 |
| 2. | $f$ | $g$ | 0 | 110 |
| 3. | $e$ | $f$ | $g$ | 111 |
| 4. | $g \oplus d$ | $e \oplus g$ | $f$ | 001 |
| 5. | $f \oplus c$ | $f \oplus g \oplus d$ | $e \oplus g$ | 010 |
| 6. | $b \oplus e \oplus g$ | $e \oplus g \oplus f \oplus c$ | $f \oplus g \oplus d$ | 101 |
| početno 7. | $a \oplus f \oplus g \oplus d$ | $f \oplus g \oplus d \oplus b \oplus e \oplus g$ | $e \oplus g \oplus f \oplus c$ | 000 |

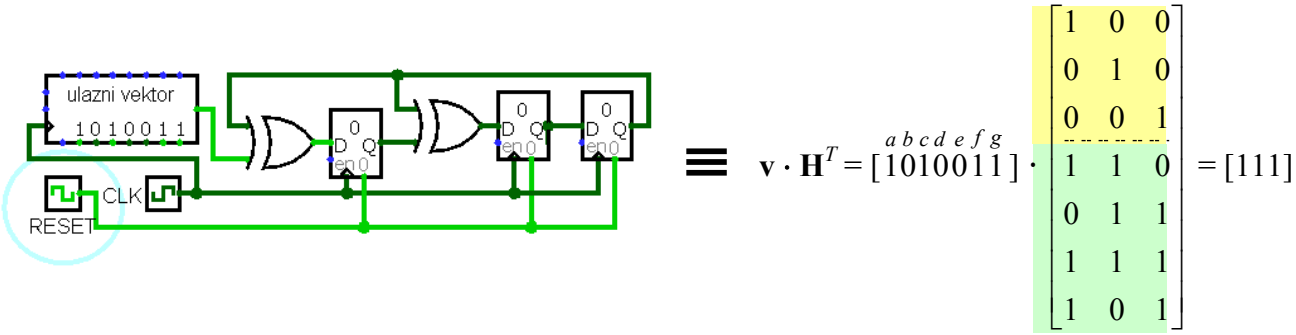
Izraz konačnoga sadržaja srednjega registra (II) osim 6 prikazanih među-stanja [0, $g$, $f$, $e \oplus g$, $f \oplus g \oplus d$, $e \oplus g \oplus f \oplus c$], nakon 7. posmika je: $f \oplus g \oplus d \oplus b \oplus e \oplus g$. Imajte na umu da se nakon 7. posmika, $g$ pojavljuje dva puta. Budući da $\oplus$ znači XOR, a znamo da XOR bita samoga sobom vodi 0 bez obzira na vrijednost bita, slijedi da je $g \oplus g = 0$ pa je $f \oplus g \oplus d \oplus b \oplus e \oplus g = f \oplus d \oplus b \oplus e$. Konačni sadržaji sva 3 memorijska elementa [I, II i III] registra R1 (pravilo množenja vektora i matrice) onda su:

$$\underbrace{[1 \cdot a \oplus 0 \cdot b \oplus 0 \cdot c \oplus 1 \cdot d \oplus 0 \cdot e \oplus 1 \cdot f \oplus 1 \cdot g]}_{\text{I}}, \underbrace{[0 \cdot a \oplus 1 \cdot b \oplus 0 \cdot c \oplus 1 \cdot d \oplus 1 \cdot e \oplus 1 \cdot f \oplus 0 \cdot g]}_{\text{II}}, \underbrace{[0 \cdot a \oplus 0 \cdot b \oplus 1 \cdot c \oplus 0 \cdot d \oplus 1 \cdot e \oplus 1 \cdot f \oplus 1 \cdot g]}_{\text{III}}$$

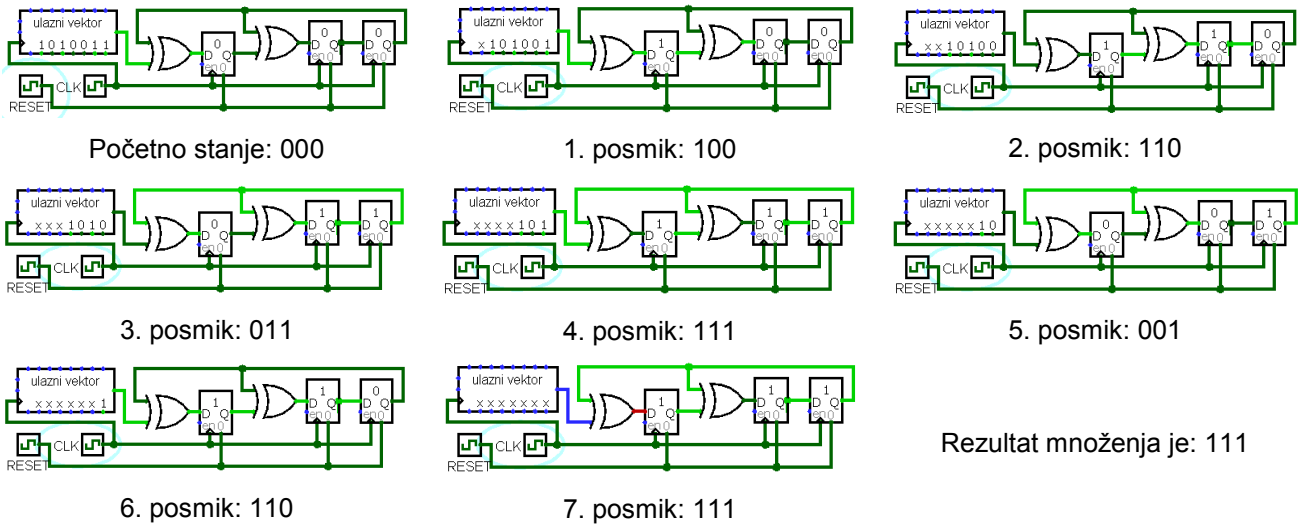
$$[\underbrace{a \oplus d \oplus f \oplus g}_{\text{I}}, \underbrace{b \oplus d \oplus e \oplus f}_{\text{II}}, \underbrace{c \oplus e \oplus f \oplus g}_{\text{III}}],$$

Dakle, samo jedinice u svakome stupcu matrice $\mathbf{H}^T$, množe se svakom znamenkom ulaznoga vektora, jer množenje nulom bilo koje znamenke ulaznoga vektora (1 ili 0), kao rezultat daje 0. Dobio se upravo prethodno prikazan rezultat *množenja*.

Primjer: Ulazni vektor **v** = [$\overset{a\ b\ c\ d\ e\ f\ g}{1\,0\,1\,0\,0\,1\,1}$]. U sklop prvi ulazi desni (g) bit [$\overset{g\ f\ e\ d\ c\ b\ a}{1\,1\,0\,0\,1\,0\,1}$] (slika 3.8).




*Slika 3.8: Množenje vektora i matrice*




*Slika 3.9: Množenje vektora i matrice korak po korak*

Primjer: $[1010011] \cdot \mathbf{H}^T$ = (zbroj 1., 3., 6. i 7. retka matrice $\mathbf{H}^T$).

Rezultat:

| | | |
|---|---|---|
| 100 | | 1. redak matrice $\mathbf{H}^T$ |
| 001 | | 3. redak matrice $\mathbf{H}^T$ |
| 111 | | 6. redak matrice $\mathbf{H}^T$ |
| 101 | $\oplus$ | 7. redak matrice $\mathbf{H}^T$ |
| 111 | ← | konačan rezultat množenja |

$$\begin{matrix} 1.\text{redak} \\ 2.\text{redak} \\ 3.\text{redak} \\ 4.\text{redak} \\ 5.\text{redak} \\ 6.\text{redak} \\ 7.\text{redak} \end{matrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \mathbf{H}^T$$

Drugim riječima, posmikom kruga na slici 3.2 sedam puta, zapravo množimo vektor [*abcdefg*] matricom $\mathbf{H}^T$, gdje je rezultat ovoga umnoška, zadnji sadržaj LFSR. Ako je [*abcdefg*] primljena poruka čija je prenesena inačica kodna riječ (dakle radi se o dekoderu), sadržaj tih stanja postaje *sindrom pogreške*. Dekoder automatski stvara sindrom pogreške (sadržaji registra nakon 7. posmika), bez potrebe pospremanja sadržaja $\mathbf{H}^T$.

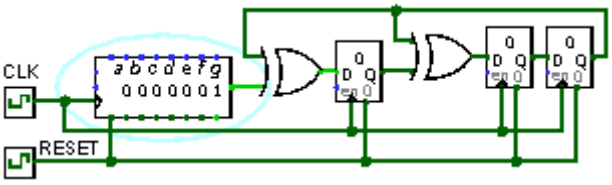
### 3.2.1. 3.2.1. ANALIZA POSMIKA I PREPOZNAVANJE MNOŽENJA

Sada ćemo pobliže analizirati prethodno opisan postupak i vidjeti koja obilježja kruga na slici 3.2 (dekodera) omogućuju operaciju množenja. Prvo, uzmimo slučaj gdje se vektor [0000001] množi matricom $\mathbf{H}^T$. Označili smo ovaj vektor kao $\mathbf{v}_7$ (indeks 7 označava položaj 1-elementa u vektoru, gdje brojanje počinje s lijeva).

Umnožak ovoga vektora matricom $\mathbf{H}^T$ daje zadnji redak matrice (tj., $\mathbf{v}_7 \cdot \mathbf{H}^T = [101]$). Pogledajmo zašto se isti rezultat dobije posmikom $\mathbf{v}_7$ u sklop na slici 3.10.

$$\mathbf{v}_7 \cdot \mathbf{H}^T = [0000001] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = [101] \equiv$$

[0000001]



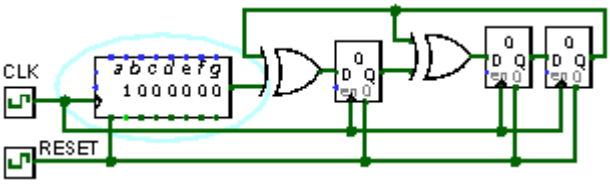

sklop na slici 3.7

*Slika 3.10: Posmik vektora $\mathbf{v}_7$ u sklop na slici 3.7*

Pri prvome posmiku 1 u desno, $\mathbf{v}_7$ se posmiče u LFSR. Tijekom sljedećih šest posmika (kada se posmiče ostatak $\mathbf{v}_7$), vrijednosti bitova kojima se puni LFSR su "sve 0". Drugim riječima, od drugoga do sedmoga posmiku, LFSR djeluje kao generator na slici 3.4, počevši početnim stanjem [100]. Njegov konačan sadržaj je onda [101], što je posljednji redak u $\mathbf{H}^T$. Prije smo objasnili da se posmikom $\mathbf{v}_7$ u krugu na slici 3.7, zapravo obavlja operacija $\mathbf{v}_7 \cdot \mathbf{H}^T$.

Uzmimo sada slučaj gdje se vektor $\mathbf{v}_1 = [1000000]$ množi matricom $\mathbf{H}^T$. Imamo da je $\mathbf{v}_1 \cdot \mathbf{H}^T = [100]$. Ovo je isti rezultat koji se postiže pomicanjem $\mathbf{v}_1$ u sklop na slici 3.7.

[1000000]




sklop na slici 3.7

*Slika 3.11: Posmik vektora $\mathbf{v}_1$ u sklop na slici 3.7*

Da bi se vidjelo zašto je to tako, primijetimo da, kada se prvih šest 0 od $\mathbf{v}_1$ pomakne u LFSR, on će i dalje sadržavati 0. U posljednjemu posmiku, 1 na lijevoj strani $\mathbf{v}_1$ posmiče se u LFSR pa se dobije [100], što je i konačan sadržaj registra jednak izračunatome rezultatu $\mathbf{v}_1 \cdot \mathbf{H}^T = [100]$.

Općenito, neka $\mathbf{v}_i$ označava vektor od sedam bitova, koji ima samo jedan element vrijednosti 1 na mjestu $i$. Tvrdimo da ako se $\mathbf{v}_i$ posmiče krugom na slici 3.2, konačan sadržaj LFSR bit će $i$-ti redak matrice $\mathbf{H}^T$, što je upravo rezultat operacije $\mathbf{v}_i \cdot \mathbf{H}^T$. Ova tvrdnja dokazuje se promatranjem da nakon posmika prvih 7–$i$ nula (0) desno od $\mathbf{v}_i$, LFSR i dalje sadržava 0. Onda se 1 ($v_i$) posmiče u krug i on se posmiče još $i$–1 put. Krug se onda ponaša kao generator na slici 3.4, rezultirajući $i$-*tim* retkom od $\mathbf{H}^T$ kao konačnim sadržajem LFSR.

Sada ćemo vidjeti što će se dogoditi ako se *općenit* vektor $\mathbf{v}$, dužine 7, posmiče krugom na slici 3.2. Takav $\mathbf{v}$ ima 1-elemente na mjestima $i$, $j$, $k$, itd. On se sastoji od zbroja vektora $\mathbf{v}_i$, $\mathbf{v}_j$, $\mathbf{v}_k$, itd. Pošto je naš sustav *linearan*, konačan sadržaj LFSR, nakon posmika $\mathbf{v}$ u krug, jednak je rezultatu dobivenome zasebnim posmikom $\mathbf{v}_i$, $\mathbf{v}_j$, $\mathbf{v}_k$, itd. te zbrajanjem ovih odvojenih rezultata. Međutim, ovi rezultati su odvojeni rezultati $i$-toga, $j$-toga i $k$-toga retka matrice $\mathbf{H}^T$. Njihov zbroj je prema definiciji jednak rezultatu dobivenome operacijom $\mathbf{v} \cdot \mathbf{H}^T$. Dakle, dokazali smo da krug na slici 3.7 općenito množi vektor i matricu.

Primjer: $[1010100] \cdot \mathbf{H}^T =$ (zbroj 1., 3. i 5. retka matrice $\mathbf{H}^T$).

Rezultat:

| | | |
|---|---|---|
| 100 | | 1. redak matrice $\mathbf{H}^T$ |
| 001 | | 3. redak matrice $\mathbf{H}^T$ |
| 011 | $\oplus$ | 5. redak matrice $\mathbf{H}^T$ |
| 110 | ← | konačan rezultat množenja |

$$\begin{matrix} 1.\text{ redak} \\ 2.\text{ redak} \\ 3.\text{ redak} \\ 4.\text{ redak} \\ 5.\text{ redak} \\ 6.\text{ redak} \\ 7.\text{ redak} \end{matrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = \mathbf{H}^T$$

## 3.3. 3.3. Postupak kodiranja

U prethodnome dijelu pokazali smo kako pri dekodiranju, automatski izračunati sindrom pogreške, na temelju nekih posebnih svojstava $\mathbf{H}^T$. U ovome dijelu pokazat ćemo kako automatski generirati *kodnu riječ* iz *informacijskoga vektora*. Ovo je postupak *kodiranja*. Već smo rekli da je naš kod sustavan, što znači da su paritetni bitovi pridruženi informacijskome vektoru kao kontinuirana skupina (lijevo ili desno).

### 3.3.1. 3.3.1. STVARANJE PARITETNIH BITOVA

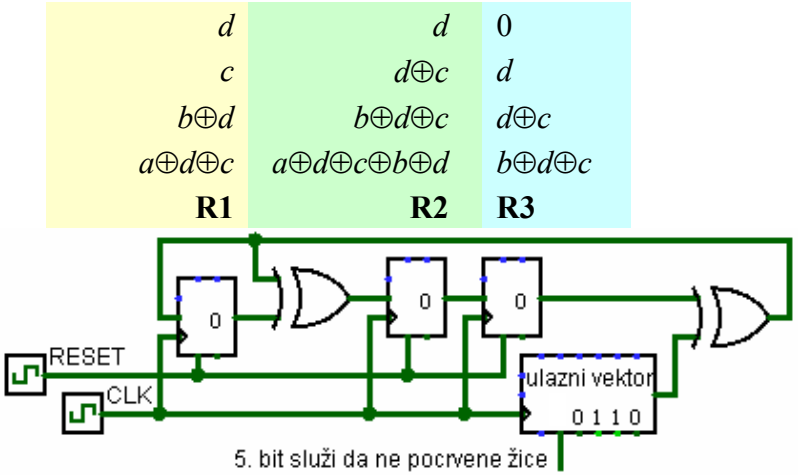
Paritetne jednadžbe, predstavljene stupcima $\mathbf{H}^T$, su:

(1) $p_0 \oplus a \oplus c \oplus d = 0$

(2) $p_1 \oplus a \oplus b \oplus c = 0$

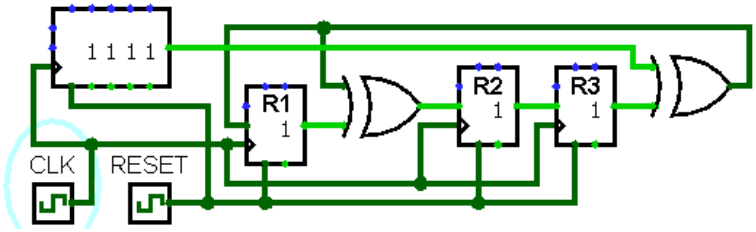
(3) $p_2 \oplus b \oplus c \oplus d = 0$

Slika 3.12 prikazuje kako se iz informacijskih bitova, automatski računaju paritetni bitovi.




*Slika 3.12: Koder koda čija matrica pariteta je $\mathbf{H}^T$*

Bitovima *abcd*, informacijski vektor puni LFSR (obrnutim redoslijedom - prvi ulazi bit *d*) slično onome na slici 3.4. Napravimo posmik kruga na slici 3.12 četiri puta i zapišimo sadržaje registara LFSR nakon svakoga posmika (slika 3.13):




*Slika 3.13: Sadržaji registara LFSR nakon svakoga od 16 posmika (prethodna i sljedeća tablica)*

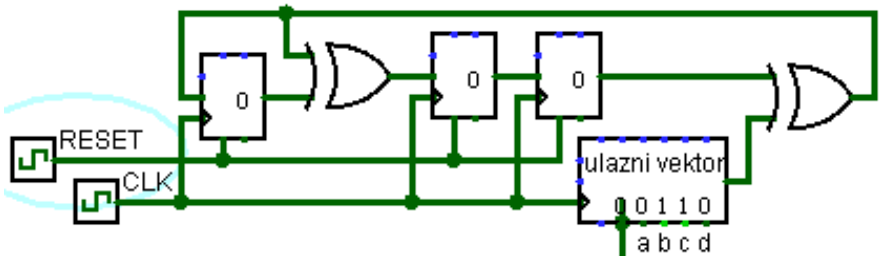
| Redni broj riječi ulaznoga vektora | | | 1. | 2. | 3. | 4. | 5. | 6. | 7. | 8. |
|---|---|---|---|---|---|---|---|---|---|---|
| Ulazni vektori = | | | 1111 | 0111 | 1011 | 0011 | 1101 | 0101 | 1001 | 0001 |
| R1 | R2 | R3 | *abcd* | *abcd* | *abcd* | *abcd* | *abcd* | *abcd* | *abcd* | *abcd* |
| | | | *sadržaji registara za ulazne vektore* | | | | | | | |
| $d$ | $d$ | 0 | 110 | 000 | 110 | 000 | 110 | 000 | 110 | 110 |
| $c$ | $d \oplus c$ | $d$ | 101 | 110 | 011 | 000 | 101 | 110 | 011 | 011 |
| $b \oplus d$ | $b \oplus d \oplus c$ | $d \oplus c$ | 010 | 101 | 001 | 110 | 100 | 011 | 111 | 111 |
| $a \oplus d \oplus c$ | $a \oplus d \oplus c \oplus b \oplus d$ | $b \oplus d \oplus c$ | 111 | 010 | 000 | 101 | 100 | 001 | 011 | 101 |
| $p_0$ | $p_1$ | $p_2$ | ⟵ sadržaji LFSR registara su paritetni bitovi | | | | | | | |

Napomena: Svejedno je kojim se sklopom [gornjim (↑) slika 3.13 ili donjim (↓) slika 3.14] posmiče ulazni vektor.

| Redni broj riječi ulaznoga vektora | | | 9. | 10. | 11. | 12. | 13. | 14. | 15. | 16. |
|---|---|---|---|---|---|---|---|---|---|---|
| Ulazni vektori = | | | 1110 | 0110 | 1010 | 0010 | 1100 | 0100 | 1000 | 0000 |
| R1 | R2 | R3 | *abcd* | *abcd* | *abcd* | *abcd* | *abcd* | *abcd* | *abcd* | *abcd* |
| | | | *sadržaji registara za ulazne vektore* | | | | | | | |
| $d$ | $d$ | 0 | 000 | 000 | 000 | 000 | 000 | 000 | 000 | 000 |
| $c$ | $d \oplus c$ | $d$ | 110 | 110 | 110 | 110 | 000 | 000 | 000 | 000 |
| $b \oplus d$ | $b \oplus d \oplus c$ | $d \oplus c$ | 101 | 101 | 011 | 011 | 110 | 110 | 000 | 000 |
| $a \oplus d \oplus c$ | $a \oplus d \oplus c \oplus b \oplus d$ | $b \oplus d \oplus c$ | 010 | 100 | 001 | 111 | 010 | 011 | 110 | 000 |
| $p_0$ | $p_1$ | $p_2$ | ⟵ sadržaji LFSR registara su paritetni bitovi | | | | | | | |

Imajte na umu da je $a \oplus d \oplus c \oplus b \oplus d = a \oplus b \oplus c$, jer se dva $d$ međusobno poništavaju (XOR). Onda su konačni sadržaji u: R1 = $a \oplus d \oplus c$, R2 = $a \oplus b \oplus c$ i R3 = $b \oplus d \oplus c$. To su zapravo vrijednosti $p_0$, $p_1$, $p_2$ na što upućuju jednadžbe (1), (2) i (3).

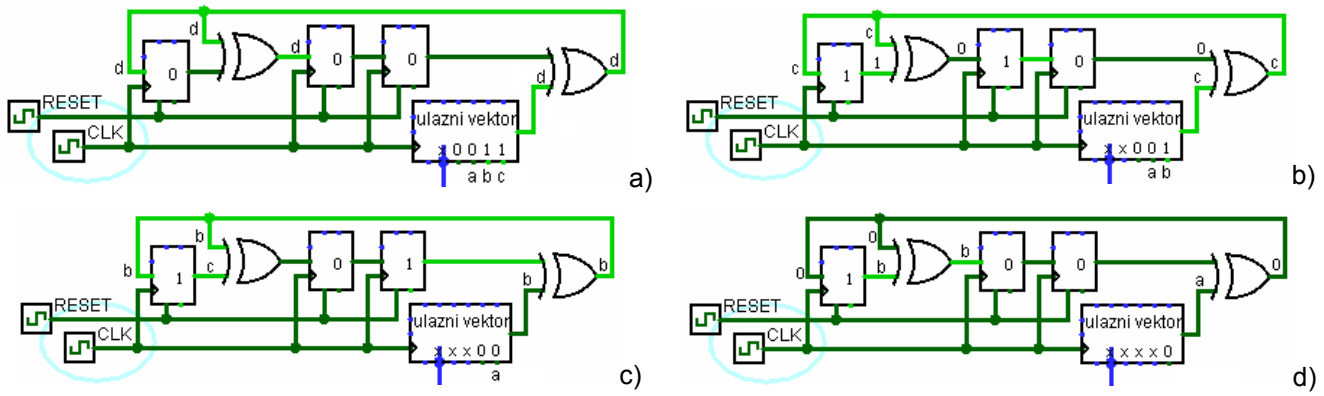
Primjer:




*Slika 3.14: Početno stanje LFSR kodera (7, 4) koda.*

Slika prikazuje kako se informacijskome vektoru od 4 bita $\mathbf{u} = [\overset{a\,b\,c\,d}{0110}]$, pridružuju stanja LFSR nakon 4 posmika s lijeve ili s desne strane.[54] Uzastopni sadržaji LFSR su:

| | R1 | | R2 | | R3 | | |
|---|---|---|---|---|---|---|---|
| a) | $d$ | = 0 | $d$ | = 0 | 0 | = 0 | 1. posmik |
| b) | $c$ | = 1 | $c \oplus d$ | = 1 | $d$ | = 0 | 2. posmik |
| c) | $b \oplus d$ | = 1 | $b \oplus c \oplus d$ | = 0 | $c \oplus d$ | = 1 | 3. posmik |
| d) | $a \oplus c \oplus d$ | = 1 | $a \oplus b \oplus c$ | = 0 | $b \oplus c \oplus d$ | = 0 | 4. posmik |
| | $p_0$ | | $p_1$ | | $p_2$ | | paritetni bitovi |




*Slika 3.15: Stanja registara nakon svakoga od 4 posmika*

Time smo pokazali da sklop na slici 3.12 automatski stvara paritetne bitove. Nakon što su se stvorili, jednostavno ih pridružimo (kao rep ali s lijeve strane) informacijskim bitovima, oblikujući kodnu riječ:

$$p_0\, p_1\, p_2\, a\, b\, c\, d,$$

ili posebno za ovaj primjer i za informacijski vektor $\mathbf{u} = [0110]$, kodirana riječ je:

$$[1\,0\,0\,0\,1\,1\,0].$$

[54] Automatizirati pridruživanje bitova stanja. (Napraviti i usporediti s tablicom 2.1) Također napraviti C++ program.

Prethodna rasprava bila je samo proba, a ne dokaz. Nismo dali nikakvo suvislo obrazloženje, objašnjavajući kako i zašto smo u stanju oblikovati takav sklop. Da bismo vidjeli zašto krug na slici 3.15, stvara paritetne bitove koji odgovaraju informacijskome vektoru, opet prikazujemo generator-matricu koda:

$$\mathbf{G} = \overbrace{\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}}^{\mathbf{P}} \overbrace{\left|\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}\right.}^{\text{jedinična matrica}}$$

Paritetni bitovi, pridruženi informacijskome vektoru [1000110], oblikuju jedno-redan vektor (umnožak) $\mathbf{u} \cdot \mathbf{P}$ (matricama $\mathbf{L}^T$ i **G**, zajednička pod-matrica je **P** - vidi gornji prikaz matrice **G**). Množenje ulaznoga vektora **u**, jediničnom matricom kao desnim dijelom matrice **G**, jednako je praktičnoj situaciji u kojoj se **u** izravno prenosi kao dio informacijskoga vektora.

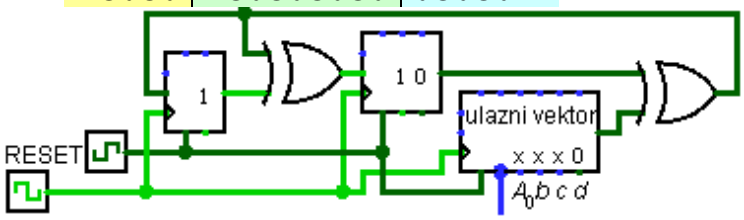
### 3.3.2. 3.3.2. UMNOŽAK INFORMACIJSKOGA VEKTORA I POD-MATRICE

Sada pokazujemo zašto krug na slici 3.15 ostvaruje operaciju $\mathbf{u} \cdot \mathbf{P}$, gdje je $\mathbf{u} = [a, b, c, d]$. Retci pod-matrice **P** četiri su uzastopna sadržaja generatora prikazana na slici 3.1, počevši od [110].

$$\mathbf{u} \cdot \mathbf{P} = [0\ 1\ 1\ 0] \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = [1\ 0\ 0]$$

Imajte na umu da LFSR na slici 3.3 predstavlja isti generator kao i na slici 3.15. Pogledajmo što će se dogoditi ako se informacijski vektor $\mathbf{u} = [A_0, 0, 0, 0]$ pomakne u krug na slici 3.16, gdje je $A_0$ jednak 0 ili 1 (ali ostali bitovi vektora **u**, sigurno su "sve 0").

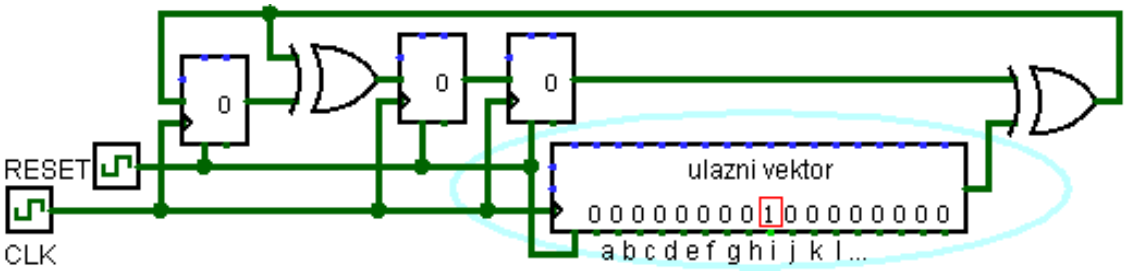
| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0⊕0 | 0 |
| 0⊕0 | 0⊕0⊕0 | 0⊕0 |
| 1⊕0⊕0 | 1⊕0⊕0⊕0⊕0 | 0⊕0⊕0 |




*Slika 3.16: Posmik informacijskoga vektora* $[A_0, 0, 0, 0]$ *u LFSR*

Dokle god se $A_0$ ne posmikne, LFSR još uvijek sadrži "sve 0". Kada se posmiče $A_0$ i događa se posljednji posmik, sadržaj LFSR je $A_0 \cdot [110]$. Ako je $A_0 = 1$, ova jedinica posmikne se u dva lijeva stanja registra, tvoreći sadržaj [110]. Ako je $A_0 = 0$, registar će i dalje sadržavati samo nule. Imajte na umu da je [110] prvi redak u pod-matrici **P**, što znači da su paritetni bitovi pridruženi informacijskome vektoru $\mathbf{u} = [A_0, 0, 0, 0]$, zapravo umnožak $\mathbf{u} \cdot \mathbf{P}$.

Općenito, neka je **u** informacijski vektor koji na mjestu *i* ima $A_i$, računajući s lijeve strane, gdje je prvo mjesto označeno kao #0. (Ostatak elemenata **u**, osim $A_i$, su "sve 0"). Ako se vektor **u** posmiče u krug na slici 3.16, LFSR će napraviti posmik 4–*i*–1 put, sadržavajući "sve 0". Idućim posmikom, posmiče se element $A_i$ pa se dobije sadržaj $A_i \cdot [110]$. Nakon toga posmika, LFSR nastavlja radom, ali sada kao generator na slici 3.4, za *i* posmika (i nastavlja se puniti nulama, slika 3.17).

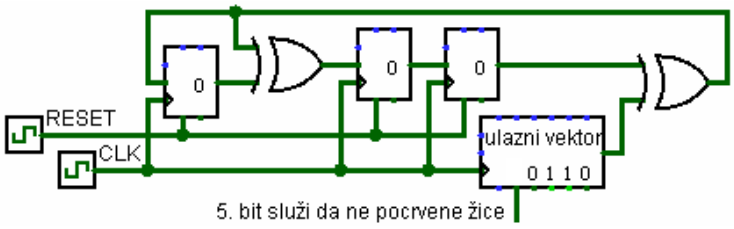



*Slika 3.17: Posmik ulaznoga vektora i nastavak punjenja nulama*

Njegov konačan sadržaj bit će *i*-ti redak pod-matrice **P** ako je $A_i = 1$, a "sve 0" ako je $A_i = 0$, tj., ovdje imamo pridružene paritetne bitove informacijskome vektoru **u** kao **u** · **P**. Na temelju linearnosti registra, onda bismo te paritetne bitove pridružili općemu informacijskomu vektoru **u** = [$A_0$, $A_1$, $A_2$, $A_3$, …] iz vektora **u** · **P**. Iako se naš *dokaz* smatra posebnim za (7, 4) kod, on je dovoljno općenit za inačice koda bilo koje dimenzije.

Sada možemo zaključiti da *i koder i dekoder koriste LFSR **jednake** strukture*.

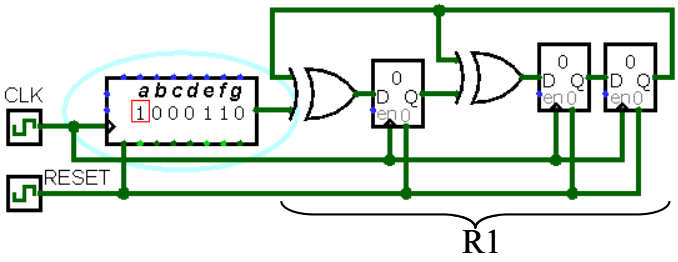
- U *koderu*, informacijski vektor puni se u zadnje (desno) stanje registra, u kojemu konačan sadržaj registra tvore paritetni bitovi koji se pridružuju ovome informacijskome vektoru (lijevo ili desno od njega), kao što prikazuje slika 3.18.




$R(x) = 1 \oplus x \oplus x^3$

*Slika 3.18: Pridruživanje paritetnih bitova informacijskome vektoru*

- U *dekoderu*, primljena poruka puni se u prvo (lijevo) stanje u registru, gdje konačan sadržaj registra čini ***sindrom pogreške***, kao što prikazuje slika 3.7 ili slika 3.19.




$R(x) = 1 \oplus x \oplus x^3$

*Slika 3.19: Sindrom pogreške kao konačan sadržaj registra*

*Definicija* LFSR, što se koristi u koderu i dekoderu koda, naziva se **LFSR za stvaranje koda** (*the generating LFSR of the code*).

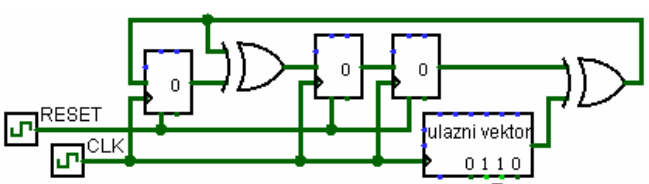
# 7. *Laboratorijska vježba 6: Standardno polje za (8, 2) kod*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

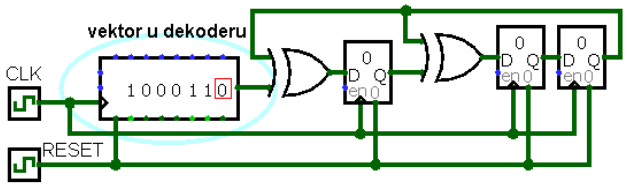
## 3.4. 3.4. Kodiranje/dekodiranje vektora kraćih i dužih od 4 bita

Razmotrimo sada slučajeve gdje se u LFSR na slici 3.12 posmiču informacijski vektori, *kraći* ili *duži* od četiri (4) bita, a zatim se sadržaj LFSR pridružuje informacijskomu vektoru. Još uvijek dobivamo kodnu riječ čiji *dekoder* je krug na slici 3.7. Posmikom vektora stvorenoga koderom kruga na slici 3.7 i dalje ćemo dobivati stanja "sve 0" kao konačan sadržaj.

Onda su slika 3.7 (*množenje vektora matricom* $\mathbf{H}^T$) i slika 3.12 (*koder koda čija matrica pariteta je* $\mathbf{H}^T$), par koder-dekoder za bilo koju duljinu informacijskih vektora (slika 3.20).




*Koder-množenje* vektora *matricom* $\mathbf{H}^T$ (slika 3.7)




*Dekoder*, čija *matrica pariteta je* $\mathbf{H}^T$ (slika 3.12)

*Slika 3.20: Koder (puni se na izlazu) i dekoder (puni se na ulazu) imaju istu strukturu* $R(x)=1+x+x^3$

### 3.4.1. 3.4.1. INFORMACIJSKI VEKTOR DULJINE 3 BITA

Osnova ovoga opažanja je, da se rasprave ovdje i u prethodnome poglavlju, uopće ne odnose na činjenicu da je *duljina informacijskoga* vektora jednaka 4 bita. Ako je *duljina* jednaka 3 bita, paritetna matrica koda **P** i generator-matrica koda **G**, izgledat će:

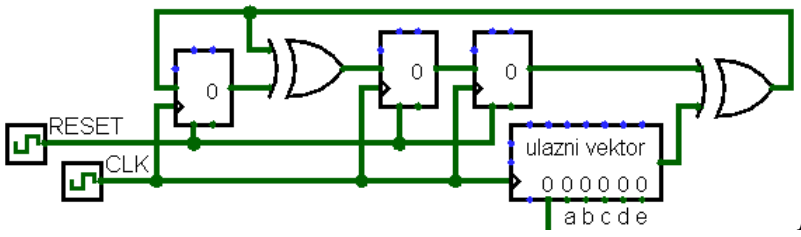
$$\mathbf{H}_6^T = \mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \Rightarrow [\mathbf{P}_3 \mid \mathbf{I}_3],$$

Dakle, vidi se da nema 4. posmika pa tako nema ni 4. (zadnjega) retka u matrici **P** što se posredno odražava i na matricu **G**. Podsjetimo se da je donji dio matrice **P** zapravo prethodno opisana matrica **P** ali bez zadnjega retka (jer nema 4. posmika):

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ \hline 1 & 0 & 1 \end{bmatrix}$$

### 3.4.2. 3.4.2. DULJINA INFORMACIJSKOGA VEKTORA JEDNAKA JE 5 BITOVA

Promotrimo slučaj da je informacijski vektor **t** = [*a*, *b*, *c*, *d*, *e*] duljine 5 bitova i posmiče ga se u krug kodiranja na slici 3.12 (slika 3.21)




$R(x) = 1 \oplus x \oplus x^3$

*Slika 3.21: Posmika informacijskoga vektora duljine 5 bitova u LFSR koder*

Iz objašnjenja danoga u poglavlju 3.3, znamo da su konačni sadržaji registara R1…R3:

| e = 0 | e = 1 | d | c | b | a | |
|---|---|---|---|---|---|---|
| 1. posmik | | 2. posmik | 3. posmik | 4. posmik | 5. posmik | Konačni sadržaji R1…R3 |
| 000 | 110 | *d* · [101] | *c* · [111] | *b* · [011] | *a* · [110] | |
| | | | | | | |

$$a \cdot [110] + b \cdot [011] + c \cdot [111] + d \cdot [101] + [\text{doprinos od } e].$$

- Ako je $e = 0$, sadržaji R1…R3 nakon prvoga posmika su [000].
- Ako je $e = 1$, sadržaji R1…R3 nakon prvoga posmika su [110].

Nakon toga, posmiče ga se još četiri puta. Doprinos $e$ konačnome sadržaju R1…R3, jednak je završnome sadržaju generatora koji slobodno radi na slici 3.4, nakon 4. posmika, polazeći od početnoga sadržaja [110]. Konačan sadržaj je [100]. Onda su paritetni bitovi što ih generira koder jednaki $[a, b, c, d, e] \cdot \mathbf{G}$, gdje generator-matrica $\mathbf{G}$ ima sljedeći oblik ($n$=8, $k$=5). Opće polazište je:

$$\mathbf{G} = [\mathbf{P}_{n-k} \mid \mathbf{I}_k] \Rightarrow \mathbf{H} = [\mathbf{I}_{n-k} \mid \mathbf{P}_{n-k}^T] \Rightarrow \quad \mathbf{H}^T = \begin{bmatrix} \mathbf{I}_{n-k} \\ \hline \mathbf{P}_{n-k} \end{bmatrix}$$

$$\mathbf{G} = [\mathbf{P}_3 \mid \mathbf{I}_5] \Rightarrow \mathbf{H} = [\mathbf{I}_{8-5} \mid \mathbf{P}^T] \Rightarrow \mathbf{H}^T = \begin{bmatrix} \mathbf{I}_3 \\ \hline \mathbf{P}_3 \end{bmatrix} = \mathbf{P}_7 \text{ (za ovaj dio skripte),}$$

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \Rightarrow \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \Rightarrow \mathbf{H}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Na temelju povezanosti *matrice pariteta* i *generator-matrice* sustavnoga koda, imamo da je paritetna matrica $\mathbf{P}_7$ našega koda:

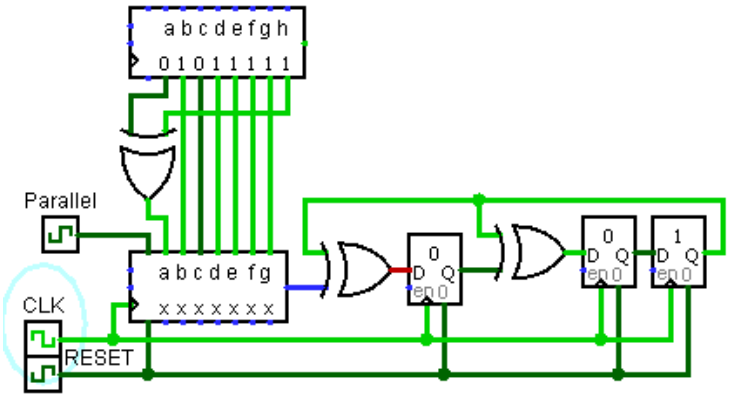
$$\mathbf{H}^T = \mathbf{P}_7 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$ (Važna napomena: 1. redak [100] ponavlja se i kao 8. redak)

Prvi i zadnji red matrice $\mathbf{P}$ je [100]. To znači da naš (8, 5) kod ne može ispraviti jednu pogrešku, budući da pojava jedne pogreške na prvomu ili na zadnjemu mjesta u primljenoj poruci daje isti sindrom pogreške. Množenje poruke $\mathbf{t}$ duljine 8, matricom $\mathbf{P}$, obavlja se istim sklopom kao što je dekoder prikazan na slici 3.7. Sadržaj registara R1…R3, nakon što se poruka $\mathbf{t}$ posmikne u njega, je $\mathbf{t} \cdot \mathbf{P}_7$. Objašnjenje ove tvrdnje slijedi izravno iz detaljne rasprave u poglavlju 3.2.

Sada razmotrimo mogućnost izrade prečaca za izračun konačnoga sadržaja registara na slici 3.7, a nakon posmika vrlo duge poruke u njih. Ako se u registar posmiče poruka $\mathbf{t}$ duljine 8, tvrdi se da matematički obavljamo operaciju $\mathbf{t} \cdot \mathbf{P}_7$. Međutim, budući da se prvih sedam redaka $\mathbf{P}_7$ sastoji od $\mathbf{H}^T$, gdje je posljednji redak $\mathbf{P}_7$ jednak prvome retku $\mathbf{P}_7$, slijedi da je:

$$[a, b, c, d, e, f, g, h] \cdot \mathbf{P}_7 = [a + h, b, c, d, e, f, g] \cdot \mathbf{H}^T.$$

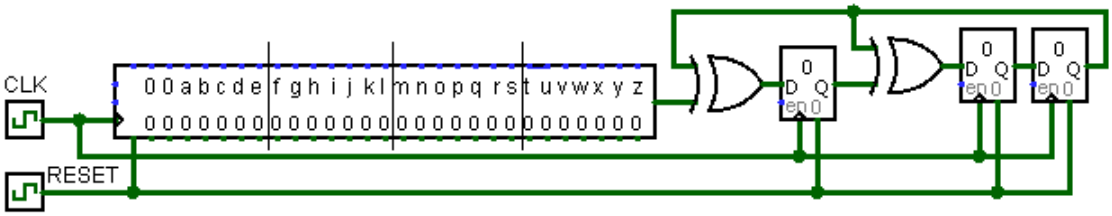
Umjesto posmika poruke dužine 8 u krug dekodera, možemo napraviti posmik poruke duljine 7 ako prethodno zbrojimo $a$ i $h$ (slika 3.22).




*Slika 3.22: Posmik poruke duljine 7, umjesto posmika duljine 8 ako se prethodno zbroje bitovi*

### 3.4.3. 3.4.3. PROIZVOLJNA DUŽINA VEKTORA

Općenito, pretpostavimo da želimo posmik poruke **t** = [*abcdefghijklmnopqrstuvwxyz*] u dekoder na slici 3.7 (slika 3.23).

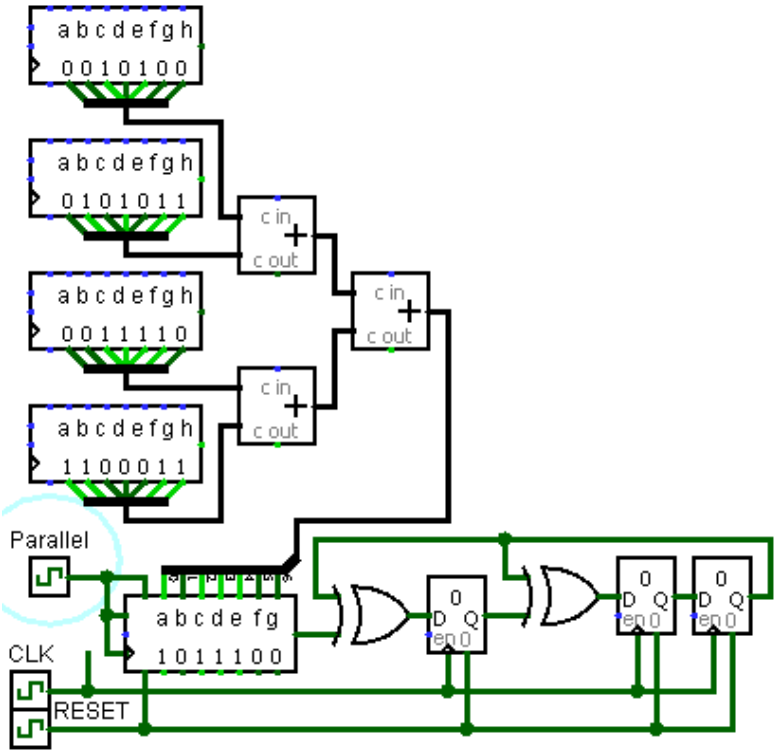



*Slika 3.23: Posmik poruke* [*abcdefghijklmnopqrstuvwxyz*] *u dekoder*

Možemo podijeliti **t** u četiri vektora duljine 7 kao što se prikazuje u nastavku. Napominjemo da se početku poruke (lijevo) moraju pridružiti dvije 0, da bi svi vektori imali duljinu 7:

$$\mathbf{t} = \underbrace{00abcde}_{\mathbf{a}}\ \underbrace{fghijkl}_{\mathbf{b}}\ \underbrace{mnopqrs}_{\mathbf{c}}\ \underbrace{tuvwxyz}_{\mathbf{d}}$$

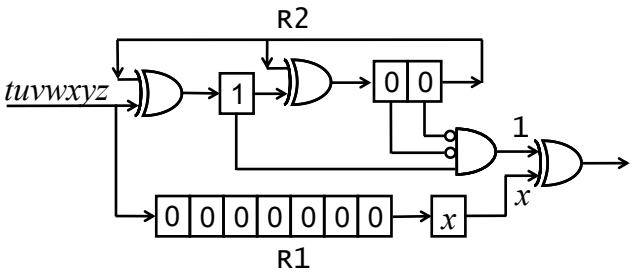
Ako je **u** = **a** ⊕ **b** ⊕ **c** ⊕ **d**, onda posmikom **t** u dekoder (za što će trebati 26 vremenskih odsječaka), ili posmikom **u** (za što će trebati sedam posmika), daje isti rezultat (slika 3.24)

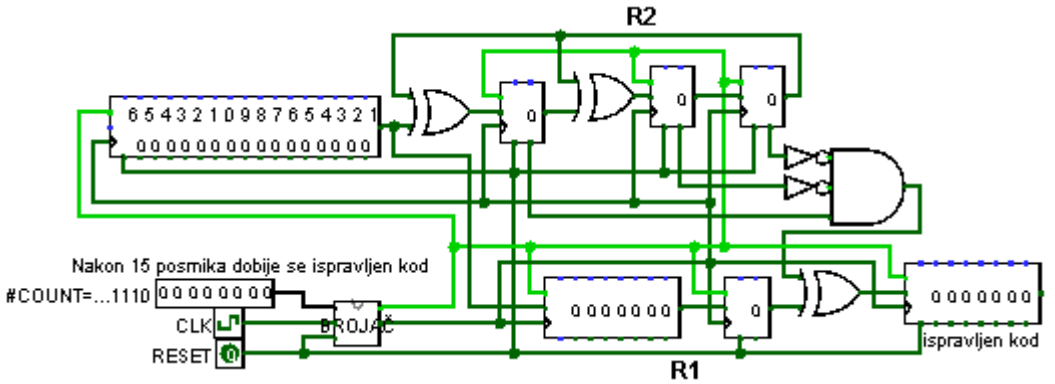



*Slika 3.24: Simulacijska shema zbrajanja 4 vektora (26 bitova)*

Automatsko stvaranje sindroma pogreške primljene poruke prikazuje slika 3.7. Još uvijek nismo pokazali kako ispraviti pogrešku znajući sindrom tj., kako otkriti položaj pogreške, uz pretpostavku da imamo samo jednu pogrešku. Slika 3.25 prikazuje cjelovit dekoder koji stvarno ispravlja jednostruku pogrešku.

| Tablica 2.1(c) (7, 4) kod | | | |
|---|---|---|---|
| izlaz | ulaz | izlaz | ulaz |
| 0000 000 | 0000 | 1000 110 | 1000 |
| 0001 101 | 0001 | 1001 011 | 1001 |
| 0010 111 | 0010 | 1010 001 | 1010 |
| 0011 010 | 0011 | 1011 100 | 1011 |
| 0100 011 | 0100 | 1100 101 | 1100 |
| 0101 110 | 0101 | 1101 000 | 1101 |
| 0110 100 | 0110 | 1110 010 | 1110 |
| 0111 001 | 0111 | 1111 111 | 1111 |
| Ako ima sadržaja, upis u dekoder je s lijeva | | | |




$$\mathbf{H}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{matrix} 1. \\ 2. \\ 3. \\ 4. \\ 5. \\ 6. \\ 7. \\ 8. \end{matrix}$$

*Slika 3.25: Cjelovit (Meggittov) dekoder (nakon 15 posmika dobije se ispravljena kodna riječ)*

U krug ulazi *sedam* bitova (koji su došli preko kanala, npr. [0011110]) *tuvwxyz*, a to je primljena poruka. Nakon posmika u krug sedam puta, počevši opisanim početnim uvjetima, sadržaj registra R1 je *tuvwxyz*. Registar R2 sadrži sindrom pogreške [011], kao što se detaljno opisalo u poglavlju 3.1. Pretpostavimo, na primjer, da je pogreška u primljenoj poruci na 5. mjestu s lijeva, tj., bit *x* je pogrešan. Također, pretpostavimo postojanje samo jedne pogreške. Množenjem primljene poruke matricom $\mathbf{H}^T$, a to je matrica pariteta koda kojemu pripada primljena kodna riječ, dobijemo sindrom pogreške u petome retku matrice $\mathbf{H}^T$ pa je isprav(lje)na kodna riječ, [0011010]. Sljedeća tablica 3.1 prikazuje kodnu riječ [0011010] za sve inačice jednostrukih pogrešaka od 1. do 7. mjesta.

*Tablica 3.1: Kodna riječ* [0011010] *i sve inačice jednostrukih pogrešaka*

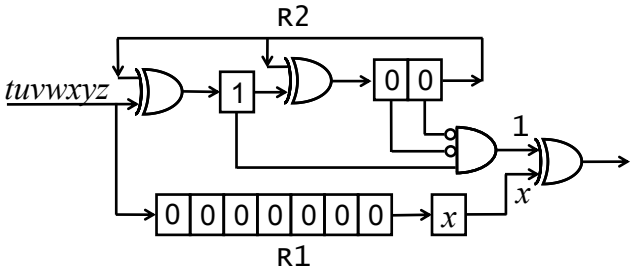
| Polinom-generator je $1 + x + x^3$ | | | |
|---|---|---|---|
| Pogreška na mjestu u kodnoj riječi (čitano s lijeva u desno) | Kodna riječ [0011010] | Sadržaj R2 nakon 7. (15.) posmika | Redak u $\mathbf{H}^T$, (oznake redaka od 1. do 7.) |
| 1. | 1011010 | 100 | 1. |
| 2. | 0111010 | 010 | 2. |
| 3. | 0001010 | 001 | 3. |
| 4. | 0010010 | 110 | 4. |
| 5. | 0011110 | 011 | 5. |
| 6. | 0011000 | 111 | 6. |
| 7. | 0011011 | 101 | 7. |

Budući smo pokazali da je posmik primljene poruke u R2 jednak množenju poruke matricom $\mathbf{H}^T$, konačan sadržaj R2 jednak je [011]. Dalje nastavimo posmik cjelokupnoga kruga na slici 3.25, nakon što R1 već sadrži primljenu poruku, a R2 sadrži [011]. Pošto je sada ulaz u R2 niz uzastopnih 0, on radi kao sklop na slici 3.4 (tj., uzastopno stvara retke matrice $\mathbf{H}^T$, počevši s [011]). U tome trenutku,

[55] Napraviti kao vježbu!

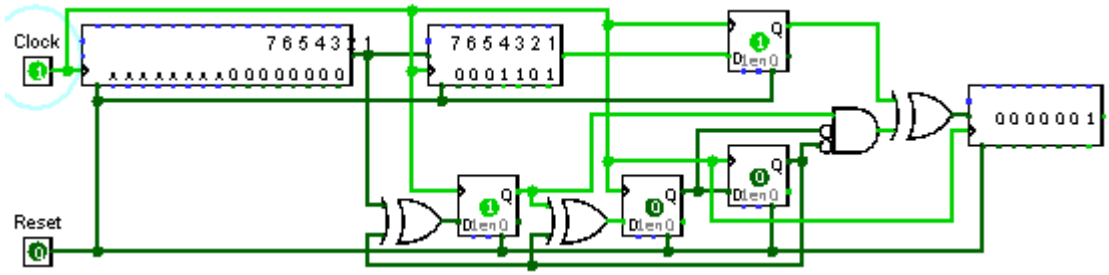
primljena poruka pomakla se iz R1. Zapišimo uzastopne sadržaje R1 i R2, počevši stanjem koje je primljena poruku imala u R1 (slika 3.26):

| R1 | R2 |
|---|---|
| *t u v w x y z* | 011 |
| *t u v w x y* | 111 |
| *t u v w x* | 101 |
| *t u v w* | 100 |




*Slika 3.26: Trenutno stanje*

Točno u trenutku kada je sadržaj R2 jednak [100], pogrešan bit *x* napušta R1 i ulazi u njegovo izolirano stanje, desno. Ovo je jedino stanje R2, nakon 10. posmika, tijekom kojega je izlaz I (AND) vrata jednak 1. Postoje samo jedna takva vrata na slici 3.26, jer su prva 2 ulaza negirane 0, a na treći, donji ulaz, preslikava se jedinica prvoga stanja registra R2. Ova 1 onda se zbraja (XOR) s *x* na svojemu putu do izlaza i preokreće njegovu vrijednost (izlaz je 0). Budući je *x* pogrešan bit, njegovim preokretanjem, ispravlja se primljena poruka. Na putu prema izlazu, ostali bitovi primljene poruke zbrajaju se (XOR) nulama pa su nepromijenjeni. Slika 3.27 prikazuje cjelovit Meggittov dekoder.




*Slika 3.27: Meggittov dekoder - pogrešan bit broj 7, upravo je ušao u izolirano desno stanje*

Pokazali smo kako se automatski ispravlja primljena poruka, koja ima jednu pogrešku na 5. mjestu. Student može lako vidjeti da sklop ispravlja jednu pogrešku koja se nalazi na bilo kojemu drugomu mjestu u poruci. Ako se pogrešan bit izluči iz R1, onda će se on uvijek zbrojiti (XOR) s 1. Svi ostali bitovi primljene poruke ostat će nepromijenjeni.

Valjanost postupka ispravke pogrešaka može se objasniti kako slijedi. Neka poruka *tuvwxyz* sadrži pogrešku na mjestu *i*, računajući s lijeva. Nakon što se poruka posmikne u krug na slici 3.25, registar R2 sadrži *i*-ti redak matrice $\mathbf{H}^T$. Posmik R2 dodatno će generirati retke *i*+1, *i*+2, itd., budući da R2 sada radi kao i generator na slici 3.4. Nakon 8–*i* posmika, sadržaj R2 bit će [100] (redak stvoren nakon 7-oga posmika, opet je prvi redak). Vrata I (AND) sada stvaraju 1.

Stanja registra R1 istodobno se posmiču kao i stanja R2 registra. Nakon 8–*i* posmika, pogrešan bit boravi u izoliranome stanju registra R1 (desno), te se zbraja (XOR) s 1 stvorenom I (AND) vratima i tako se ispravlja.

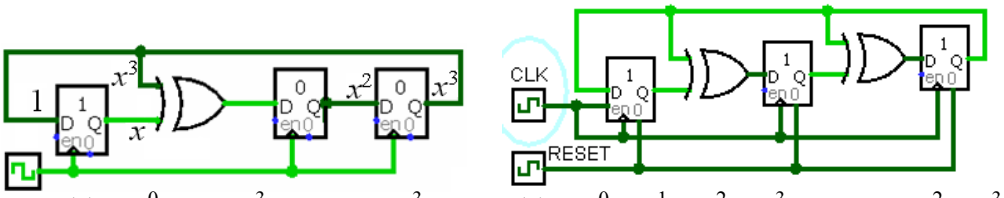
Ako primljena poruka ne sadrži pogreške, poruka koja se posmiče iz kruga na slici 3.25, jednaka je ulaznoj poruci, nakon što prođe opisani postupak. Ako primljena poruka ne sadrži pogreške (tj., jednaka je kodnoj riječi), nakon što se poruka posmikne u njega, jasno se vidi da je sadržaj R2 "sve 0". Tijekom ponavljanja posmika kruga, sadržaj stanja R2 ostaju "sve 0", čime se osigurava da izlaz iz I (AND) vrata nikada neće postati 1, a to osigurava nepromijenjen sadržaj stanja u R1 bez obzira na posmike.

## *3.6. 3.6. Općenit kod ispravke jedne pogreške*

Rasprave u prethodnim poglavljima obrađivale su specifično određen (7, 4) kod. Svi njegovi sklopovi inačice su LFSR na slici 3.4. Ovaj osnovni krug ima svojstvo da, ako se pokrene bilo kojim početnim sadržajem osim "sve 0", krug se vraća u isti početni sadržaj tek nakon sedam posmika. Ovaj krug ima *najveću periodičnost* u smislu da prije povratka u početni položaj, prolazi kroz sva moguća stanja,

osim stanja "sve 0". Budući da matrica pariteta (7, 4) koda ima 7 različitih redaka duljine 3, slijedi da samo *krug maksimalne periodičnosti* može generirati sve ove retke.

Poopćimo sada prethodnu ideju za bilo koji Hammingov ($2^n$–1, $2^n$–$n$–1) kod. Matrica pariteta takvoga koda ima $2^n$–1 redaka duljine $n$. Ako želimo generirati sve retke ove matrice pomoću LFSR s $n$ stanja (krug slici 3.28 je LFSR s 3 stanja), prema definiciji, ovaj krug mora imati maksimalnu periodičnost. Imajte na umu da sve moguće povratne veze NE omogućuju maksimalnu periodičnost.




$R(x) = x^0 + x + x^3 = 1 + x + x^3$ (a) LFSR generira 7 stanja

$R(x) = x^0 + x^1 + x^2 + x^3 = 1 + x + x^2 + x^3$ (b) LFSR generira 4 (odnosno 2) stanja

*Slika 3.28 :Dva posmična registra linearne povratne veze različite periodičnosti*

Na primjer krug na slici 3.28.a ima *najveću periodičnost*, poput kruga slici 3.4. Ali krug na slici 3.28.b ju nema, što se može provjeriti tako da se napravi posmik iz bilo kojega početnoga stanja. Uvijek se vraćamo u to stanje nakon *manje* od sedam posmika (4 odnosno 2 posmika ovisno o početnome stanju).[56]

Ovaj sklop na slici 3.28(b) NE može se stoga upotrijebiti kao generator redaka matrice pariteta za (7, 4) kod, jer takva matrica ima 7 *različitih* redaka što se ne može postići ovim krugom.

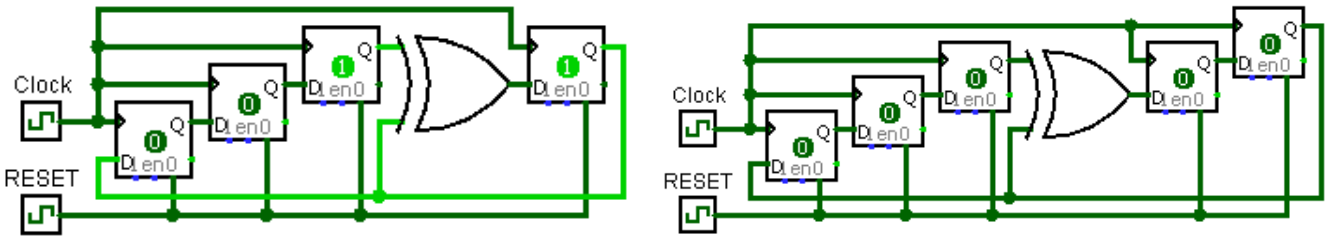
*Definicija* **Periodičnost LFSR što se odnosi na stanje** *S*, minimalan je broj posmika koliko ih LFSR treba napraviti, počevši od stanja *S*, prije nego što njegov sadržaj opet bude jednak *S*.

Oznaka *Pu* označava periodičnost jednoga LFSR s pridruženim stanjem [1000 … 0]. Maksimalna periodičnost LFSR, duljine $n$, ima periodičnost $2^n$–1 u odnosu na bilo koje ne-nulto stanje, što znači da je za takve LFSR $Pu = 2^n$–1.

Općenito, LFSR može imati različite periodičnosti u odnosu na različita stanja. Uzmite na primjer registar na slici 3.28.b. Ako je posmik, započeo početnim stanjem [100], njegovi uzastopni sadržaji bit će [100], [010], [001], [111], [100], itd. Onda za ovaj LFSR vrijedi da je $Pu = 4$. Ako posmičemo ovaj registar, počevši od početnoga stanja [110], njegovi uzastopni sadržaji su [110], [011], [110], itd. pa mu je periodičnost jednaka 2. Isto tako, *svaki* LFSR imat će periodičnost 1 u odnosu na stanje [000], ako nakon samo jednoga posmika, dođe natrag u stanje [000].

Pošto LFSR registar s $n$ stanja *mora imati maksimalnu periodičnost* kako bi generirao retke paritetne matrice ($2^n$–1, $2^n$–$n$–1) Hammingova koda, a budući da sve povratne veze NE jamče maksimalnu periodičnost, vrijedi se zapitati: "Kako možemo znati koje povratne veze omogućuju maksimalnu periodičnost, a koje ne (i to bez isprobavanja raznih mogućnosti)?" Srećom, na to pitanje već se odgovorilo i svaki temeljni udžbenik[57] o kodovima za ispravak pogrešaka daje nam popis raspoloživih povratnih veza što omogućuju maksimalnu periodičnost za razne vrijednosti $n$.

Slika 3.29 daje dva sklopa koji imaju maksimalnu periodičnost (za $n = 4$, $n = 5$).



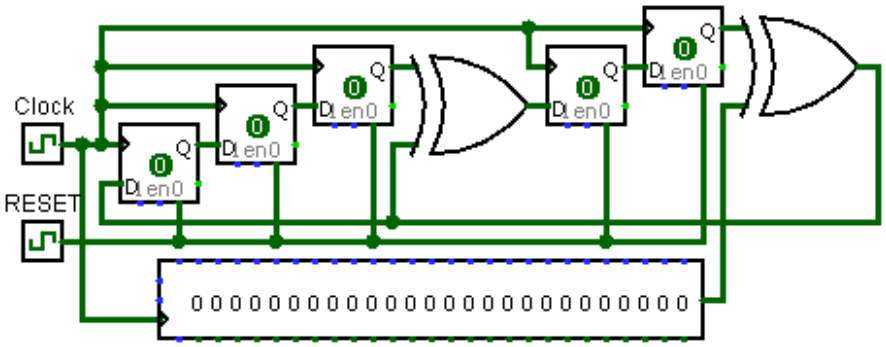

(a) Početno stanje 1000 - 16 posmika (b) Početno stanje 10000 - 32 posmika

*Slika 3.29: Dva* LFSR *maksimalne periodičnosti*

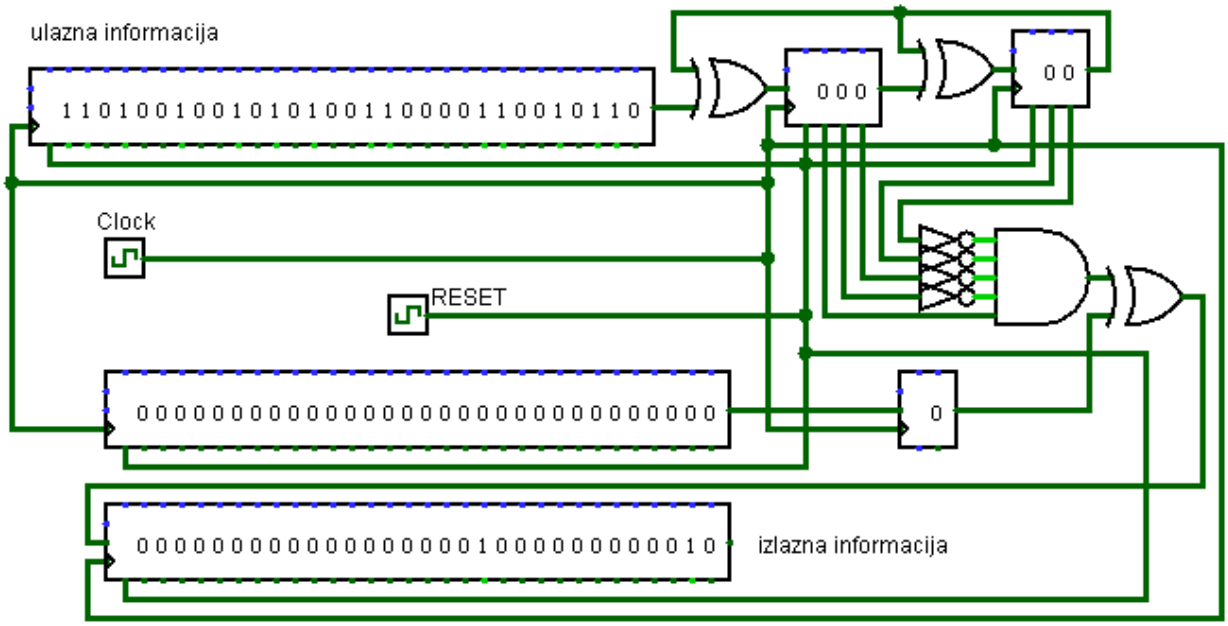
[56] Napraviti 2 vježbe!
[57] lfsr_table.pdf

Postoji nekoliko različitih LFSR iste duljine koji imaju maksimalnu periodičnost. Na primjer, za $n = 3$ krugovi na slici 3.4 i na slici 3.28a imaju maksimalnu periodičnost. Svaki od tih krugova može se koristiti za stvaranje redaka matrice pariteta Hammingova koda. Nakon odabira generatora koji stvara retke paritetne matrice, cjeloviti: koder i dekoder, izrađuju se na isti način kao i krugovi na slikama 3.12 i 3.25, a temelj izradbe je onoj na slici 3.4.

Slika 3.30 prikazuje koder i Meggittov dekoder (31, 26) koda, na temelju kruga na slici 3.29.b.




(a) *kodiranje* - $R(x) = 1 \oplus x^3 \oplus x^5$




(b) *dekodiranje* (~~Meggitt~~ dekoder) $R(x) = 1 \oplus x^3 \oplus x^5$

*Slika 3.30: Primjena (31, 26) Hammingova koda*

Važno je napomenuti da *bilo* koji LFSR može oblikovati koder i dekoder linearnoga koda. Sposobnost otkrivanja/ispravke pogreške koda onda će se razmotriti kao problem. Na primjer, LFSR na slici 3.28.b (koji nema maksimalnu periodičnost) može se spojiti na način sličan onome prikazanome na slici 3.12, dodavanjem paritetnih bitova informacijskome vektoru. Ako izgrađena kodna riječ sada ulazi u isti LFSR, na način sličan onome koji je prikazan na slici 3.7, konačan sadržaj registra bit će "sve 0". Dokaz valjanosti ove tvrdnje slijedi iste crte kao dokaz povezanosti kodera na slici 3.12 i dekodera na slici 3.7. Veza između matrice pariteta **H** i generator-matrice **G** sustavnoga koda vrijedi i ovdje, a ona je:

$$\mathbf{G} = [\mathbf{P} \mid \mathbf{I}_k] \Rightarrow \mathbf{H} = [\mathbf{I}_{n-k} \mid \mathbf{P}^T].$$

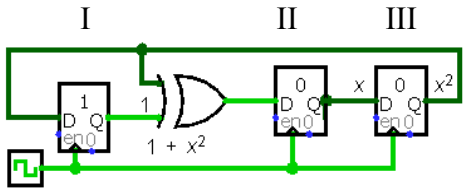
Pitanje obrađeno u poglavlju 3.4 bavi se prečacima u izračunu konačnoga sadržaja LFSR, nakon posmika dugačkoga vektora **t** u njega. Tamo smo na slici 3.23 posebno obradili krug sa slike 3.7. Krugom se prikazalo da "izrezivanje" **t** na pod-vektore duljine 7 i zbrajanjem "odsječaka", dobijemo vektor duljine 7 pa je njegov posmik u registar jednak posmiku cijeloga vektora **t**. U općemu slučaju, ako je LFSR duljine $n$, a njegove povratne veze ne moraju nužno odgovarati maksimalnoj

periodičnosti, ideja je slična. U registar ne moramo posmaknuti vektor koji je dulji od *Pu*. Duže ulazne vektore možemo podijeliti na dijelove duljine *Pu*, a zatim ih obraditi na prethodno opisan način.

### 3.6.1. 3.6.1. ZADATAK ZA (6, 3) KOD

Za zadanu matricu pariteta $\mathbf{H}^T$ odredite matrice **G**, **H**, koder i dekoder.

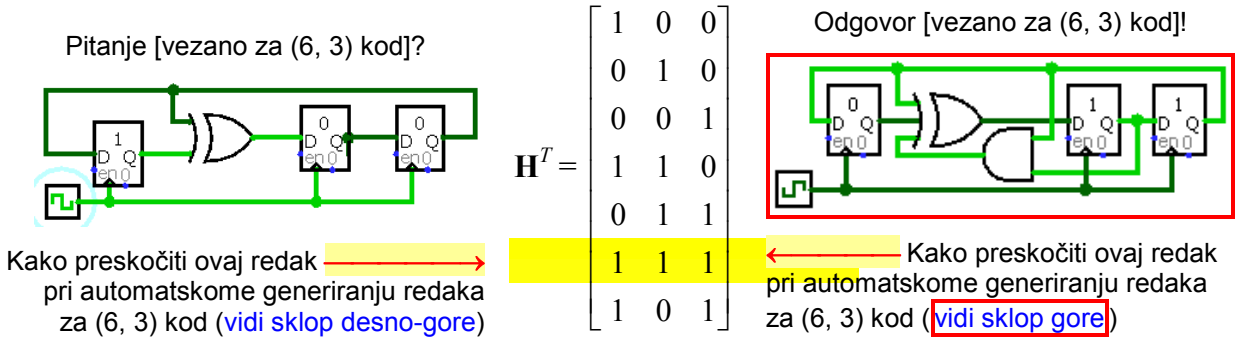
Polazište je:




| posmik | stanja registra R1 | I | II | III |
|---|---|---|---|---|
| | početno stanje | 1 | 0 | 0 |
| | 1. | 0 | 1 | 0 |
| | 2. | 0 | 0 | 1 |
| | 3. | 1 | 1 | 0 |
| | 4. | 0 | 1 | 1 |
| | ~~5.~~ | ~~1~~ | ~~1~~ | ~~1~~ |
| | 6. | 1 | 0 | 1 |

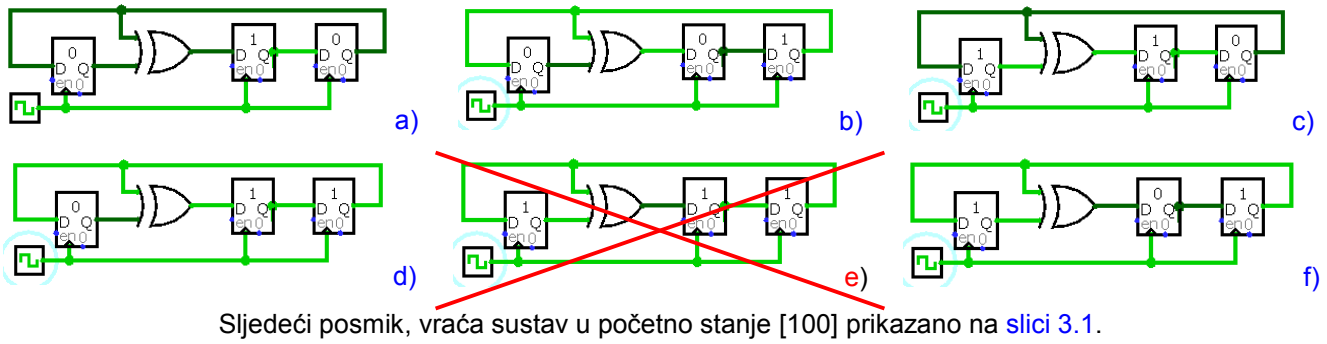
*Slika 3.31: Koder (6, 3) koda*

Rezultat je matrica $\mathbf{H}^T$ prikazana dolje:




*Slika 3.32: Sklop za automatsko "preskakanje" 6. retka u matrici pariteta*

To se postiže sklopom na slici 3.4 u 7 koraka (slika 3.33) uključujući kao prvi korak (slika 3.32), plus koraci a-f (bez e):




*Slika 3.33: "Preskakanje" 6. retka u matrici pariteta*

# 8. *Laboratorijska vježba 7: Hammingovi kodovi: (7, 4)-2. dio te Meggittov dekoder za Hammingove kodove (15, 11) i (31, 26)*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

# 4. POGLAVLJE 4 - CIKLIČKI KODOVI

## 4.1. 4.1. Neka svojstva matrica koje generira LFSR

*Definicija* **Ciklički pomak** (*rotation*) **vektora** u desno za $k$ mjesta znači pomak $k$ elemenata s desnoga kraja vektora i njihovo pridruživanje lijevoj strani.

*Primjer* Ciklički pomak vektora [1011110] u desno za četiri mjesta, daje vektor [1110101]. Općenito možemo definirati i lijevi ciklički pomak istoga vektora duljine $n = 7$ u desno za $i$ mjesta, a on je jednak cikličkome pomaku u lijevo za $n–i = 3$ mjesta (vidi donji program).

*10. Ciklički pomak bitova desno (ili lijevo) svejedno*

```
#include<iostream>
#include<bitset>
using namespace std;
template<size_t N>
//desni ciklički pomak: b=b>>4|b<<(N-m)
inline void rotate(bitset<N>&b,unsigned
m){b=b>>4|b<<(N-m);}
//lijevi ciklički pomak: b=b<<4|b>>(N-m)
void main(){
bitset<7>b(94);
cout<<b<<endl;
rotate(b,4);
cout<<b<<endl;
}
```

```
C:\windows\system32\cmd.exe
1011110
1110101
Press any key to continue . . .
```

Neka je $\mathbf{M}^T$ matrica čiji su retci nastali uzastopnim posmikom LFSR duljine $q$, počevši od stanja [1000 … 0]. Uvijek pretpostavljamo da broj redaka u $\mathbf{M}^T$ premašuje $q$ i ograničen je s $Pu$ (kao što se definiralo u prethodnome poglavlju), $Pu$ označava periodičnost LFSR, povezanu određenim stanjem. Neke osnovne značajke matrice $\mathbf{M}^T$ navode se u nastavku:

Prvih $q$ redaka matrice $\mathbf{M}^T$ oblikuju jediničnu matricu. Ova značajka slijedi izravno iz činjenice:

$$\mathbf{u} \cdot \mathbf{P} = [0\ 1\ 1\ 0] \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = [1\ 0\ 0],$$

a) da se prvih $q$ redaka matrice $\mathbf{M}^T$ stvorilo uzastopnim $q$ posmicima sadržaja [1000 … 0].

Odatle zaključujemo:

*Zaključak 4.1* Kod $C$, čija matrica pariteta je $\mathbf{M}^T$, jest *sustavan kod*.;

*Oznaka* $R_i$ označava $i$-ti redak matrice $\mathbf{M}^T$.;

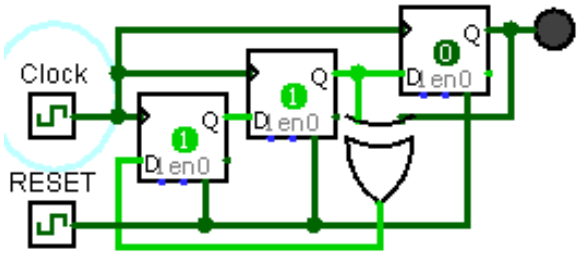
b) 1-elementi u $\mathbf{R}_{(q+i)}$ naznačuju ona stanja LFSR-generatora u koja ulaze izlazi iz zadnjih (desnih) stanja.

*Primjer*

Promatrajmo četvrti redak matrice $\mathbf{H}^T$ koja se obrađivala prije (ovdje je $q = 3$).

$$\mathbf{H}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{matrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \\ R_7 \end{matrix} \Leftarrow \text{četvrti redak}$$

Ovaj redak je [110]. Dvije lijeve 1 pokazuju da se izlaz iz posljednjega (desnoga) stanja LFSR-generatora, povratnom vezom, vraća u prva dva stanja[58], kao što se jasno vidi na slici 4.1.




$$R(x) = 1 \oplus x^2 \oplus x^3$$

*Slika 4.1: LFSR s pražnjenjem (vanjska povratna veza)*

Ispitivanjem ovoga kruga može se također objasniti zašto naša tvrdnja općenito vrijedi. Drugačijim iskazivanjem tvrdnje, možemo napisati:

*Zaključak 4.2* Ako sadržaj LFSR generira vektor **m** oblika [$a$, $b$, $c$, …, $x$, $y$], onda je njegov sadržaj nakon još jednoga posmika [0, $a$, $b$, $c$, …, $x$] + $y \cdot R_{(q+i)}$.

c) Neka je **Q**, matrica dimenzije $q \times q$ i sastoji se od redaka matrice $\mathbf{M}^T$, od retka $R_2$ do retka $R_{(q+1)}$. Onda je $R_{i+1} = R_i \cdot \mathbf{Q}$.

*Primjer* Za matricu $\mathbf{H}^T$, matrica **Q** je:

$$\mathbf{Q} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{matrix} 2.\,redak \\ 3.\,redak \\ 4.\,redak \end{matrix}$$

Uzevši za primjer *peti redak matrice*, $H_5$ = [011] i množeći ga s **Q**, dobivamo šesti redak: [011]·**Q** = [111],

$$[011] \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = [111]$$

a uzimajući za primjer *šesti redak matrice*, $H_6$ = [111] i množeći ga s **Q**, dobivamo sedmi redak: [111]·**Q** = [101].

$$[111] \cdot \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} = [101]$$

[58] Suprotno izvornoj tvrdnji ("output from the last (right) stage of the generating LFSR is fed back into the first two stages"), na slici 4.5 (koja NE PRIPADA OVDJE) vidljivo je da ova tvrdnja nije u potpunosti podudarna!

Valjanost ove značajke slijedi izravno iz *Zaključka 4.2*. To je zapravo ponavljanje iskaza *Zaključka 4.2*, uz napomenu da dva uzastopna sadržaja stvorena u LFSR predstavljaju dva uzastopna retka matrice $\mathbf{M}^T$.

Spomenuto svojstvo može se poopćiti na način koji će nam omogućiti izraziti $\mathbf{R}_{i+2}$ u smislu $\mathbf{R}_i$, budući da je, na temelju ovoga svojstva, $R_{i+2} = R_{i+1}\cdot\mathbf{Q} = [R_i\cdot\mathbf{Q}]\cdot\mathbf{Q} = R_i\cdot\mathbf{Q}^2$. Prema definiciji umnoška dviju matrica, prvi redak $\mathbf{Q}^2$ jednak je $Q_1\cdot\mathbf{Q}$ gdje $Q_1$ označava prvi redak matrice $\mathbf{Q}$. Dok je $Q_1 = R_2$, a zatim primjenom našega svojstva, imamo da $Q_1\cdot\mathbf{Q} = R_3$. Drugim riječima, prvi redak $\mathbf{Q}^2$ je $R_3$. Napominjemo da je sada drugi redak matrice $\mathbf{Q}$ redak $R_3$, a primjenom istoga postupka kao prije, dobije se da je drugi redak $\mathbf{Q}^2$ jednak $R_4$. Tako smo dokazali sljedeći zaključak.

*Zaključak 4.3* Neka je $\mathbf{P}_j$ pod-matrica dimenzije $q \times q$, sastavljena od redaka matrice $\mathbf{M}^T$, od retka $R_j$ do retka $R_{(q+j-1)}$. Zatim je $R_i\cdot\mathbf{P}_j = R_{i+j-1}$.

*Primjer* Za matricu $\mathbf{H}^T$ imamo:

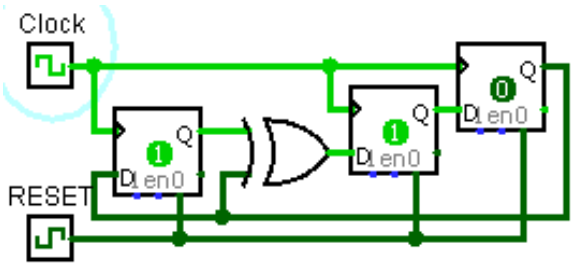
$$\mathbf{P}_6 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{matrix} 6.\,redak \\ 7.\,redak \\ 1.\,redak \end{matrix}$$

Namjerno se uzeo slučaj u kojemu se koristi svojstvo "*ponavljanja*" (*end around*) matrice čiji su retci nastali od LFSR maksimalne periodičnosti, tj., [101], što je posljednji redak matrice $\mathbf{H}^T$. Smatra se da nakon njega slijedi [100], što je prvi redak $\mathbf{H}^T$. Nakon toga smo imali, na primjer, da je $R_4\cdot\mathbf{P}_6 = [110]\cdot\mathbf{P}_6 = [010] = R_2$. Imajte na umu da prema Zaključku 4.3, $R_4\cdot\mathbf{P}_6$ treba biti jednako $R_{(4+6-1)} = R_9$. Međutim, obzirom na svojstvo *ponavljanja*, iza $R_7$ slijedi $R_1$, iz čega proizlazi da je $R_9 \cong R_2$.

d) Neka je **G** generator-matrica sustavnoga koda čija je pod-matrica **P**, matrica pariteta. Ciklički pomak u desno za jedno mjesto bilo kojega retka matrice **G** osim zadnjega, daje kodnu riječ. Kako bi pokazali zašto gornja tvrdnja vrijedi, zbog jednostavnosti prvo promotrimo (7, 4) kod (čija je matrica pariteta $\mathbf{H}^T$). Opet promatramo strukturu generator-matrice **G'** koda.

$$\mathbf{G'} = \left[ \underbrace{\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}}_{\mathbf{P}\ \text{matrica}} \vdots \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{jedinična matrica}} \right]$$

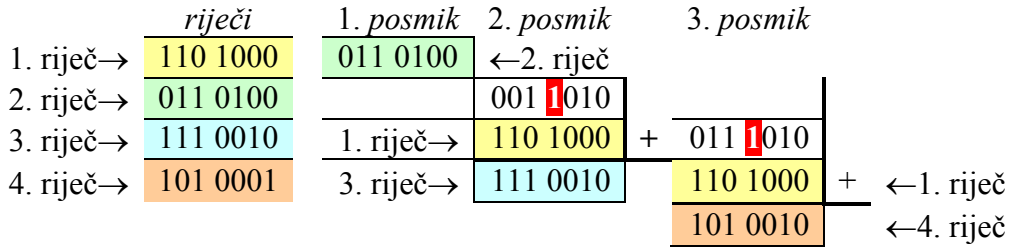
Retci pod-matrice **P**, stvaraju se uzastopnim posmikom LFSR, početnim stanjem [110], a prikazuje ih slika 4.2.




*Slika 4.2: Generator redaka matrice* **P**

Retci nove matrice **G'** kodne su riječi, a odgovaraju *informacijskim* vektorima koji imaju samo jedan element vrijednosti 1. Neka je $C_j$, $j$-ti redak matrice **G'**, za $j$ = 1, 2, 3, 4. Način na koji $C_j$ može generirati sljedeći redak $C_{(j+1)}$, može se okarakterizirati ovako: Za $j$ = 1, 2, 3, $C_j$ posmiče se ciklički za jedno mjesto u desno. Na ovaj način, premješta se 1-element u dijagonali jedinične matrice na svoje pravo mjesto u $C_{(j+1)}$. Ovaj pomak premješta 1. riječ 3 puta, kao sadržaj registra,

zatim ih (prema zakonima linearne algebre) međusobno zbraja s ostalim prikladnim retcima matrice **G**, stvarajući na taj način retke pod-matrice **P**, a time i kodne riječi matrice **G'** (vidi sliku 4.3).




*Slika 4.3: Zbroj prikladnih redaka matrice* **G***, stvara retke pod-matrice* **P**

Ako ovaj posmik ne pomakne 1-element iz **P** dijela od **G'** u desni dio od **G'**, onda se vektor dobije pomakom jednak $C_{(j+1)}$ pa se i dalje ne poduzima nikakva akcija. Ako se 1 pomakne u desni dio, tako da se remeti zahtjev o oblikovanju jedinične matrice u desnome dijelu, *ova 1 automatski se poništava tako da se pomaknutome $C_j$, doda prvi redak* ($C_1$). Ovaj opis vrijedi za 2. [011 0100] i 3. [111 0010] kodnu riječ, a zorno ga prikazuje slika 4.3. Posljednja tvrdnja jasna je promatranjem načina na koji LFSR stvara retke pod-matrice **P**.

Ako ciklički pomak $C_j$ pomiče 1 u desni dio **G'**, što znači da su lijeva tri bita $C_j$ u obliku ($a$, $b$, 1). Oni čine trenutan sadržaj LFSR stvarajući retke pod-matrice **P**. Daljnji posmik premjestit će 1 izvan LFSR, oblikovanjem novoga sadržaja (1, $a$+1, $b$), a 1 što se posmikne vani iz LFSR, odbacuje se. Ukupan učinak jednak je zbroju $C_1$ i cikličkome pomaku $C_j$.

Gornja rasprava može se *poopćiti* na bilo koji sustavan kod čiju matricu pariteta stvara LFSR. Ovo će zahtijevati da se pripadna generator-matrica sastoji od pod-matrice **P** u prije prikazanome obliku i jedinične matrice. Neka je **G** generator-matrica koda čija matrica pariteta je $\mathbf{M}^T$. Svaki redak $C_j \cdot \mathbf{G}$ (osim posljednjega) stvara sljedeći redak cikličkim pomakom $C_j$ u desno za jedno mjesto, a onda se pomaku $C_j$ dodaje prvi redak matrice **G** ili se ništa ne poduzima (slika 4.3).

Valjanost značajke d) sada je očita. Pomak bilo kojega retka matrice **G** ciklički u desno za jedno mjesto, osim njenoga zadnjega retka, uvijek daje kodnu riječ. Ova kodna riječ je ili bilo koji redak matrice **G** ili zbroj drugoga i prvoga retka matrice **G**. Zbog linearnosti koda, taj zbroj je kodna riječ.

## 4.2. 4.2. Linearni posmici kodnih riječi i njihov utjecaj na generator-matricu

*Definicija* **Posmik vektora linearno** u desno za $k$ mjesta, znači prijenos $k$ elemenata desno od vektora uz pridruživanje 0 lijevo od $k$.

*Primjer* Posmik vektora [1011110] linearno u desnu stranu za četiri mjesta, daje vektor [0000101].

*Tvrdnja 4.1* Neka je $C$, ($n$, $k$) kod čija je matrica pariteta **P** (definirana u prethodnome odlomku), a **G** je generator-matrica od $C$.

1. Bilo koja kodna riječ u $C$, sastoji se od zbroja pojedinih linearnih posmika u desno prvoga retka matrice **G**, gdje posmici nisu veći od $k$–1 mjesta.
2. Bilo koji vektor duljine $n$, a koji se sastoji od zbroja nekih linearnih posmika prvoga retka matrice **G** u desno (gdje posmici nisu veći od $k$–1 mjesta), kodna je riječ u $C$.

Prije dokazivanja Tvrdnje 4.1, ovo će se objasnit primjerom sustavnoga (7, 4) koda.

### 4.2.1. 4.2.1. SUSTAVAN KOD

Matrica pariteta $\mathbf{H}^T$ sustavnoga Hammingovoga (7, 4) koda dana je kao:

$$\mathbf{H}^T = \begin{bmatrix} \mathbf{I}_{n-k} \\ \hline \mathbf{P} \end{bmatrix} \Rightarrow \mathbf{G} = [\mathbf{P} \mid \mathbf{I}_k] \Rightarrow \mathbf{H} = [\mathbf{I}_{n-k} \mid \mathbf{P}^T] \qquad (4.1)$$

$$\mathbf{H}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{matrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ R_5 \\ R_6 \\ R_7 \end{matrix} \Rightarrow \mathbf{G} = \begin{matrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{v}_4 \end{matrix} \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \Rightarrow \mathbf{H} = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right]$$

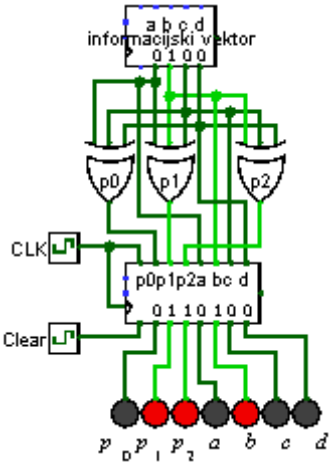
Prvi dio ove Tvrdnje navodi da se bilo koja kodna riječ sastoji od zbroja nekih od sljedećih vektora: $\mathbf{v}_1 = [1101000]$, $\mathbf{v}_2 = [0110100]$, $\mathbf{v}_3 = [0011010]$, $\mathbf{v}_4 = [0001101]$ iz pripadne generator-matrice **G** (vidi gore-desno).

Ovo se može provjeriti analizom kodnih riječi navedenih u prethodnome poglavlju. Drugi dio Tvrdnje iskazuje da bilo koji vektor duljine 7, što se sastoji od zbroja bilo kojih složaja četiriju vektora generator-matrice $\mathbf{G} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4]^T$, kodna je riječ našega koda. Takvih složaja zbrojeva ima: ($\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, $\mathbf{v}_4$, $\mathbf{v}_1 \oplus \mathbf{v}_2$, $\mathbf{v}_1 \oplus \mathbf{v}_3$, $\mathbf{v}_1 \oplus \mathbf{v}_4$, $\mathbf{v}_2 \oplus \mathbf{v}_3$, $\mathbf{v}_2 \oplus \mathbf{v}_4$, $\mathbf{v}_3 \oplus \mathbf{v}_4$, $\mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \mathbf{v}_3$, $\mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \mathbf{v}_4$, $\mathbf{v}_1 \oplus \mathbf{v}_3 \oplus \mathbf{v}_4$, $\mathbf{v}_2 \oplus \mathbf{v}_3 \oplus \mathbf{v}_4$, $\mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \mathbf{v}_3 \oplus \mathbf{v}_4$):

$$K_n^r = \binom{n}{r} = \frac{n!}{r!(n-r)!} = \binom{n}{n-r} = \binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 4 + 6 + 4 + 1 = 15.$$

Kodna riječ "sve 0" dobije se kao zbroj 7 kodnih riječi dobivenih iz informacijskih vektora koji započinju brojem "0". Kodna riječ "sve 1" dobije se kao zbroj 7 kodnih riječi dobivenih iz informacijskih vektora koji započinju brojem "1". Te zbrojeve prikazuje donja tablica.

| | | y | | | y |
|---|---|---|---|---|---|
| 1 | $\mathbf{v}_4$ | 1010001 | 9 | $\mathbf{v}_1$ | 1101000 |
| 2 | $\mathbf{v}_3$ | 1110010 | 10 | | 0111001 |
| 3 | | 0100011 | 11 | | 0011010 |
| 4 | $\mathbf{v}_2$ | 0110100 | 12 | | 1001011 |
| 5 | | 1100101 | 13 | | 1011100 |
| 6 | | 1000110 | 14 | | 0001101 |
| 7 | | 0010111 | 15 | | 0101110 |
| 8 | $\oplus$ | 0000000 | 16 | $\oplus$ | 1111111 |




$$\mathbf{G} = \begin{matrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{v}_4 \end{matrix} \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

Isti rezultati dobiju se matematički navedenim zbrojevima složaja četiriju vektora generator-matrice **G** (vidi donju tablicu). Osim toga, vektor "sve 1" dobije se također i kao zbroj svih četiriju vektora generator-matrice **G** ($\mathbf{v}_1 \oplus \mathbf{v}_2 \oplus \mathbf{v}_3 \oplus \mathbf{v}_4$).

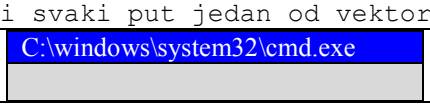
| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathbf{v}_1$ | 1101000 | $\mathbf{v}_1$ | 1101000 | $\mathbf{v}_1$ | 1101000 | $\mathbf{v}_1$ | 1101000 | $\mathbf{v}_1$ | 1101000 | $\mathbf{v}_1$ | 1101000 | $\mathbf{v}_1$ | 1101000 |
| $\mathbf{v}_2$ | 0110100 | $\mathbf{v}_3$ | 1110010 | $\mathbf{v}_4$ | 1010001 | $\mathbf{v}_2$ | 0110100 | $\mathbf{v}_2$ | 0110100 | $\mathbf{v}_3$ | 1110010 | $\mathbf{v}_2$ | 0110100 |
| | | | | | | $\mathbf{v}_3$ | 1110010 | $\mathbf{v}_4$ | 1010001 | $\mathbf{v}_4$ | 1010001 | $\mathbf{v}_3$ | 1110010 |
| | | | | | | | | | | | | $\mathbf{v}_4$ | 1010001 |
| $\oplus$ | 1011100 | $\oplus$ | 0011010 | $\oplus$ | 0111001 | $\oplus$ | 0101110 | $\oplus$ | 0001101 | $\oplus$ | 1001011 | $\oplus$ | 1111111 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $\mathbf{v}_2$ | 0110100 | $\mathbf{v}_2$ | 0110100 | $\mathbf{v}_3$ | 1110010 | $\mathbf{v}_2$ | 0110100 |
| $\mathbf{v}_3$ | 1110010 | $\mathbf{v}_4$ | 1010001 | $\mathbf{v}_4$ | 1010001 | $\mathbf{v}_3$ | 1110010 |
| | | | | | | $\mathbf{v}_4$ | 1010001 |
| $\oplus$ | 1000110 | $\oplus$ | 1100101 | $\oplus$ | 0100011 | $\oplus$ | 0010111 |

Svojstva kodnih riječi nesustavnoga Hammingovoga (7, 4) koda za istu paritetnu matricu, analiziraju se u Laboratorijskoj vježbi 8.

### 11. *Međusobni zbrojevi dvaju od četiriju vektora generator-matrice G daju jedan od postojećih vektora*

```
//#include <conio.h> // getch();
Zadana su 4 vektora generator-matrice G. Međusobno ih zbrojiti prema gornjoj
formuli i provjeriti daju li svaki put jedan od vektora koji se zbrajaju.
```




*Dokaz Tvrdnje 4.1* Dokaz je prilično dug. Odlučili smo ga čitavoga uključiti, jer predstavlja neka važna opća razmatranja. Da bi dokaz bio razumljiviji, podijelimo ga u nekoliko koraka.

**1. korak:** Suodnos cikličkih pomaka i linearnih posmika redaka matrice **G**.

Kod *C* je sustavan (Zaključak 4.1). Onda generator-matrica **G** ima jediničnu matricu na svojoj *desnoj strani* (kao što se prikazalo u gornjemu primjeru), što znači da **svi** retci **G**, osim posljednjega, imaju 0 kao svoj desni element. Samo je u desnome stupcu, u zadnjemu retku jedinične matrice, 1-element). Iz toga slijedi da *ciklički* pomak u desno za jedno mjesto bilo kojega retka u **G** osim posljednjega, daje isti rezultat kao *linearan* posmik za jedno mjesto u desno. U cikličkome pomaku 0 u desno, ona se pomiče na lijevu stranu. U linearnome posmiku ova 0 odbacuje se, a nova 0 stvara se na lijevoj strani, što daje isti rezultat.

**2. korak:** Izražavanje redaka **G** u smislu linearnih posmika prvoga retka.

U raspravi koja potvrđuje svojstvo d) navedeno je da se svaki redak **G**, osim prvoga, generira *cikličkim* pomakom prethodnoga retka za jedno mjesto u desno. U slučaju ako se 1 posmiče iz **P** dijela matrice u dio jedinične matrice **G**, (ponekada) se zbrajaju prvi redak i vektor dobiven tim posmikom (vidi sliku 4.3). Iz rezultata navedenoga u 1. koraku slijedi da ovdje možemo riječ "ciklički" zamijeniti riječju "linearno".

Budući da se svaki redak, od drugoga pa nadalje, stvara posmikom prethodnoga retka linearno za jedno mjesto u desno, a ponekad i zbrojem prvoga retka i ovoga rezultata, slijedi da se *bilo koji* redak iz **G**, sastoji od zbroja nekih linearnih posmika prvoga retka u desno, gdje posmici nisu veći od $k$–1 mjesta. Ovo svojstvo svakako vrijedi i za prvi redak. Ono također vrijedi i za drugi redak, obzirom na upravo navedeno svojstvo. Treći redak dobije se posmikom drugoga retka, a zatim možda i zbrojem vektora dobivenoga posmikom prvoga retka. Ovaj redak može se izraziti u smislu zbroja posmika prvoga retka. Isto sada vrijedi i za sljedeći redak, itd.

**3: korak** Razmatranje opće kodne riječi.

Bilo koja kodna riječ nekoga koda sastoji se od zbroja pojedinih redaka **G**. Budući da se svaki takav redak sastoji od zbroja pojedinih linearnih posmika prvoga retka u desno, gdje posmici nisu veći od $k$–1 mjesta, slijedi da se bilo koja kodna riječ također sastoji od zbroja pojedinih linearnih posmika prvoga retka u desno, gdje posmici nisu veći od $k$–1 mjesta. Time je dovršen dokaz 1. dijela Tvrdnje 4.1.

**4: korak** Dokaz 2. dijela Tvrdnje 4.1.

Drugi dio Tvrdnje 4.1 navodi da je bilo koji vektor duljine $n$ kodna riječ u $C$, ako se sastoji od zbroja pojedinih linearnih posmika u desno prvoga retka od **G** (za ne više od $k$–1 mjesta).

Neki ($n$, $k$) kod ima $2^k$ različitih kodnih riječi. Ovo je broj mogućih informacijskih vektora, a svaki je duljine $k$. Budući da svaki informacijski vektor oblikuje različite kodne riječ, to je ujedno i broj kodnih riječi. Budući da se dokazao prvi dio Tvrdnje 4.1, bilo koja od ovih $2^k$ kodnih riječi ima svojstvo da se sastoji od zbroja nekih linearnih posmika prvoga retka generator-matrice **G** u desno, gdje posmici nisu veći od $k$–1 mjesta. Pitanje je, postoje li (ili ne postoje) drugi vektori duljine $n$ koji imaju isto svojstvo, a nisu kodne riječi u $C$.

Odgovor je *ne*, jer nema više od $2^k$ vektora koji zadovoljavaju ovo svojstvo. To je zato, jer je broj mogućih linearnih posmika vektora, gdje posmici nisu veći od $k$–1 mjesta, upravo $k$. Postoji, dakle, ne više od $2^k$ vektora duljine $n$ koji se mogu izraziti kao zbroj nekih od tih posmika. Postoje $2^k$ različita

načina zbrajanja $k$ vektora, izvan skupa od $k$ vektora. Svi ovi vektori, dakle, kodne su riječi u $C$. Time je dovršen dokaz Tvrdnje 4.1. Iz Tvrdnje 4.1 formulirali smo sljedeću Tvrdnju:

*Tvrdnja 4.2* Neka je $C$ ($n$, $k$) linearan sustavan kod. Neka $C\mathbf{i}$ označava kodnu riječ kojoj odgovara informacijski vektor [1000 … 0]. Kod $C$ može se poopćiti pomoću LFSR, ako su svi $k$–1 linearni desni pomaci $C\mathbf{i}$, također kodne riječi koda. LFSR koji generira kod definira se kao krug na kojemu se temelji kodiranje i dekodiranje u obliku prikazanome na slikama 3.2 i 3.3.

Uz pretpostavku da u našemu sustavnomu kodu, informacijski vektor oblikuje desni dio odgovarajuće kodne riječi, valjanost Tvrdnje 4.2 ne objašnjava ništa osim da je $C\mathbf{i}$ kodna riječ koja ima $k$–1 nula na svojoj desnoj strani, jer odgovara informacijskomu vektoru [1000 … 0]. Prema definiciji, onda $C\mathbf{i}$ oblikuje prvi redak matrice koja stvara kod.

Ostatak dokaza Tvrdnje 4.2 ostaje kao vježba pa je kodu $C$ svojstvena činjenica da njegovu paritetnu matricu $\mathbf{M}^T$ generira LFSR.

## 4.3. 4.3. Svojstva kodnih riječi maksimalne duljine

### 4.3.1. 4.3.1. SVOJSTVA PARITETNE MATRICE I GENERATOR-MATRICE KODNIH RIJEČI MAKSIMALNE DULJINE

*Oznake*: {**H**} označava skup svih matrica čiji su retci nastali posmikom bitova u LFSR, gdje je početno stanje LFSR jednako [1000 … 0] i zaustavlja se korak prije nego što se vrati u to početno stanje Dakle, broj redaka matrice koja pripada skupu matrica {**H**} je *Pu*. Ako zadnji redak bilo koje matrice u skupu {**H**} oblikuje njezin sadržaj pomoću LFSR, a LFSR se posmiče još jednom, njegov sadržaj bit će opet [1000 … 0].

{$C$} označava skup svih kodova koji imaju matricu pariteta u skupu {**H**}. Duljina kodne riječi u kodu što pripada {$C$}, onda je vrijednost *Pu* odgovarajućega LFSR. Takve kodne riječi definirane su kao **kodne riječi najveće duljine**.

Dakako, značajke od a) do d) navedene u poglavlju 4.1 također vrijede za bilo koju matricu koja pripada skupu {**H**}. Zaključak 4.1 znači da je sustavan svaki kod koji pripada skupu {$C$}. Sljedeće tvrdnje odnose se samo na matrice koje pripadaju skupu {**H**}:

e) Neka $R_n$ označava *zadnji* redak matrice $\mathbf{M}^T$ koja pripada skupu {**H**}. Ciklički pomak $R_n$ za jedno mjesto u desno daje $R_{(q+i)}$. To se vidi ponovnim promatranjem matrice $\mathbf{H}^T$ (koja pripada skupu {**H**}). Njezin $R_n$ je [101]. Ciklički pomak za jedno mjesto u desno daje [110], a to je njezin redak $R_{(q+1)}$. Dokazujemo da ova tvrdnja vrijedi za bilo koju matricu $\mathbf{M}^T$ u {**H**}. Imajte na umu da ako $R_n$ predstavlja sadržaj LFSR-generatora, onda je nakon još jednoga posmika, njegov sadržaj jednak [1000 … 0]. Ovo se temelji na definiciji {**H**}.

   Jedinica lijevo od ovoga novoga sadržaja može biti na prvome mjestu - lijevo, samo ako je prethodan sadržaj $R_n$ imao 1 desno (ova 1 zatim se prebacuje lijevo) pa je onda $R_n$ struktura [$a$, $b$, $c$, …, $x$, 1]. Na temelju zaključka 4.2 imamo da je [1000…0] = [0, $a$, $b$, $c$, …, $x$] + $R_{(q+i)}$. Iz toga slijedi da je $R_{(q+i)}$ = [0, $a$, $b$, $c$, …, $x$] ⊕ [1000 … 0] = [1, $a$, $b$, $c$, …, $x$]. Budući se izraz [1, $a$, $b$, $c$, …, $x$] odnosi na ciklički pomak $R_n$, za jedno mjesto u desno, općenito smo dokazali našu tvrdnju.

f) Neka je matrica $\mathbf{H}^T$ (pripada skupu {**H**}), matrica pariteta koda. Ovaj kod pripada {$C$} i sustavan je obzirom na Zaključak 4.1. Neka je **G**, generator-matrica toga koda. Zatim ciklički pomak *zadnjega retka* u **G** za jedno mjesto u desno, daje *prvi redak* u **G**. Prije nego što se dokaže ovo svojstvo, prikažimo ga za kod čija matrica pariteta je $\mathbf{H}^T$. Njegova generator-matrica je:

$$\mathbf{G}' = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{G} = \left[ \underbrace{\begin{bmatrix} & \\ & \\ & \\ & \end{bmatrix}}_{\mathbf{P}\text{ matrica}} \vdots \underbrace{\begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}}_{\text{jedinična matrica}} \right]$$

Pomakom *zadnjega* retka **G'** ciklički za jedno mjesto u desno daje njezin prvi redak. Za razumjeti zašto je to uglavnom točno, opet imajte na umu sljedeću osnovnu strukturu generator-matrice **G**:

Korištenjem prethodno uvedenih oznaka $R_n$ i $R_{(q+1)}$, znamo da se matrica **P** sastoji od redaka paritetne matrica $\mathbf{H}^T$, od $R_{(q+1)}$ do $R_n$. Iz odnosa $R_{(q+1)}$ i $R_n$, dokazana je prethodna tvrdnja pa onda imamo:

$$\mathbf{P} = \begin{bmatrix} 1abc\ldots x \\ \ddots \\ abc\ldots x1 \end{bmatrix} \quad \text{i zbog toga slijedi:} \quad \mathbf{G} = \left[\begin{bmatrix} 1abc\ldots x \\ \ddots \\ abc\ldots x1 \end{bmatrix} \vdots \begin{bmatrix} 100\ldots0 \\ \ddots \\ 000\ldots1 \end{bmatrix}\right]$$

Promatrajući posljednji oblik **G**, sada je jasno zašto ciklički pomak njegovoga posljednjega retka za jedno mjesto u desno, daje njegov prvi redak.

### 4.3.2. 4.3.2. CIKLIČKI POMACI KODNIH RIJEČI MAKSIMALNE DULJINE

*Definicija* Kod je ***ciklički***, ako su vektori, dobiveni bilo kakvim cikličkim pomakom bilo koje od njegovih kodnih riječi, također kodne riječi. Hammingov (7, 4) kod, čija **H** matrica je

$$\mathbf{H} = \left[\begin{array}{ccc:cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right]$$

je *ciklički*, kao što se jasno vidi iz slijedećega popisa njegovih kodnih riječi (radi se o dvije kodne riječi: [1101000] i [1110010], koje cikličkim pomakom stvaraju sve ostale kodne riječi (16 kodnih riječi). Njihov popis i sastav prikazuje sljedeća tablica (poredana na 2 različita načina):

| | | |
|---|---|---|
| [0000000] | | |
| [0100011] | [1010001] | [1110010] |
| [1000110] | [0110100] | [1100101] |
| [0111001] | [0010111] | [1101000] |
| [1011100] | [0011010] | [1001011] |
| [1111111] | [0001101] | [0101110] |

=

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 00 | 0 | 0 | 000 | 01 | 1 | 1 | 001 |
| 10 | 1 | 0 | 001 | 00 | 1 | 1 | 010 |
| 11 | 1 | 0 | 010 | 11 | 0 | 1 | 000 |
| 01 | 0 | 0 | 011 | 10 | 0 | 1 | 011 |
| 01 | 1 | 0 | 100 | 10 | 1 | 1 | 100 |
| 11 | 0 | 0 | 101 | 00 | 0 | 1 | 101 |
| 10 | 0 | 0 | 110 | 01 | 0 | 1 | 110 |
| 00 | 1 | 0 | 111 | 11 | 1 | 1 | 111 |

Cikličkim pomacima dvaju uzoraka: [1101000] i [1110010] tvore se kodne riječi (ne vrijedi za NEsustavan kod).

| 0000000 | 0010111 | | $p_0$ | $p_1$ | $a$ | $p_2$ | $b$ | $c$ | $d$ | | $p_0$ | $p_1$ | $a$ | $p_2$ | $b$ | $c$ | $d$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0100011 | 0011010 | uzorak=1101000 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | uzorak=1110010 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1000110 | 0001101 | | 0 | 1 | 1 | 0 | 1 | 0 | 0 | | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0111001 | 1110010 | | 0 | 0 | 1 | 1 | 0 | 1 | 0 | | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1011100 | 1100101 | | 0 | 0 | 0 | 1 | 1 | 0 | 1 | | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1111111 | 1101000 | | 1 | 0 | 0 | 0 | 1 | 1 | 0 | | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1010001 | 1001011 | | 0 | 1 | 0 | 0 | 0 | 1 | 1 | | 1 | 0 | 0 | 1 | 0 | 1 | 1 |
| 0110100 | 0101110 | | 1 | 0 | 1 | 0 | 0 | 0 | 1 | | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| | | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Očito je da uzorci: [1101000] i [1110010], cikličkim pomacima tvore cjelovit kod (za sustavan kod).

| uzorak=1101000 | | | | | | | | | uzorak=1110010 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1101000 | | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1110010 |
| | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0110100 | | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0111001 |
| | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0011010 | | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1011100 |
| | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0001101 | | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0101110 |
| | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1000110 | | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0010111 |
| | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0100011 | | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1001011 |
| | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1010001 | | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1100101 |
| ⊕ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1111111 | ⊕ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0000000 |

Uočite da se kodne riječi [0000000] i [1111111] dobiju zbrajajući sve stupce kodnih riječi dobivenih odgovarajućim cikličkim pomacima, uzoraka [1101000] odnosno [1110010].

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 00 | 0 | 0 | 000 | 11 | 0 | 1 | 000 |
| 10 | 1 | 0 | 001 | 00 | 1 | 1 | 010 |
| 11 | 1 | 0 | 010 | 01 | 1 | 1 | 001 |
| 01 | 0 | 0 | 011 | 10 | 0 | 1 | 011 |
| 01 | 1 | 0 | 100 | 10 | 1 | 1 | 100 |
| 11 | 0 | 0 | 101 | 00 | 0 | 1 | 101 |
| 10 | 0 | 0 | 110 | 01 | 0 | 1 | 110 |
| 00 | 1 | 0 | 111 | 11 | 1 | 1 | 111 |

| | | | |
|---|---|---|---|
| 00 | 0 | 0 | 000 |
| 10 | 1 | 0 | 001 |
| 11 | 1 | 0 | 010 |
| 01 | 0 | 0 | 011 |
| 01 | 1 | 0 | 100 |
| 11 | 0 | 0 | 101 |
| 10 | 0 | 0 | 110 |
| 00 | 1 | 0 | 111 |
| 11 | 0 | 1 | 000 |
| 01 | 1 | 1 | 001 |
| 00 | 1 | 1 | 010 |
| 10 | 0 | 1 | 011 |
| 10 | 1 | 1 | 100 |
| 00 | 0 | 1 | 101 |
| 01 | 0 | 1 | 110 |
| 11 | 1 | 1 | 111 |

Napomena:

Hammingovome kodu svojstvena je relacija: $(n, k) = (2^m–1, 2^m–1–m)$. Ako je ona zadovoljena, radi se o Hammingovome kodu. Za provjeru npr. (6, 3) koda imamo:

$$m = n - k = 7 - 4 = 3$$

$$k = 2^m - m - 1 = 2^3 - 3 - 1 = 4 \neq 3$$

$$n = 2^m - 1 = 2^3 - 1 = 7 \neq 6$$

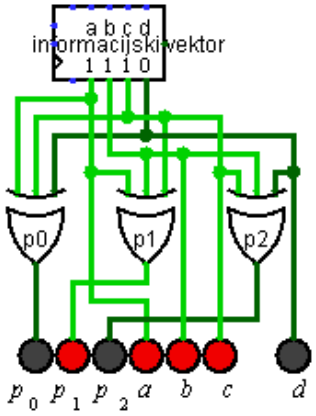
Kod nije Hammingov pa treba odrediti njegove parametre: **G**, **H**, $\mathbf{H}^T$, ….

| | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 0 | 0 | 000 | 01 | 1 | 1 | 001 | 100 | 0 | 110 | 100 | 1 | 011 | 011 | 0 | 100 | 101 | 1 | 100 |
| 10 | 1 | 0 | 001 | 00 | 1 | 1 | 010 | 010 | 0 | 011 | 010 | 1 | 110 | 111 | 0 | 010 | 001 | 1 | 010 |
| 11 | 1 | 0 | 010 | 11 | 0 | 1 | 000 | 001 | 0 | 111 | 001 | 1 | 010 | 101 | 0 | 001 | 011 | 1 | 001 |
| 01 | 0 | 0 | 011 | 10 | 0 | 1 | 011 | 110 | 0 | 101 | 110 | 1 | 000 | 100 | 0 | 110 | 001 | 1 | 010 |
| 01 | 1 | 0 | 100 | 10 | 1 | 1 | 100 | 011 | 0 | 100 | 011 | 1 | 001 | 010 | 0 | 011 | 100 | 1 | 011 |
| 11 | 0 | 0 | 101 | 00 | 0 | 1 | 101 | 111 | 0 | 010 | 111 | 1 | 111 | 001 | 0 | 111 | 111 | 1 | 111 |
| 10 | 0 | 0 | 110 | 01 | 0 | 1 | 110 | 101 | 0 | 001 | 101 | 1 | 100 | 110 | 0 | 101 | 000 | 1 | 101 |
| 00 | 1 | 0 | 111 | 11 | 1 | 1 | 111 | 000 | 0 | 000 | 000 | 1 | 101 | 000 | 0 | 000 | 110 | 1 | 000 |

Koder koji ih generira je:

$$\mathbf{G} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \overset{1}{} \\ \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

$$p_0 = a \oplus c \oplus d,$$
$$p_1 = a \oplus b \oplus c,$$
$$p_2 = b \oplus c \oplus d.$$




| ulaz | izlaz | ulaz | izlaz |
|---|---|---|---|
| 0000 | 0000000 | 1000 | 1101000 |
| 0001 | 1010001 | 1001 | 0111001 |
| 0010 | 1110010 | 1010 | 0011010 |
| 0011 | 0100011 | 1011 | 1001011 |
| 0100 | 0110100 | 1100 | 1011100 |
| 0101 | 1100101 | 1101 | 0001101 |
| 0110 | 1000110 | 1110 | 0101110 |
| 0111 | 0010111 | 1111 | 1111111 |

*Slika 4.4: Stvaranje kodnih riječi iz informacijskih vektora (Tablica 2.2)*[59]

Provjera pripadnosti generiranih kodnih riječi Hammingovome (7, 4) kodu:

| Gornje iste kodne riječi drugačije raspoređene | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *Tablica 2.2* | | | | *Tablica 2.2* | | | | *Tablica 2.1 (7, 4) kod* | | | |
| 0000 | 000 | 1110 | 000 | 000 | 0000 | 000 | 1111 | 0000 | 000 | 1000 | 110 |
| 1101 | 001 | 0011 | 001 | 001 | 0110 | 001 | 1001 | 0001 | 101 | 1001 | 011 |
| 0101 | 010 | 1011 | 010 | 010 | 0101 | 010 | 1010 | 0010 | 111 | 1010 | 001 |
| 1000 | 011 | 0110 | 011 | 011 | 0011 | 011 | 1100 | 0011 | 010 | 1011 | 100 |
| 1001 | 100 | 0111 | 100 | 100 | 0011 | 100 | 1100 | 0100 | 011 | 1100 | 101 |
| 0100 | 101 | 1010 | 101 | 101 | 0101 | 101 | 1010 | 0101 | 110 | 1101 | 000 |
| 1100 | 110 | 0010 | 110 | 110 | 0110 | 110 | 1001 | 0110 | 100 | 1110 | 010 |
| 0001 | 111 | 1111 | 111 | 111 | 0000 | 111 | 1111 | 0111 | 001 | 1111 | 111 |

*Tvrdnja 4.3* **Bilo koji kod što pripada** skupu $\{C\}$ **je ciklički**. Drugim riječima: bilo koji kod što ga stvori LFSR čija je duljina kodne riječi *Pu*, je ciklički.

[59] Napraviti vježbu!

Ova Tvrdnja objašnjava zašto se kodovi, što pripadaju skupu $\{C\}$, zovu ***ciklički kodovi***. Prije dokazivanja teze trebamo malo više razrade. Koncept *cikličkih kodova* obično se odnosi na bilo koji kod što ga generira LFSR, bez obzira na duljinu kodne riječi. Međutim, treba objasniti da kodne riječi takvih kodova nemaju nužno cikličko svojstvo osim ako duljina kodne riječi nije *Pu*.

*Dokaz Tvrdnje 4.3* Kao što smo naveli, bilo koji kod što pripada skupu $\{C\}$, je sustavan. Neka **G** označava generator-matricu našega koda. Ova matrica ima gornja svojstva d) i f). Prema d), ciklički pomak bilo kojega retka **G** osim posljednjega, za jedno mjesto u desno, daje kodnu riječ. Prema f), ciklički pomak zadnjega retka od **G** za jedno mjesto u desno daje prvi redak, a to već znamo da je kodna riječ. Zaključak: Ciklički pomak *bilo kojega* retka u **G** za jedno mjesto u desno daje kodnu riječ.

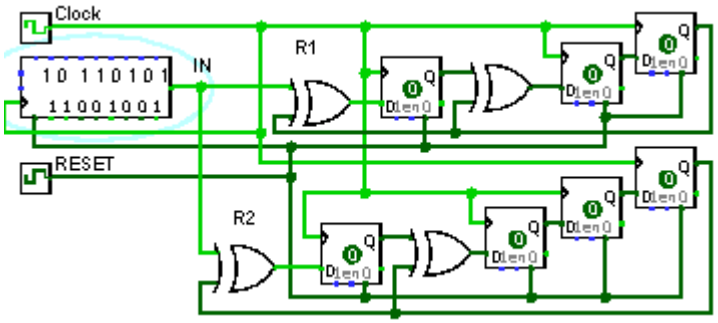
Svaka kodna riječ **c** našega koda, sastoji se od zbroja pojedinih redaka **G**. Množenjem informacijskoga vektora **u** matricom **G**, daje kodnu riječ podudarnu s **u**, gdje takva *množenja* znače *zbrajanje* nekih redaka od **G**. Ciklički pomak **c** za jedno mjesto u desno, može se smatrati pomakom svih redaka **G** čiji zbroj daje **c**. Obzirom na ono što je već iskazano, svaki takav pomak redaka matrice **G** daje kodnu riječ. Zbroj ovih kodnih riječi, a to nije ništa drugo nego pomak za **c**, također je kodna riječ.

Dokazali smo da ciklički pomak bilo koje kodne riječ našega koda za jedno mjesto u desno, daje kodnu riječ. To dokazuje da pomak kodne riječ *za bilo koliki* broj mjesta također daje kodnu riječ. Ako jedan pomak u desno daje kodnu riječ, primjenjujući jedan pomak uzastopno nekoliko puta također će dati kodne riječi, jer je svaki rezultat u tome lancu kodna riječ. Tvrdnja 4.3 time je dokazana.

## 4.4. 4.4. Koder i dekoder koda koji zadovoljava jednadžbe pariteta nekoliko neovisnih kodova

### 4.4.1. 4.4.1. PROBLEM

Slika 4.5 prikazuje sklop što se sastoji od dvaju LFSR, obilježenih kao R1 i R2, a pune se preko zajedničkoga ulaza.




$$R(x) = x^0 \oplus x^1 \oplus x^3$$

$$R(x) = 1 \oplus x^1 \oplus x^4$$

*Slika 4.5: Paralelno povezivanje dvaju LFSR*

Postoji kod *C* čije kodne riječi imaju svojstvo da nakon pomaka u krug na slici 4.5, *oba registra* (i R1 i R2) imaju sadržaj "sve 0". Ovaj kod može se stvoriti jednim LFSR. Jednostavan način koji prikazuje da takav LFSR postoji, jest primjena Tvrdnje 4.2. Zatim se bavimo priključkom (IN) između toga LFSR što generira kod i registara R1, R2.

### 4.4.2. 4.4.2. ODREĐIVANJE DULJINE LFSR ZA GENERIRANJE KODA

Duljina LFSR, koji stvara kod, jednaka je broju jednadžbi pariteta što ih kodna riječ mora zadovoljiti. Budući da se kodne riječi našega koda *C* posmiču i u R1 i u R2 (slika 4.5) dobiju se sindromi "sve 0" pa slijedi da kodne riječi u *C* moraju zadovoljiti dva neovisna skupa paritetnih jednadžbi. To su tri jednadžbe što ih određuje kod čiji dekoder je samo R1 i četiri jednadžbe što ih određuje kod čiji dekoder je samo R2.

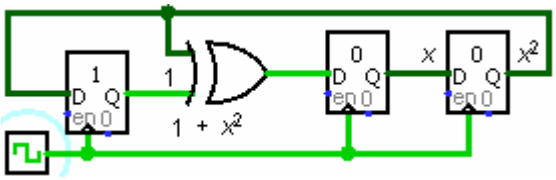
Onda iz *C* možemo izgraditi kodne riječi zadovoljenjem sedam nezavisnih paritetnih jednadžbi. Ne *mora* postojati baš točno sedam jednadžbi, ponekad ih je i manje sasvim dovoljno. To će se dogoditi ako dva skupa jednadžbi, što ih uvjetuju dva neovisna koda, imaju neke zajedničke jednadžbe. Međutim, sedam jednadžbi je sigurno dovoljno za predstaviti ograničenja pariteta oba koda pa se sljedeća rasprava temelji na tome broju. Imajte na umu da ($n$, $k$) vrijednosti od *C* nisu određene u ovome stanju. Samo smo utvrdili da je $n–k = 7$.

### 4.4.3. 4.4.3. TRAŽENJE VEKTORA ČIJI ELEMENTI PREDSTAVLJAJU POVRATNE VEZE ZAHTIJEVANOGA LFSR

Već smo spomenuli da su povratne veze u LFSR koje stvaraju ($n$, $k$) sustavan kod, označene bitovima u ($q$+1)-ome retku matrice pariteta koda, gdje je $q = n - k$. Ti isti bitovi predstavljaju prvih $q$ bitova u prvome retku generator-matrice koda.

Razjasnimo ponovno ovo, promatranjem *prvoga* retka matrice koja stvara popularan (7, 4) kod koji se koristi u ovome tekstu, a čija matrica pariteta je $\mathbf{H}^T$.

Ovaj redak je [1101000]. Prva tri bita su [110], koja naznačuju da povratna veza iz desnoga stanja LFSR koji generira kod, puni dva stanja lijevo, kako prikazuje slika 4.6.




*Slika 4.6: Povratna veza iz desnoga stanja LFSR, puni dva stanja lijevo (ponovljena slika 3.4)*

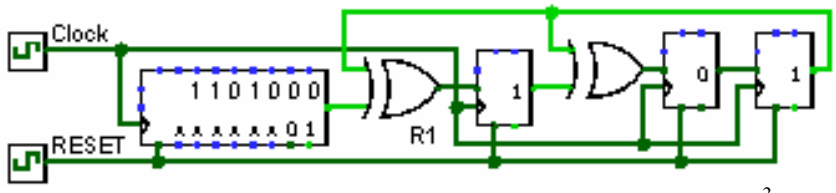
Prvi redak generator-matrice koda, čija kodna riječ $C\mathbf{i}$, odgovara informacijskome vektoru [1000 … 0], a svojstveno mu je da ima $k$–1 nula na svojoj desnoj strani (tj., ne-nulti elementi $C\mathbf{i}$ izdvojeni su na $q$+1 mjesta lijevo). Važno je napomenuti da je ($q$+1) bit u tome nizu sigurno 1, jer je to 1-element informacijskoga vektora. Također, u svim praktičnim slučajevima, prvi bit u ovome retku je 1.

Jedan od načina objašnjenja ove činjenice je pomoću opažanja da 1 na prvome mjestu, pokazuje kako se povratna veza puni u prvome stupnju LFSR koji generira kod. Ovo mora vrijediti, jer u protivnome, mogli bismo odbaciti prvo stanje bez utjecaja na ponašanje kruga. Iz toga slijedi, da bi pronašli LFSR koji stvara naš kod $C$ (za kojega smo odlučili da je $q$ = 7), dovoljno je pronaći ne-nulti vektor dužine 8, tako da nakon što ga se posmikne u R1 i R2 na slici 4.5, oba registra dovede u sadržaj "sve 0".

Nadalje, detaljnije ćemo razmotriti svojstva ovoga vektora. Punjenje vektora $\mathbf{v}_1$ = [1101000] u R1 daje [000], kao njegov konačan sadržaj.

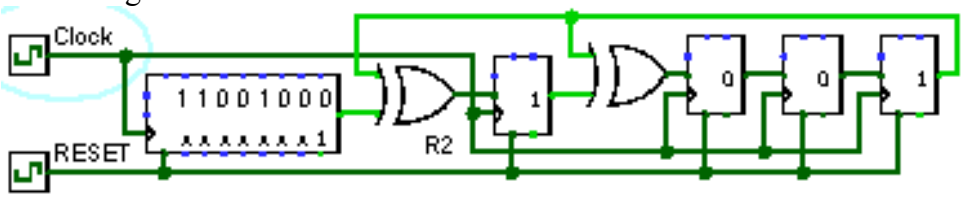
Ovo se može objasniti na dva načina.

1. Vektor [1101000] je kodna riječ $C\mathbf{i}$ koda kojega stvara R1, a dodatne 0 desno, sadržane u $\mathbf{v}_1$, ne čine nikakvu razliku.



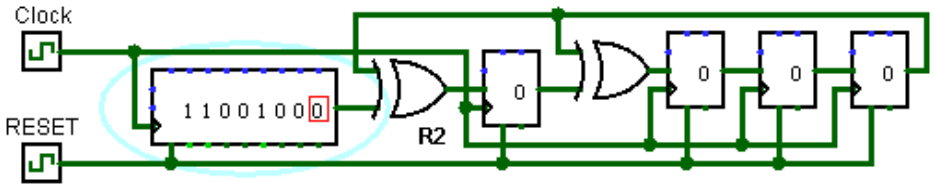

*Slika 4.7: Ponovljena slika 4.5 za R1 ($R(x) = 1 \oplus x \oplus x^3$)*

2. Provjerom, posmik [1101000] u R1 daje [000], jer 1 na desnoj strani napušta registar i ulazi u XOR vrata upravo kada se dvije 1 unose na ostala ulazna vrata. Stoga se međusobno poništavaju. Isto vrijedi i za registar R2.




*Slika 4.8: Ponovljena slika 4.5 za R2 [$R(x) = 1 \oplus x \oplus x^4$]*

Zbog dosadašnjih višestrukih objašnjenja, znamo da se vektor za punjenje sastoji od zbroja desnih linearnih posmika $\mathbf{v}_1$ u R1, gdje posmik nije veći od četiri mjesta, također daje sadržaj [000]. Slično tome, ulaz $\mathbf{v}_2$ = [11001000] u R2 daje [0000], kao njegov konačan sadržaj. Isto vrijedi i za vektor koji se sastoji od zbroja desnih linearnih posmika $\mathbf{v}_2$, gdje posmici nisu veći od tri mjesta.




*Slika 4.9: Posmik vektora koji se sastoji od zbroja desnih linearnih posmika* $\mathbf{v}_2$

Sada promatramo vektor $\mathbf{v}_3$ koji se *istodobno* sastoji od zbroja linearnih posmika $\mathbf{v}_1$ i $\mathbf{v}_2$ (gdje posmici nisu veći od četiri odnosno tri mjesta). Onda je $\mathbf{v}_3$ vektor dužine 8, a ima svojstvo da, ako ga se istovremeno posmikne u R1 i R2, oba registra nakon 8 posmika dolaze u stanje "sve 0". Vektor $\mathbf{v}_3$ je stoga vektor kojega smo tražili, a on naznačuje povratne veze ([11001]) LFSR-generatora našega koda *C*. Dokaz je u sljedećem poglavlju 4.4.4.

### 4.4.4. 4.4.4. ODREĐIVANJE VEKTORA $V_3$

*Tvrdnja 4.4*

$$\mathbf{v}_3 = [11001] \cdot \overset{\text{ciklički pomaci} >>> \rightarrow \Rightarrow \hookleftarrow}{\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}} = [10110101]$$

*Dokaz* Razmotrite sljedeće jednadžbe:

$$\mathbf{a} = [vwxyz] \cdot \begin{bmatrix} a & b & c & d & 0 & 0 & 0 & 0 \\ 0 & a & b & c & d & 0 & 0 & 0 \\ 0 & 0 & a & b & c & d & 0 & 0 \\ 0 & 0 & 0 & a & b & c & d & 0 \\ 0 & 0 & 0 & 0 & a & b & c & d \end{bmatrix} =$$

$$= [va \mid +vb+wa \mid +vc+wb+xa \mid +vd+wc+xb+ya \mid +wd+xc+yb+za \mid +xd+yc+zb \mid +yd+zc \mid +zd] =$$

$$= [va+vb+vc+vd \mid +wa+wb+wc+wd \mid +xa+xb+xc+xd \mid +ya+yb+yc+yd \mid +za+zb+zc+zd]$$

$$\mathbf{b} = [abcd] \cdot \begin{bmatrix} v & w & x & y & z & 0 & 0 & 0 \\ 0 & v & w & x & y & z & 0 & 0 \\ 0 & 0 & v & w & x & y & z & 0 \\ 0 & 0 & 0 & v & w & x & y & z \end{bmatrix} =$$

$$= [av \mid +aw+bv \mid +ax+bw+cv \mid +ay+bx+cw+dv \mid +az+by+cx+dw \mid +bz+cy+dx \mid +cz+dy \mid +dz] =$$

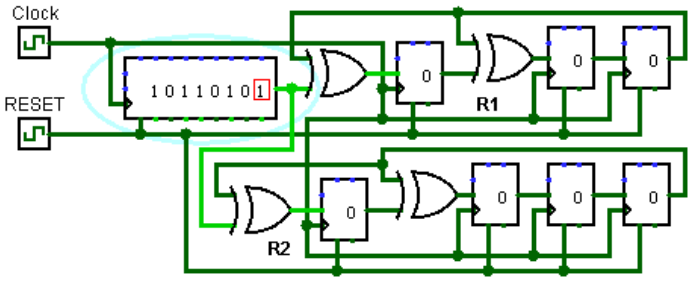
$$= [av+aw+ax+ay+az \mid +bv+bw+bx+by+bz \mid +cv+cw+cx+cy+cz \mid +dv+dw+dx+dy+dz]$$

Jednadžbe znače da je množenje vektora **a** matricom, čiji se retci sastoje od linearnih pomaka vektora **b**, jednako množenju vektora **b** matricom čiji se retci sastoje od linearnih pomaka vektora **a**. Da bismo vidjeli zašto jednadžba vrijedi, množimo na svakoj strani i provjeravamo dobije li se kao rezultat, isti vektor. Elementi ovoga vektora su **av**, **bv** ⊕ **aw**, **cv** ⊕ **bw** ⊕ **ax**, itd. Možemo napisati da je:

$$[11001] \cdot \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} = [1101] \cdot \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} = [10110101]$$

Retci dviju matrica u gornjoj jednadžbi sastoje se od linearnih posmika $\mathbf{v}_1$ i $\mathbf{v}_2$. Prema lijevoj strani jednadžbe, rezultat množenja sastoji se od zbroja pojedinih linearnih posmika $\mathbf{v}_1$.

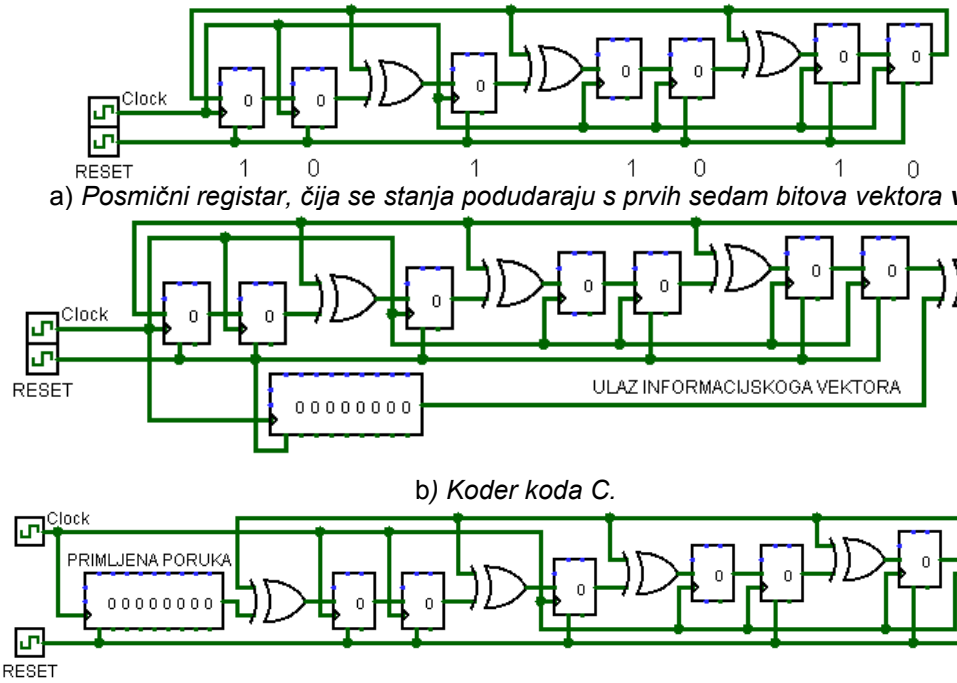
Prema desnoj strani jednadžbe, isti rezultat sastoji se od zbroja pojedinih linearnih posmika $\mathbf{v}_2$. Ovaj rezultat, a to je vektor [10110101], sastoji se od zbroja pojedinih linearnih posmika vektora $\mathbf{v}_1$ i $\mathbf{v}_2$, a to je upravo definicija $\mathbf{v}_3$. Time je dovršen dokaz Tvrdnje 4.4, a slika 4.10 praktično dokazuje tvrdnju.




*Slika 4.10: Dokaz Tvrdnje 4.4*

#### 4.4.5. 4.4.5. IZGRADNJA LFSR KOJI ODGOVARA VEKTORU $V_3$

LFSR pri stvaranju našega koda *C*, proizvodi vektor $\mathbf{v}_3 = [1 \oplus x^2 \oplus x^3 \oplus x^5 \oplus x^7] = [10110101]$, Njegovih prvih sedam bitova naznačuju povratne veze registra, kao što prikazuje slika 4.11.



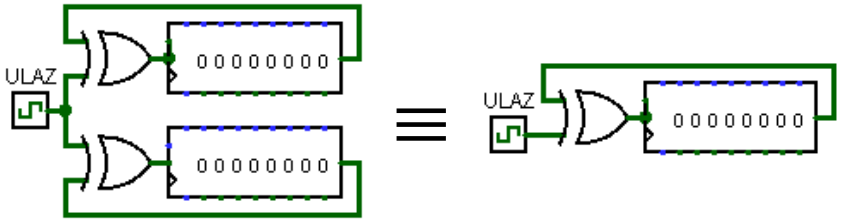

a) *Posmični registar, čija se stanja podudaraju s prvih sedam bitova vektora* $\mathbf{v}_3$.

b) *Koder koda C.*

c) *Generator sindroma koda C*

*Slika 4.11: Vektor* $R(x) = 1 \oplus x^2 \oplus x^3 \oplus x^5 \oplus x^7 = [10110101]$ *naznačuje povratne veze registra*

Prvih sedam bitova $\mathbf{v}_3$ napisano je redom ispod odgovarajućih sedam stupnjeva LFSR na slici 4.11.a. Stanje je, ili izravno (samo za prvo stanje-lijevo), ili preko XOR vrata, spojeno na povratnu vezu onda i samo onda ako ispod njega piše 1. Slike 4.11.b i 4.11c prikazuju kako se ovaj registar upotrebljava u krugovima *kodiranja* (slika 4.11.b) i *dekodiranja* (*sindrom*) koda *C* (slika 4.11.c). Sada poopćujemo postupak traženja LFSR čija je funkcija jednaka dvjema LFSR spojenima paralelno (na način da je LFSR iz slike 4.11 jednak onome na slici 4.5). Jedan krug takvoga oblika prikazuje slika 4.12.




*Slika 4.12: Općenit slučaj dvaju LFSR spojenih paralelno i jednakost paralelnoga spoja*

Cilj je pronaći LFSR koji zamjenjuje dva registra na slici 4.5. Povratne veze našega LFSR, pronaći ćemo množenjem vektora **v** matricom $\mathbf{M}^T$, gdje **v** predstavlja povratne veze jednoga registra i retke matrice $\mathbf{M}^T$ koja se sastoji od linearnih posmika vektora koji predstavljaju povratne veze drugoga registra. To se ponovno objašnjava promatranjem određenoga množenja vektora i matrice kao što se iskazalo Tvrdnjom 4.4.

Glavno pitanje koje treba postaviti je: Postoji li neki drugi LFSR čija funkcija sliči dvjema navedenim paralelno spojenim vezama, osim one pronađene našim posebnim postupkom, u kojemu duljina takvoga LFSR ne prelazi iznos dužine pojedinoga registara? Odgovor je, u mnogim slučajevima, **da**. Ovo nije slučaj za poseban spoj na slici 4.5. Takav slučaj proučava se u sljedećem poglavlju.

## 9. *Laboratorijska vježba 8: Rekurzija i stvaranje kodnih riječi cikličkim pomacima*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

lijeva i desna ciklička rotacija (*a* <<< *b*, *a* >>> *b*).

Figure 7_10.circ, Slika 4.17.circ

Rekurzija.doc, Rekurz.doc

C++ programi za stvaranje cikličkih pomaka (>>>, <<<):

Rotacija bitova (*bit rotation*) rotacija ili kružni pomak (*circular shift*) pomak je sličan posmiku osim što bitovi koji izlaze na jednome kraju pojavljuju se na drugome kraju.

Rekurzijska relacija

*Definicija* Kod je ***ciklički***, ako su vektori, dobiveni bilo kakvim cikličkim pomakom bilo koje od njegovih kodnih riječi, također kodne riječi.

*Primjer* Ciklički pomak vektora [1011110] u desno za četiri mjesta, daje vektor [1110101]. Općenito možemo definirati i lijevi ciklički pomak istoga vektora duljine $n = 7$ u desno za $i$ mjesta, a on je jednak cikličkome pomaku u lijevo za $n–i = 3$ mjesta. (vidi donji program.)

### 4.5.1. UVOD

*Uvodna napomena* U ovome dijelu prikazana je *teorija o nizovima maksimalne duljine* i nije izravno povezana teorijom o kodovima za ispravak pogrešaka. Temelji se na do sada obrađenim razmatranjima te je uključena u tekst iz dvaju razloga:

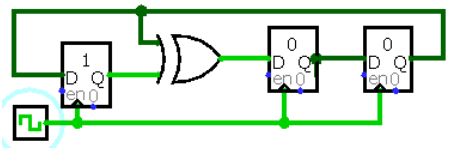
a) može poslužiti kao zgodan alat za podrobnije istraživanje predstavljenoga materijala,

b) poslužuje mnoge aplikacije u rasponu od *tehnike raspršenoga spektra* i *generiranja slučajnih brojeva* do *teorije kodiranja* i *kriptografije*.

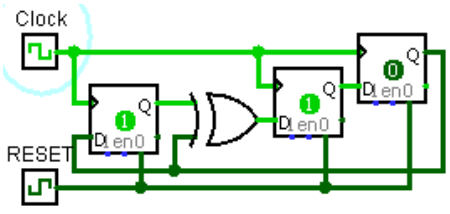
Ova materija zaslužuje svoju vlastitu skriptu, a ovdje se raspravlja samo o nekim osnovnim značajkama. Međutim, cio ovaj dio može se preskočiti, a da se i dalje očuva cjelovitost spoznaje.

*Definicija* Neka se bilo koji početni sadržaj različit od nule, posmikne $2^n-1$ puta u LFSR maksimalne periodičnosti i duljine $n$. Niz bitova duljina $2^n-1$, koji se sastoji od uzastopnih sadržaja bilo kojega stanja LFSR (koje se generira tijekom posmika), naziva se **niz maksimalne duljine, što ga generira LFSR** ili kraće **niz maksimalne duljine**.

*Primjer* Razmotrimo maksimalnu periodičnosti LFSR na slici 3.4 (ponovno nacrtana kao slika 4.2 i sada kao slika 4.13).



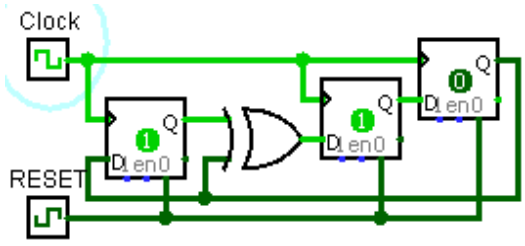
Slika 3.4




Slika 4.2

*Slika 4.13: Ponovno nacrtane: slika 3.4 i slika 4.2*

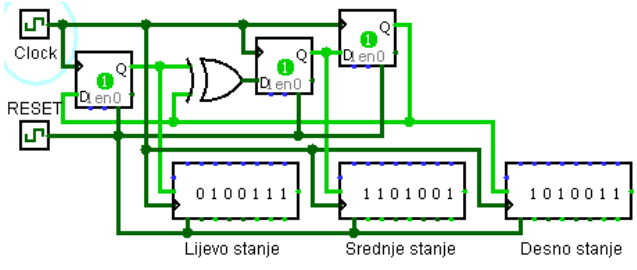
$$\mathbf{G} = [\mathbf{P} \mid \mathbf{I}_k] \Rightarrow \mathbf{H} = [\mathbf{P}^T \mid \mathbf{I}_{n-k}] \Rightarrow \quad \mathbf{H}^T = \begin{bmatrix} \mathbf{I}_{n-k} \\ \hline \mathbf{P} \end{bmatrix}$$



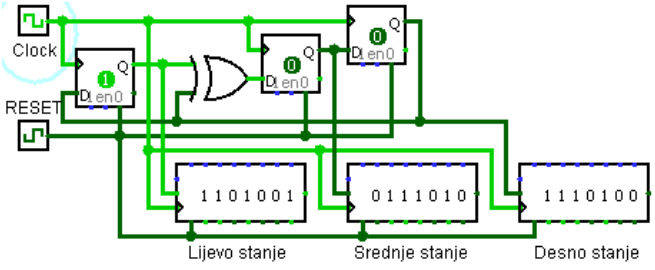

| | | | | | | |
|---|---|---|---|---|---|---|
| 1. | 1 | | | 0 | 0 | 100 |
| 2. | 0 | exor(1 ⊕ 0) | = | 1 | 0 | 010 |
| 3. | 0 | exor(0 ⊕ 0) | = | 0 | 1 | 001 |
| 4. | 1 | exor(0 ⊕ 1) | = | 1 | 0 | 110 |
| 5. | 0 | exor(1 ⊕ 0) | = | 1 | 1 | 011 |
| 6. | 1 | exor(0 ⊕ 1) | = | 1 | 1 | 111 |
| 7. | 1 | exor(1 ⊕ 1) | = | 0 | 1 | 101 |
| 1. | 1 | exor(1 ⊕ 1) | = | 0 | 0 | 100 |

*Slika 4.14: LFSR i sadržaji njegovih registara*

Sadržaji ovoga LFSR, počevši od početnoga stanja [111] su: [111], [101], [100], [010], [001], [110], [011]. Sljedeći sadržaj je opet [111]. Sadržaji triju stanja su sljedeći nizovi duljine 7: (izvorno) (slika 4.15, lijevo).




Ako je početno stanje 111




Ako je početno stanje 100

*Slika 4.15: Prikaz stanja u svim registrima i pri svakome posmiku*

Sadržaji ovoga LFSR, počevši od početnoga stanja [100] su redom: [100], [010], [001], [110], [011], [111], [101]. Sljedeći sadržaj je opet [100] (slika 4.15, desno).

Uzastopni sadržaji triju stanja registra, sljedeći su nizovi maksimalne duljine 7 prikazani u tablici:

| | Ako je početno stanje 111 | | Ako je početno stanje 100 | |
|---|---|---|---|---|
| | redoslijed pomaka | konačan prikaz | redoslijed pomaka | konačan prikaz |
| Uzastopna lijeva stanja: | 1110010 | 0100111 | 1001011 | 1101001 |
| Uzastopna srednja stanja: | 1100101 | 1010011 | 0101001 | 0111010 |
| Uzastopna desna stanja: | 1001011 | 1101001 | 0010111 | 1110100 |

Lijeva stanja [1110010].
Srednja stanja: [1100101].
Desna stanja: [1001011].

Bilo koji od tih triju nizova je niz maksimalne duljine.

Ovaj dio bavi se specifičnim svojstvima nizova maksimalne duljine.

**Svojstvo 1**. Svaki niz od ***n*** uzastopnih bitova u nizu maksimalne duljine, dužine $2^n–1$ je drugačiji i takav niz naziva se ***n-torka***. Ovo uključuje slučaj gdje se za bitove niza smatra da su poredani ciklički tj., za posljednji bit smatra se da se pojavljuje nakon prvoga bita. Jedini uzorak od *n* bitova, koji ne postoji kao jedna *n*-torka u nizu, je uzorak "sve 0".

Kao primjer uzmimo niz [1110010] duljine $2^3–1$. Počevši s lijeva, uočavamo redom sljedeće trojke: [111], [110], [100], [001], [010], [101], [011].

LFSR na slici 3.1 generator je redaka matrice $\mathbf{H}^T$ i detaljno se obradio u prethodnim poglavljima. Tri niza najveće duljine, uvedena u prethodnome primjeru, su tri ciklički pomaknuta stupca od $\mathbf{H}^T$. Savjetuje se studentu usporedba ovih nizova sa stupcima od $\mathbf{H}^T$. Iako se cijela problematika niza maksimalne duljine može obraditi bez povezivanja matricom pariteta cikličkoga Hammingova koda, odlučili smo primijeniti takvu povezanost zbog obuke.

Od sada (ako je to prikladno), usmjerit ćemo se na *stupce paritetne matrice cikličkoga Hammingova koda*, a ne na koncept *niza maksimalne duljine*, znajući da su istoznačni. Razmotrit ćemo stupčane vektore od $\mathbf{H}^T$ kao redne vektore (tj., napisat ćemo ih vodoravno umjesto okomito).

$$\mathbf{H}^T = \begin{matrix}\mathbf{h}(1)\mathbf{h}(2)\mathbf{h}(3)\\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}\end{matrix} \equiv \begin{matrix}\mathbf{h}(1) = [1001011]^T \\ \mathbf{h}(2) = [0101110]^T \\ \mathbf{h}(3) = [0010111]^T\end{matrix}$$

### 4.5.2. 4.5.2. VEZA IZMEĐU REKURZIJSKE RELACIJE I NIZA MAKSIMALNE DULJINE

Uočimo da bilo koji element $a_n$ u bilo kojemu stupcu matrice $\mathbf{H}^T$, zadovoljava relaciju

$$a_n = a_{n-2} + a_{n-3}$$

tj., svaki element, jednak je zbroju dvaju elemenata koji mu prethode dva odnosno tri mjesta lijevo.

Na primjer, razmotrimo vektor **h**(1) = [1001011]. Jedinica na četvrtome mjestu s lijeva, jednaka je zbroju (⊕) znamenaka na drugome i prvome mjestu, a one su 0 odnosno 1. Na primjer, nula na petome mjestu u vektoru **h**(1) = [1001011], jednaka je zbroj dvaju elemenata na trećemu i drugome mjestu (obje su 0). Ova relacija, poznata kao **rekurzijska relacija**, odnosi se na *ciklički* redoslijed elemenata vektora **h**(1) = [1001011]. Na primjer [1001011], prvi element=1 (kojemu prethodi sedmi element=1), jednak je zbroju šestoga=1 i petoga=0 elementa.

| rekurzijska relacija | $a_n = a_{n-2} \oplus a_{n-3}$ |
|---|---|
| 1001011 | 1 = 0 ⊕ 1 |
| 1001011 | 0 = 0 ⊕ 0 |
| 1001011 | 1 = 1 ⊕ 0 |
| … | … |

**h**(1) = 1001011 [1001011, 1001011, 1001011, 1001011, 1001011, 1001011, 1001011]

**h**(2) = 0101110 [0101110, 0101110, 0101110, 0101110, 0101110, 0101110, 0101110]

**h**(3) = 0010111 [0010111, 0010111, 0010111, 0010111, 0010111, 0010111, 0010111]

Gornja rekurzijska relacija objašnjava se uzimanjem kodne riječi **v** = [1101000] Hammingova (7, 4) koda čija je matrica pariteta $\mathbf{H}^T$. Činjenica da je **v** kodna riječ znači da je $\mathbf{v} \cdot \mathbf{H}^T = [000]$ = sindrom.

Napomene:

$$\mathbf{G} = [\mathbf{P} \mid \mathbf{I}_k] \Rightarrow \mathbf{H} = [\mathbf{P}^T \mid \mathbf{I}_{n-k}] \Rightarrow \quad \mathbf{H}^T = \begin{bmatrix} \mathbf{I}_{n-k} \\ \hline \mathbf{P} \end{bmatrix} \qquad \mathbf{G} = [\mathbf{I}_k \mid \mathbf{P}] \Rightarrow \mathbf{H} = [\mathbf{I}_{n-k} \mid \mathbf{P}^T] \Rightarrow \quad \mathbf{H}^T = \begin{bmatrix} \mathbf{P} \\ \hline \mathbf{I}_{n-k} \end{bmatrix}$$

$$\mathbf{G} = [\mathbf{P} \mid \mathbf{I}_k] \Rightarrow \mathbf{H} = [\mathbf{P}^T \mid \mathbf{I}_{n-k}] \Rightarrow \quad \mathbf{H}^T =$$

Označimo tri stupca matrice $\mathbf{H}^T$ kao: **h**(1), **h**(2) i **h**(3).

$$\mathbf{H}^T = \begin{array}{c} \\ 1. \\ 2. \\ 3. \\ 4. \\ 5. \\ 6. \\ 7. \end{array} \begin{array}{c} \mathbf{h}(1)\mathbf{h}(2)\mathbf{h}(3) \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \end{array}$$

Operacija $\mathbf{v} \cdot \mathbf{h}(1)$, gdje je **v** = [1001000] (kodna riječ), znači u praksi, zbrajanje prvoga (1.), drugoga (2.) i četvrtoga (4.) elementa od **h**(1). Kako je $\mathbf{v} \cdot \mathbf{H}^T = \mathbf{0}$, znači da je zbroj tih triju elemenata u stupcu **h**(1) jednak 0, jer su preostala 3 elementa informacijskoga vektora jednaka 0, a to se vidi u nastavku:

$$\mathbf{v} \cdot \mathbf{h}(1) = \overset{1.\,2.\quad 4.}{[1101000]} \cdot \begin{array}{c} \mathbf{h}(1) \\ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \end{array} \begin{array}{l} 1. \\ 2. \\ 3. \\ 4. \\ 5. \\ 6. \\ 7. \end{array} = 1 \cdot 1 \oplus 1 \cdot 0 \oplus 0 \cdot 0 \oplus 1 \cdot 1 \oplus \ldots \oplus 0 \cdot 1 = 1 \oplus 0 \oplus 0 \oplus 1 = \mathbf{0}$$

Budući da je kod *ciklički* (tj. ima rekurzijsko svojstvo), znači da sljedeća kodna riječ **v** = [0110100] (ciklički pomak prethodne riječi u desno) daje $\mathbf{v} \cdot \mathbf{h}(1) = \mathbf{0}$, što znači da je zbroj drugoga, trećega i petoga elementa u **h**(1) također jednak 0. Nastavkom ovoga postupka, vidjet ćemo zašto svi elementi od **h**(1) zadovoljavaju rekurzijsku relaciju $a_n = a_{n-2} + a_{n-3}$. I u trećemu nastavku ovoga postupka, sljedeća kodna riječ (ciklički pomak prethodne riječi u desno) daje $[0011010] \cdot \mathbf{h}(1) = \mathbf{0}$. To znači da je zbroj trećega, četvrtoga i šestoga elementa u **h**(1) također jednak 0. Konačno, zbroj četvrtoga, petoga i sedmoga elementa od **h**(1), za ciklički pomaknutu prethodne kodnu riječ u desno, koja je sada **v** = [0001101] pa je umnožak također jednak 0 [$\mathbf{v} \cdot \mathbf{h}(1) = \mathbf{0}$].

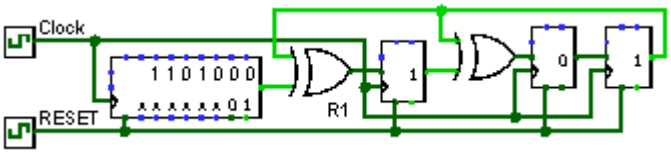
U sljedeća 3 ciklička pomaka iste kodne riječi u desno, iscrpljuju se sve raspoložive kodne riječi Hammingova koda dobivene cikličkim pomakom jedne te iste riječi. Isti argumenti sada se mogu primijeniti na stupce **h**(2) odnosno **h**(3), što objašnjava zašto oni također zadovoljavaju istu rekurzijsku relaciju.

Pokazali smo kako iz *kodne riječi* **v** = [1101000], pronaći ***sve stupce*** matrice $\mathbf{H}^T$ koji zadovoljavaju određenu rekurzijsku relaciju, gdje se koeficijenti rekurzijske relacije temelje na *ne-nultim* elementima kodne riječi **v**, tj.: relacija $a_n = a_{n-2} \oplus a_{n-3}$ odgovara npr. uzorku [1101] u kodnoj riječi. Elementi ovoga uzorka, *računajući s desna u lijevo*, odgovaraju koeficijentima:

| 1 | 1 | 0 | 1 |
|---|---|---|---|
| $a_{n-3}$ | $a_{n-2}$ | $a_{n-1}$ | $a_n$ |

rekurzijske relacije.[60]

Da smo uzeli još jednu kodnu riječ i primijenili iste argumente dane prije, navodno bi dobili još jednu rekurzijsku relaciju koja odgovara elementima ove kodne riječi različitima od nule. Međutim, iz Tvrdnje 4.1, proizlazi da se ta "druga" relacija još uvijek može izraziti našom osnovnom relacijom $a_n = a_{n-2} \oplus a_{n-3}$. Imajte na umu da rekurzijski uzorak [1101] također odgovara strukturi povratnih veza LFSR [1101] koji stvara retke matrice $\mathbf{H}^T$ (slika 4.16).




$R(x) = 1 \cdot x^0 \oplus 1 \cdot x^1 \oplus 0 \cdot x^2 \oplus 1 \cdot x^3 = [1101]$

Bez sklopa za ulazni informacijski vektor i XOR vrata (prva lijevo), jednadžba LFSR bila bi:

$R(x) = 1 \cdot x^0 \oplus 0 \cdot x^1 \oplus 1 \cdot x^2 = [101]$

*Slika 4.16: Odnos polinoma R(x), rekurzijske relacije i povratnih veza LFSR (ponovljena slika 3.4)*

Iz prije razmatrane povezanosti između nizova maksimalne duljine i stupaca matrica, prethodni argumenti mogu se poopćiti na sljedeći način:

**Svojstvo 2** Elementi niza maksimalne duljine zadovoljavaju rekurzijsku relaciju koja je zadana *strukturom povratne veze* LFSR što stvara niz. To nas vodi izravno sljedećemu svojstvu.

**Svojstvo 3** Bilo kojih *n* elemenata niza maksimalne duljine, jedinstveno određuju ostatak elemenata.

Navedeno svojstvo može se objasniti kao što slijedi. Rekurzijska relacija određena strukturom LFSR-generatora niza, pokazuje kako vrijednost pojedinoga elementa određuju vrijednosti *n* prethodnih elemenata. Drugim riječima, zadavanjem *n* uzastopnih elemenata niza, vrijednost sljedećega elementa određuje rekurzijska relacija. Posljednjih *n* elemenata izvan *n*+1-elemenata koje smo do sada imali, određuju sljedeći element, itd..

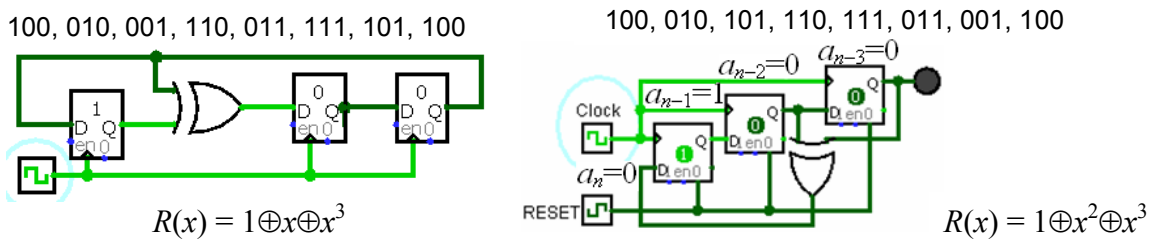
Budući da niz maksimalne duljine, dugačak $2^n-1$, sadrži sve moguće ne-nulte *n*-torke, nemoguće je da dva niza maksimalne duljine, koja zadovoljavaju istu rekurzijsku relaciju, budu različita pa je jedina razlika ciklički pomak. Prikazujemo našu logiku. Obzirom na takva dva niza, određena *n*-torka odabere se u oba. Svaka *n*-torka koja se nalazi u jednome, također se nalazi i u drugome nizu te tako obuhvaćaju sve mogućnosti. Elementi što slijede iz njih su isti u oba niza, jer oba zadovoljavaju istu rekurzijsku relaciju. Onda imamo sljedeće svojstvo.

**Svojstvo 4** Svi nizovi maksimalne duljine koji zadovoljavaju istu rekurzijsku relaciju jednaki su. Jedina razlika među njima je ciklički pomak.

Svojstvo 4 objašnjava zašto tri stupca matrice pariteta $\mathbf{H}^T$, predstavljaju ciklički pomak istoga vektora. Posmični registri (LFSR) što stvaraju retke paritetne matrice Hammingova koda, mogu se smatrati *generatorima* niza maksimalne duljine.

Jednostavno im se učita ne-nulto početno stanje i posmikne ga se $2^n-1$ puta (ako je LFSR duljine *n*). Uzastopni sadržaji bilo kojega od LFSR stanja, čine niz maksimalne duljine. Za svaki takav generator međutim, postoji *još jedan krug koji stvara isti niz*. Ova tvrdnja objašnjava se razmatranjem sklopa prikazanoga na slika 4.17 koji stvara isti niz kao i onaj što ga stvara krug na slici 3.4.

[60] Napraviti vježbu!

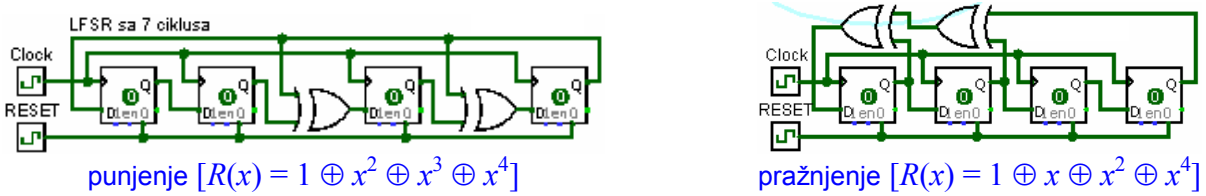



*Slika 4.17: Dva generatora niza maksimalne duljine što ga stvara krug na slici 3.4*

Da bi razumjeli zašto krugovi na slikama 3.4 i 4.17 generiraju isti (sličan, vidi redoslijed stanja) niz maksimalne duljine (skup stanja je isti), primijetimo da je u ovome posljednjemu, svaka nova *priprema ulaska* bita u LFSR s lijeve strane. Pripremljen bit na ulazu u 1. stanje LFSR, ulazi tek pri sljedećemu taktu (na slici 4.17. desno). Ovaj bit na izlazu XOR vrata odnosno *na ulazu* u prvo stanje, $a_n$=0, a ne bit koji se već *nalazi* u prvome stanju, $a_{n-1}$=( 1 ), jednak je zbroju dvaju bitova koji mu prethode 2 i 3 mjesta (2. i 3. *stanje* LFSR odnosno *izlazi* iz tih stanja, jer se stanje D bistabila, ***odmah*** preslikava na *izlaz* D bistabila). To se na slici uočava, jer 0 na ulazu u 1. registar, još mu nije promijenila stanje ( 1 ). Drugim riječima, *vidimo jasno da je zadovoljena rekurzijska relacija* $a_n = a_{n-2} \oplus a_{n-3}$. Dakle, rekurzijska relacija jedinstveno određuje niz maksimalne duljine (Svojstvo 4).

Krug na slici 3.4 koristi povratne veze "*punjenja*" (LFSR s unutarnjom povratnom vezom), tj., izlaz iz zadnjega (desnoga) stanja puni unatrag neka prethodna stanja. Krug slike 4.17 koristi vezu "*pražnjenja*" (LFSR s vanjskom povratnom vezom), gdje se sadržaji nekih stanja zbrajaju XOR vratima i zatim pune prvo (lijevo) stanje.

**Svojstvo 5** Niz maksimalne duljine može se generirati pomoću dviju vrsta LFSR (slika 4.18), korištenjem veze *punjenja* (*fed-in*) (Galois)[61] ili *pražnjenja* (*fed-out*) (Fibonacci) (*Pu* = max = 7).




*Slika 4.18: Punjenje (Galois) i pražnjenja (Fibonacci) LFSR*

Razmotrit ćemo operaciju zbrajanja nekoliko cikličkih pomaka istoga niza maksimalne duljine. Uočite, na primjer, da zbroj bilo kojega podskupa triju stupaca matrice $\mathbf{H}^T$, a to su okomiti ciklički pomaci nekoga niza maksimalne duljine, ponovno daju ciklički pomak istoga niza (npr., zbroj prvoga i trećega stupca matrice $\mathbf{H}^T$ je [1011100]. Zbroj svih triju stupaca iste matrice jednak je [1110010]. Prethodne tvrdnje temelje se na samorazumljivoj činjenici da *ako svaki niz za sebe zadovoljava rekurzijsku relaciju, onda i* ***zbroj*** *svih pod-nizova, zadovoljava tu istu relaciju*.

**Svojstvo 6** Zbroj bilo kojega podskupa triju stupaca matrice $\mathbf{H}^T$, a to su okomiti ciklički pomaci nekoga niza maksimalne duljine, ponovno daju ciklički pomak istoga niza. Na primjer zbroj prvoga i trećega stupca je [1011100]. Zbroj svih triju stupaca je [1110010]. Ako je svaki element u svakome nizu jednak zbroju dvaju prethodnih elemenata 2 i 3 mjesta prije, to isto vrijedi i za njihov zbroj. Zbroj cikličkih pomaka niza neke maksimalne duljine, daje isti niz, a određuje ga ciklički pomak (odnosno struktura LFSR).

### 4.5.3. 4.5.3. PSEUDO-SLUČAJNI NIZOVI

U ovome pod-poglavlju bavimo se množenjem i zbrajanjem brojeva koji nisu nužno samo 0 ili 1. Zbroj će ponekad znači obično zbrajanje brojeva, a ponekad će značiti XOR operaciju, kao što se najčešće čini u ovoj skripti. Da bi se izbjegla pomutnja, operaciju običnoga zbroja označavamo +, a

[61] Évariste Galois (Francuz: [evaʁist ɡa'lwa]; 25 listopada 1811 – 31 svibnja 1832) bio je Francuski matematičar rođen u Bourg-la-Reine. Još kao mladić-adolescent, odredio je nužne i dovoljne uvjete za rješenje polinoma pomoću *n*-tih korijena, time rješavajući problem star 350 godina. Njegov rad utemeljio je *Galois teoriju* i *teoriju grupe*, dva osnovna područja apstraktne algebre te *pod-polje* Galois veza. Umro je u dobi od 20 godina od rana zadobivenih u dvoboju.

XOR operaciju, $\oplus$ (kao i do sada). Opća teorija koja se bavi različitim definicijama slučajnosti, izvan je okvira ove skripte, a ovdje se obrađuje samo jedno njezino motrište.

*Definicija* Neka $\mathbf{v}_i$ označava vektor dobiven pomicanjem elemenata vektora $\mathbf{v}$ ciklički za $i$ mjesta u desno. **Ciklička auto-korelacija** za $\mathbf{v}$ je vektor iste duljine čiji je $i$-ti element skalarni umnožak $\mathbf{v} \cdot \mathbf{v}_i$.

*Primjer* Ciklička *auto-korelacija* vektora $\mathbf{v} = [1, 2, 3, 4, 5]$, je vektor $A(\mathbf{v}) = [A_0, A_1, A_2, A_3, A_4]$, gdje su:

$A_0 = [1, 2, 3, 4, 5] \cdot [1, 2, 3, 4, 5] = 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3 + 4 \cdot 4 + 5 \cdot 5 = 1 + 4 + 9 + 16 + 25 = 55$
$A_1 = [1, 2, 3, 4, 5] \cdot [5, 1, 2, 3, 4] =$ … 45
$A_2 = [1, 2, 3, 4, 5] \cdot [4, 5, 1, 2, 3] =$ … 40
$A_3 = [1, 2, 3, 4, 5] \cdot [3, 4, 5, 1, 2] =$ … 40
$A_4 = [1, 2, 3, 4, 5] \cdot [2, 3, 4, 5, 1] =$ … 45

Možemo opisati *auto-korelacijsku funkciju vektora* $\mathbf{v}$ pomoću množenja vektora matricom $\mathbf{v} \cdot \mathbf{M}^T$, gdje se stupci $\mathbf{M}^T$ sastoje od poredanih cikličkih pomaka od $\mathbf{v}$.

*Primjer* Ciklička auto-korelacija vektora [*abcde*], rezultat je umnoška:

$$[abcde] \cdot \begin{bmatrix} a & e & d & c & b \\ b & a & e & d & c \\ c & b & a & e & d \\ d & c & b & a & e \\ e & d & c & b & a \end{bmatrix}$$

Koristeći neposredno stečene spoznaje, možemo reći da elementi funkcije auto-korelacije pokazuju rezultate dobivene iz "preklapanja" elemenata vektora i cikličkoga nizanja tih elemenata te pokušava tražiti sličnosti. Uzmimo određen vektor $\mathbf{v} = [-1, -1, -1, +1, +1, -1, +1]$. Ciklička auto-korelacija $\mathbf{v}$ je:

$$[-1, -1, -1, 1, 1, -1, 1] \cdot \begin{bmatrix} -1 & 1 & -1 & 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & 1 & -1 \\ -1 & 1 & 1 & -1 & -1 & -1 & 1 \\ 1 & -1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix} = [7, -1, -1, -1, -1, -1, -1]$$

Prvi element ove funkcije, rezultat je postignut "preklapanjem" vektora samim sobom, dajući neku vrstu vrha funkcije (u gornjemu primjeru to je broj 7). Ako su ostale vrijednosti funkcije auto-korelacije stalno male, to ukazuje na poremećaj među elementima vektora, a to naznačuje "dobra" svojstva slučajnosti.

Na temelju auto-korelacijske funkcije od $\mathbf{v}$, za njegove elemente smatra se da imaju optimalnu slučajnost. Studenti upoznati temom, prepoznat će ovdje "određenu $\Delta$ funkciju". U nastavku pokazujemo kako smo došli do ovoga vektora $\mathbf{v}$. Pogledajmo prvo jednostavnu vezu između XOR operacije i množenja brojeva s predznakom (tablica 4.1).

*Tablica 4.1 Pokazuje sličnost između operacije XOR i množenja brojeva s predznakom*

| | | |
|---|---|---|
| $0 \oplus 0 = 0$ | $0 \cong +1$ | $(+1) \cdot (+1) = +1$ |
| $0 \oplus 1 = 1$ | $1 \cong -1$ | $(+1) \cdot (-1) = -1$ |
| $1 \oplus 0 = 1$ | $\oplus \cong \cdot$ | $(-1) \cdot (+1) = -1$ |
| $1 \oplus 1 = 0$ | | $(-1) \cdot (-1) = +1$ |

Uspoređujući dva stupca u tablici 4.1 uočavamo zanimljivu sličnost. Operacija XOR ($\oplus$) može se "pretvoriti" u operaciju umnoška ($\cdot$) između brojeva s predznakom apsolutne vrijednosti 1, sljedećom zamjenom:

$$\oplus \rightarrow \cdot \; ; \; 0 \rightarrow 1 \; ; \; 1 \rightarrow -1$$

Pogledajmo sada kako Svojstvo 6 pretvara niz maksimalne duljine, koristeći gore navedene zamjene. S lijeve strane ispod, imamo XOR dvaju cikličkih pomaka niza maksimalne duljine te dobivamo rezultat koji je ciklički pomak istoga niza. Na desnoj strani pišemo istu operaciju (skalarni umnožak) pomoću navedene zamjene, označavajući dobivene vektore kao **x**, **y** i **z**:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1110010 | | | –1, | –1, | –1, | 1, | 1, | –1, | 1 | | = **x** |
| 1011100 | ⊕ | | –1, | 1, | –1, | –1, | –1, | 1, | 1 | · | = **y** |
| 0101110 | | | 1, | –1, | 1, | –1, | –1, | –1, | 1 | | = **z** |

Imajte na umu da je na temelju definicije skalarnoga umnoška između dvaju vektora, *zbroj elemenata* **z** jednak *skalarnome umnošku* **x** · **y**.

I u gornjemu prikazu, ovaj skalarni umnožak (tj., zbroj elemenata **z)** jednak je –1. Sada tvrdimo: *ako* **x** *i* **y** *potječu od različitih pomaka* u *nizu maksimalne duljine*, *onda je njihov skalarni umnožak uvijek jednak* –1, *bez obzira na dimenzije niza maksimalne duljine*. Promatranjem se uočava da broj elemenata vrijednosti 1 i vrijednosti –1 u **x**, **y** i **z** odgovara broju 0-elemenata i 1-elemenata u izvornome nizu maksimalne duljine. Na temelju Svojstva 1 za niz maksimalne duljine, izravno slijedi da u takvome nizu bilo koje duljine $2^n$–1, broj 1-elemenata premašuje za 1, broj 0-elemenata.

Onda slijedi da je razlika između broja elemenata vrijednosti –1 i broja elemenata vrijednosti 1 u **z**, jednaka 1. Zbroj elemenata **z** (tj., skalarni umnožak **x** i **y)** je, dakle –1. Ako su nizovi **x** i **y** (duljine $2^n$–1) identični, a njihov skalarni umnožak iznosi $2^n$–1, jer mi zbrajamo $2^n$–1-elemenata vrijednosti 1, onda imamo sljedeće:

**Svojstvo 7** Neka **s** označava niz elemenata vrijednosti ±1, dobiven iz niza maksimalne duljine, dužine $2^n$–1 prema opisanome postupku zamjene. Onda auto-korelacijska funkcija *S* ima vrijednost $2^n$–1 na mjestu 0 (vidi broj 7 u gornjemu umnošku), dok ostatak elemenata ima stalnu vrijednost –1.

Svojstvo 7 objašnjava zašto se nizovi maksimalne duljine ponekad nazivaju *pseudo-slučajni nizovi* PN (*pseudonoise*). Riječ "*slučajno*" sada je jasna. Riječ "*pseudo*" znači da ako ponovljeno koristimo postupak stvaranja nizova, oni se mogu točno obnoviti za razliku od pravih šumova. Točan preslik našega niza može se ponoviti ako se koristi isti LFSR-generator, jer *cikličke pomake započinje istim početnim stanjem*.

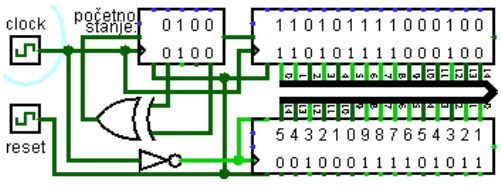
### 4.5.4. 4.5.4. ODREĐIVANJE POVRATIH VEZA LFSR ŠTO STVARA NIZ MAKSIMALNE DULJINE POZNAJUĆI SAMO DIO NIZA

#### *4.5.4.1. 4.5.4.1. Koeficijenti rekurzijskoga odnosa*

**Svojstvo 8** Povratne veze nekoga LFSR duljine *n*, što stvaraju niz maksimalne duljine MLSG (*maximum length sequence generator*), mogu se odrediti iz bilo kojih uzastopnih 2*n* bitova stvorenoga niza.

Prethodno svojstvo prije svega dokazuje se razmatranjem da povratne veze nekoga LFSR što stvara niz maksimalne duljine, odgovaraju koeficijentima *rekurzijske relacije* koji obilježavaju niz. Onda smo suočeni problemom pronalaska rekurzijske relacije. Način kako je pronaći, pokazuje se primjerom.

Niz **m** = [110101111000100] (bitovi sasvim lijevo su zadnji generirani bitovi), niz je maksimalne duljine i periodičnosti *Pu* = 15 (generira ga LFSR duljine 4), a prikazuje ga Slika 4.19.



$R(x) = 1 + x^4$

*Slika 4.19: Niz maksimalne periodičnosti*

Svaki element $a_n$, prethodno navedenoga niza **m**, zadovoljava rekurzijski *odnos* (*relaciju*) oblika:

$$a_n = A \cdot a_{n-1} + B \cdot a_{n-2} + C \cdot a_{n-3} + D \cdot a_{n-4},$$

gdje su A, B, C i D *koeficijenti rekurzijskoga odnosa*. Budući da isti koeficijenti vrijede za bilo koji $a_n$, možemo pronaći njihove vrijednosti izgradnjom četiriju nezavisnih jednadžbi, koristeći različit $a_n$.

Jednostavno se može pokazati da je D uvijek 1 (= $a_1 = x^0$), ostavljajući nam tri nepoznanice. Međutim, ovdje to nećemo koristiti pa smatramo da je D nepoznat. Jednadžba s četiri nepoznanice (A, B, C, D) sastavljena je od pet uzastopnih bitova iz **m**. Uzmite na primjer, prvih pet bitova $\mathbf{m}_1$ niza [11010]. Možemo ih indeksirati kao $a_1$, $a_2$, $a_3$, $a_4$, $a_5$. Onda opća relacija:

$$a_n = A \cdot a_{n-1} + B \cdot a_{n-2} + C \cdot a_{n-3} + D \cdot a_{n-4},$$

poprima oblik

$$a_5 = A \cdot a_4 + B \cdot a_3 + C \cdot a_2 + D \cdot a_1.$$

| **m** = [1101011111000100] | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | DCBA | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ | $a_n$ | = | A·$a_{n-1}$ | + | B·$a_{n-2}$ | + | C·$a_{n-3}$ | + | D·$a_{n-4}$ | |
| $\mathbf{m}_1$ = [11010111100010 0] | | | | | | | $a_5$ | = | A·$a_4$ | + | B·$a_3$ | + | C·$a_2$ | + | D·$a_1$ | |
| (1) | 11010 | 1 | 1 | 0 | 1 | 0 | 0 | = | 1 | + | 0 | + | 1 | + | 1 | A+C+D = 0 |
| $\mathbf{m}_2$ = [1 10101 111000100] | | | | | | | $a_5$ | = | A·$a_4$ | + | B·$a_3$ | + | C·$a_2$ | + | D·$a_1$ | |
| (2) | 10101 | 1 | 0 | 1 | 0 | 1 | 1 | = | 0 | + | 1 | + | 0 | + | 1 | B+D = 1 |
| $\mathbf{m}_3$ = [11 01011 11000100] | | | | | | | $a_5$ | = | A·$a_4$ | + | B·$a_3$ | + | C·$a_2$ | + | D·$a_1$ | |
| (3) | 01011 | 0 | 1 | 0 | 1 | 1 | 1 | = | 1 | + | 0 | + | 1 | + | 0 | A+C = 1 |
| $\mathbf{m}_4$ = [110 10111 1000100] | | | | | | | $a_5$ | = | A·$a_4$ | + | B·$a_3$ | + | C·$a_2$ | + | D·$a_1$ | |
| (4) | 10111 | 1 | 0 | 1 | 1 | 1 | 1 | = | 1 | + | 1 | + | 0 | + | 1 | A+B+D = 1 |

Uvrštenjem vrijednosti [11010] iz $\mathbf{m}_1$ počevši od **prvoga** bita za $a_1$, $a_2$, $a_3$, $a_4$, $a_5$, dobivamo

(1) $0 = D \cdot 1 + C \cdot 1 + B \cdot 0 + A \cdot 1.$

Slično tomu, pet uzastopnih bitova iz $\mathbf{m}_2$, počevši od **drugoga** bita, su [10101]. Oni čine jednadžbu:

(2) $1 = D \cdot 1 + C \cdot 0 + B \cdot 1 + A \cdot 0.$

Pet uzastopnih bitova iz $\mathbf{m}_3$, počevši od **trećega** bita, su [01011] i tvore jednadžbu:

(3) $1 = D \cdot 0 + C \cdot 1 + B \cdot 0 + A \cdot 1.$

Pet uzastopnih bitova iz $\mathbf{m}_4$, počevši od **četvrtoga** bita, su [10111], tvoreći jednadžbu:

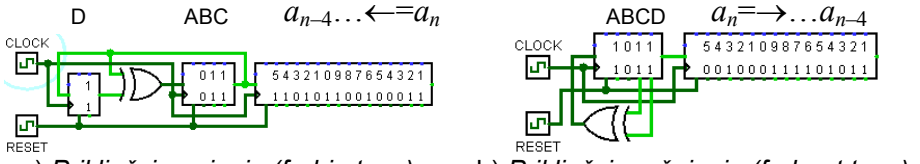
(4) $1 = D \cdot 1 + C \cdot 0 + B \cdot 1 + A \cdot 1.$

| | | | | | | |
|---|---|---|---|---|---|---|
| D+C+A | = 0 | | (1–B)+C+(1–C) = 0 | ⇒ | B=0 | |
| D+B | = 1 | ⇒ | D=(1–B) | | D=1 | |
| C+A | = 1 | | A=(1–C) | | A=0 | |
| D+B+A | = 1 | | (1–B) +B+(1–C) = 1 | ⇒ | C=1 | |

Treba napomenuti da je u *algebri binarnih brojeva*, operacija *oduzimanja* jednaka operaciji *zbrajanja*.

Rješenja za četiri nepoznanice su dakle: A = 0, B = 0, C = 1, D = 1. Naša rekurzijska relacija onda postaje $a_n = a_{n-3} + a_{n-4}$, a ona odgovara LFSR-generatoru na slici 4.20.b.

Rezultati su zrcalne slike, a razlika je u smjeru čitanja (desno→ ili ←lijevo)




a) *Priključci punjenja (fed-in taps).* b) *Priključci pražnjenja (fed-out taps).*

| $a_{n-4}$ | … | … | … | $a_n$ | |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 0 | 1. |
| 1 | 0 | 1 | 0 | 1 | 2. |
| 0 | 1 | 0 | 1 | 1 | 3. |
| 1 | 0 | 1 | 1 | 1 | 4. |

*Slika 4.20: Podudarnost LFSR i rekurzije $a_n = a_{n-3} + a_{n-4}$. (a) i (b).*[62]

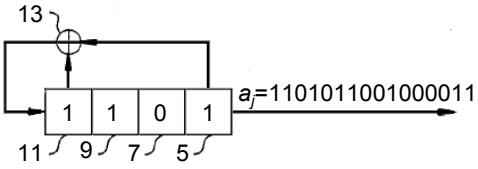
Rezultati se potvrđuju provjerom rekurzijske jednadžbe na bilo kojih 5 uzastopnih bitova niza maksimalne duljine i periodičnosti 15, **m** = [110101111000100], dobivenih sklopovima LFSR na slici 4.20.a.

[62] Napraviti kao vježbe!

Imajte na umu da smo četiri jednadžbe, iz kojih pravimo naš LFSR-generator, dobili uzimanjem osam uzastopnih bitova od **m**, počevši prvim bitom. Za istu svrhu (tj., izgradnju četiriju jednadžbi) moglo bi poslužiti uzimanje bilo kojih osam uzastopnih bitova iz **m**, a to objašnjava opći slučaj naveden u Svojstvu 8. Nismo dokazali neovisnost ovako dobivenih jednadžbi, međutim, ona je istinita.

#### 4.5.4.2. 4.5.4.2. Objašnjenje pojma "decimation"

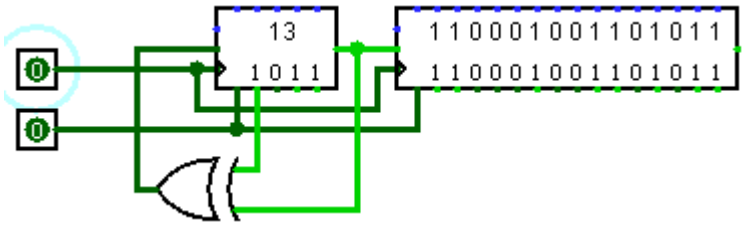
Uočite da je broj odgovarajućih (relativno prostih elemenata u odnosu na *Pu*) cikličkih koskupova (*cyclotomic cosets*) jednak broju svih primitivnih polinoma stupnja *L*.




*Slika 4.21: Generator cikličkih koskupova*

Na primjer za slučaj *Pu* = 15, brojevi {1, 2, 4, 8} oblikuju odgovarajući *ciklički koskup*, a {7, 14, 13, 11} oblikuju još jedan takav koskup, jer to "uzorkovanje/'desetkovanje'/~~kljaštrenje/bušenje~~" (*decimation*)[63] zadovoljava Tvrdnju 4.1. Tako, bilo koji 15 znamenkasti *m* niz može se uzorkovati s 2, 4 ili 8, a rezultirajući niz bit će još jedan 15 znamenkasti ciklički ekvivalent *m* niza koji može imati iste ili različite fazne pomake kao izvorne nizove. Isto tako, ako se bilo koji *m* niz od 15 znamenaka uzorkuje sa 7, a rezultirajući uzorkovani niz ponovo se uzorkuje s 2, 4 i 8, dobit će se drugi ciklički ekvivalent koskupa {7, 14, 13, 11}. Imajte na umu da uzorkovanje sa 7 × 4 = 28, ekvivalent je uzorkovanju s 13, jer je 28 – 15 = 13.

Tako, u primjeru na slici 4.21, uzorkovanje skupa $\{a_j\}$ s 2, proizvodi $\{a_{2j}\}$ = [100100011110101], što je $\{a_j\}$ lijevi pomak 6 znamenaka $\{a_j\}$, odnosno, $\{a_{2j}\} = \{a_j + 6\}$.




*Slika 4.22: U. S. Patent Jul. 5, 1988 Sheet 1 of 8 4,755,987 (stranica 45) i simulacijski primjer*

Imajte na umu da na primjer, uzorkovanjem s 2, uzorkovan niz obuhvaća prvu, treću, petu, itd. znamenku izvornoga niza. Isto tako pri uzorkovanju sa 7, uzorkovan niz obuhvaća prvu, osmu, petnaestu, itd. znamenku izvornika. Također, na slici 2, ako se niz uzorkuje sa 7, $\{a_{7j}\}$ = [101111000100110] i 13, $\{a_{13j}\}$ = [110101111000100], to su nizovi koje mogu generirati LSFR s četiri stanja uz $e = 31_8 = 11001_2$, ali različitim početnim stanjima, $a = 13_{10} = 1101_2$ odnosno $a = 15_{10} = 1111_2$ pa su to isti nizovi s različitim pomacima te stoga pripadaju istomu odgovarajućemu koskupu.

### 4.5.5. 4.5.5. UNAPRJEĐENJE OPERACIJA U NIZOVIMA MAKSIMALNE DULJINE

Zadavanjem bilo kakve ne-nulte *n*-torke **t**, moguće je odrediti sve bitove prateći **t** u nizu maksimalne duljine periodičnosti $2^n$–1 posmikom u LFSR za generiranje (povratnim vezama pražnjenjem), počevši od stanja **t**.

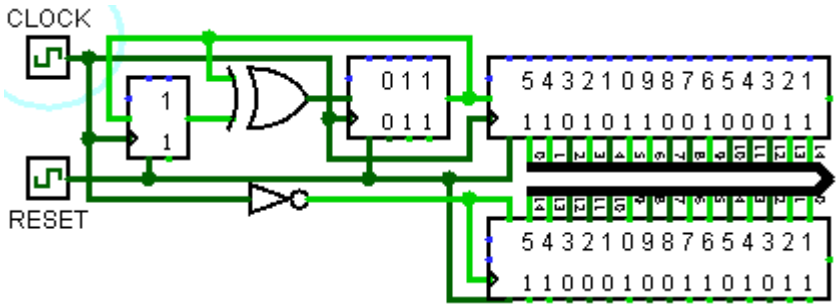
#### 4.5.5.1. 4.5.5.1. Struktura niza maksimalne duljine

U sljedećemu primjeru, kao i svima onima što slijede, usvajamo *računanje s desna u lijevo*. To znači da niz [101011] započinje podcrtanim 4-kama. Njegov prvi, drugi i treći bit su redom 1, 1 i 0. Razlog ovomu dogovoru je izbjegavanje zbunjenosti, ako se analizira ponašanje LFSR s *desnim* posmikom. Kao primjer načina kako se otkrivaju bitovi što slijede iz **t**, razmotrimo 4-vorku **t** = [1011]. U nizu

---

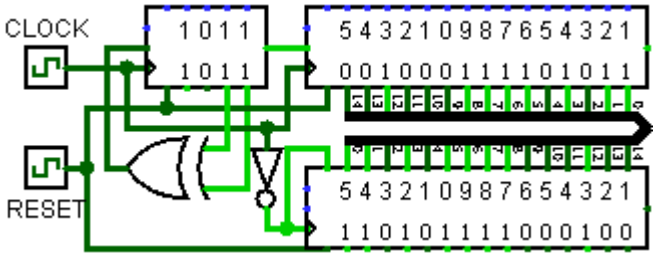
[63] Ako su *b* i *n* relativno prosti brojevi, onda $i \equiv b_i \pmod{n}$ definira permutaciju vektora **c**. Permutacija $c_i' = c_{((bi))}$ zove se cikličko "uzorkovanje/'desetkovanje'/~~kljaštrenje/bušenje~~" (*cyclic decimation*), jer se svaka *b*-ta komponenta, odabire ciklički (ili se prihvaća ~~ili se odbacuje~~).

maksimalne duljine stvorenome pomoću LFSR na slici 4.22.a, negdje se mora pojaviti **t** i to samo jednom, poput bilo koje druge ne nulte 4-vorke.




*Slika 4.23: Niz maksimalne duljine stvoren pomoću LFSR punjenjem*

Da bi se pronašli bitovi što slijede iza **t** u ovomu nizu, napunimo **t** u LFSR kao na slici 4.22.b. odnosno na slici 4.24.




slika 4.22.b

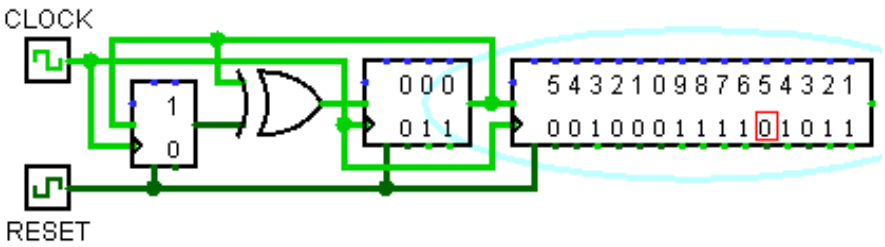
*Slika 4.24: Punjenje vektora **t** u LFSR pražnjenjem*

Uzastopan posmik registra generirat će niz maksimalne duljine [001000111101011]. Imajte na umu da se krajnji desni bitovi prvi odstranjuju iz LFSR. Razlog odabira LFSR s povratnom vezom pražnjenja (*fed-out feedback*) (slika 4.22.b) je taj, što posmik i pražnjenje prvih *n* bitova, dovodi do toga da su početni sadržaji (na desnoj strani LFSR jednaki **t**, a to nije slučaj za registre s povratnom vezom koji imaju priključke za punjenje (*fed-in connections*) (slika 4.22.a).

Sada obrađujemo sljedeći problem. Kako se može mijenjati niz što ga generira LFSR, stvarajući niz maksimalne duljine periodičnosti $2^n$–1, tako da bez obzira na to puni li se (*fed-in*) početnom *n*-torkom **t**, izlazni niz će započeti od *i*-toga mjesta nakon **t**, za bilo koji *i*. Da bi razjasnili problem, uzmimo niz [001000111101011]. On počinje 4-vorkom **t** = [1011]. Polazeći od 5-oga mjesta (s desna), niz je [… 00011110]. Pitanje glasi, kako preurediti krug na slici 4.22.b, tako da, ako ga napuni *n*-torka **t**, onda on stvara niz [… 00011110].

**Svojstvo 9**. Neka je **m** niz maksimalne duljine i periodičnosti $2^n$–1, koji započinje *n*-torkom **t**. Neka LFSR generira **r** iz **m**, vezama punjenja (*fed-in connections*) i neka $\mathbf{r}_i$ označava sadržaj **r** nakon *i*–1 posmika, počevši početnim sadržajem $\mathbf{r}_1$ = [1000 … 0]. Neka $\underline{\mathbf{r}}$ označava "zrcalni" (*reflected*) $\mathbf{r}_i$ (tj. $\mathbf{r}_i$ se piše obrnutim redoslijedom). Vrijednost *i*-toga bita od **m** je $\mathbf{t} \cdot \underline{\mathbf{r}}_i$ .

Prethodno svojstvo zapravo je ponovljena tvrdnja Zaključka 4.3 pa se ovdje neće izravno dokazati. Objasnit ćemo ga primjerom LFSR na slici 4.25.




*Slika 4.25: Objašnjenje tvrdnje Zaključka 4.3 primjerom LFSR (punjenje)*

Rad registra započinje sadržajem $\mathbf{r}_1$ = [1000]. Nakon 4 posmika, sadržaj registra je $\mathbf{r}_5$ = [1100], a njegova zrcalna slika je $\underline{\mathbf{r}}_5$ = [0011]. Podcrtan bit #5 u nizu **m** = [001000111101011], dobije se

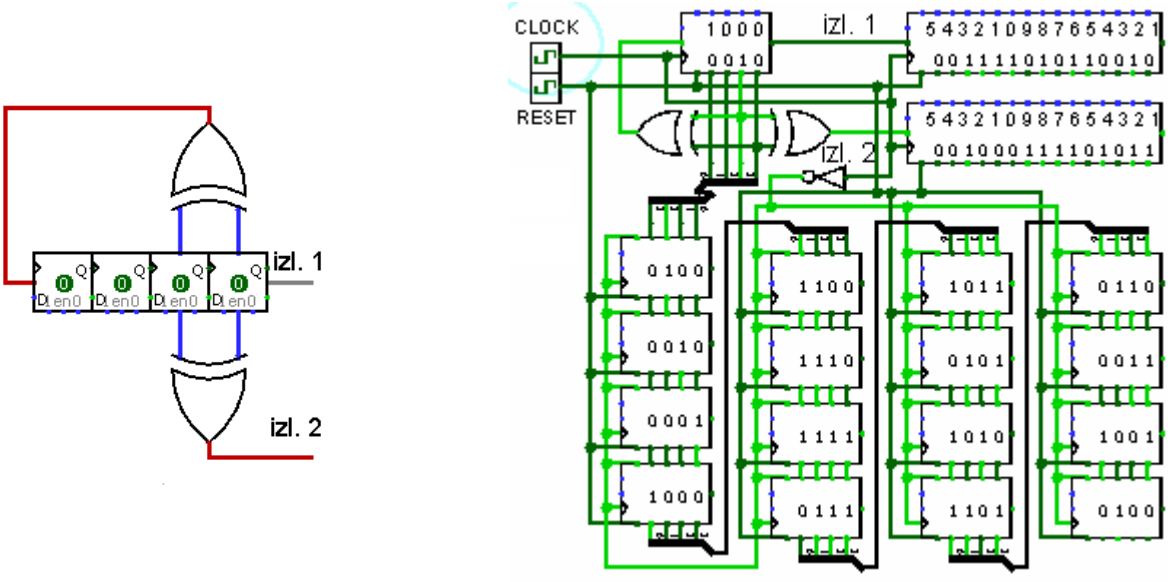
množenjem $\mathbf{t} \cdot \underline{\mathbf{r}}_5 = [1011] \cdot [0011] = 0$. Da bi se pronašao bit #6 u gornjemu nizu **m**, možemo izvoditi operaciju $[1011] \cdot \underline{\mathbf{r}}_6$ ili operaciju $[0101] \cdot \underline{\mathbf{r}}_5$.

| **m** = [0010001111010111] | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **t** | $\underline{\mathbf{R}}_i$ | $\mathbf{t} \cdot \underline{\mathbf{R}}_i$ | #5 | #6 | #7 | #8 | #9 | #10 | #11 | #12 | #13 | #14 | #15 |
| | 0001 | | | | | | | | | | | | |
| | 0010 | | | | | | | | | | | | |
| | 0100 | | | | | | | | | | | | |
| | 1000 | | | | | | | | | | | | |
| 1011 | 0011 | 1011 · 0011 | 0 | | | | | | | | | | |
| 1011 | 0110 | 1011 · 0110 | | 1 | | | | | | | | | |
| 1011 | 1101 | 1011 · 1101 | | | 0 | | | | | | | | |
| 1011 | 1010 | 1011 · 1010 | | | | 0 | | | | | | | |
| 1011 | 0101 | 1011 · 0101 | | | | | 1 | | | | | | |
| 1011 | 1011 | 1011 · 1011 | | | | | | 1 | | | | | |
| 1011 | 0111 | 1011 · 0111 | | | | | | | 0 | | | | |
| 1011 | 1111 | 1011 · 1111 | | | | | | | | 1 | | | |
| 1011 | 1110 | 1011 · 1110 | | | | | | | | | 0 | | |
| 1011 | 1100 | 1011 · 1100 | | | | | | | | | | 1 | |
| 1011 | 1000 | 1011 · 1000 | | | | | | | | | | | 1 |

Imajte na umu da je [0101] 4-vorka koja započinje u **m** na mjestu #2 na gornjoj slici (pazi: zrcalna slika s desna u lijevo).

To objašnjava sljedeće načelo: da bi se generirali bitovi od **m**, počevši bitom #5, možemo generirati uzastopne 4-vorke od **m**, počevši s **t** i pomnožiti svaki bit s $\underline{\mathbf{r}}_5$.

Budući je $\underline{\mathbf{r}}_5 = [0011]$ (sadržaj registra nakon 4. posmika), množenjem 4-vorke **x** s $\underline{\mathbf{r}}_5$ praktično znači zbrajanje prva dva bita od **x** (jer su prva dva bita od $\underline{\mathbf{r}}_5$ jednaka 00). Krug što ga prikazuje slika 4.26 ima dva izlaza.




*Slika 4.26: Napredovanje niza maksimalne duljine*

Izlaz izl. 1 je isti niz **m** stvoren krugom na slici 4.22.b. Uzastopni sadržaji registra su uzastopne 4-vorke od **m**. Bitovi na izlazu izl. 2 su zbroj prvih dvaju bitova tih uzastopnih 4-vorki. Stoga na izlazu izl. 2, niz **m** započinje na mjestu #5. Ovo kašnjenje od 4 bita između dvaju izlaza neovisno je od početnoga specifičnoga stanja **t**, u kojemu započinje **m**.

Prethodna rasprava pokazala je da Svojstvo 9 predstavlja alat za generiranje napredne inačice niza maksimalne duljine. Drugi način promatranja ponašanja kruga na slici 4.26 je sljedeći. Neka $m_1$ označava niz maksimalne duljine dobiven pomakom niza maksimalne duljine **m**, ciklički u desno za *i* mjesta.

Ako je **m** niz stvoren na izl. 1, imajte na umu da je prema definiciji izl. 2 = $\mathbf{m}_0 + \mathbf{m}_1$. Prema definiciji, $\mathbf{m}_0 = \mathbf{m}$. Budući se već pokazalo da je izl. 2 = $\mathbf{m}_4$, imamo da je $\mathbf{m}_0 + \mathbf{m}_1 = \mathbf{m}_4$. Ovaj zadnji rezultat pokazuje kako se Svojstvo 9 može upotrijebiti za proizvodnju bilo kojega željenoga cikličkoga pomaka niza maksimalne duljine *m* periodičnost $2^n$–1, zbrajanjem nekih od pomaka $\mathbf{m}_0$, $\mathbf{m}_1$, …, $\mathbf{m}_{n-1}$.

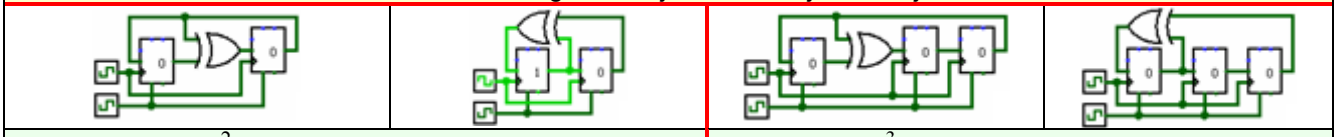
*Zaključak 4.4* Neka je **m** niz maksimalne duljine i periodičnosti $2^n$–1. Neka $\mathbf{m}_i$ označava niz dobiven cikličkim pomakom **m** u desno za *i* mjesta i neka **Y** označava matricu čiji je *i*-ti redak, *i* = 0, 1, 2, …, *n*–1, $\mathbf{m}_i$. Ako je R generatorski LFSR od **m** s ulaznim vezama, i ako $\mathbf{r}_j$ označava sadržaj R nakon *j*–1 pomaka, počevši početnim sadržajem $\mathbf{r}_1 = [1000 \ldots 0]$, onda je $\mathbf{m}_{j-1} = \mathbf{r}_j \cdot \mathbf{Y}$.

## 10. *Laboratorijska vježba 9: LFSR za generiranje nizova maksimalne duljine. Pseudo-slučajni nizovi*

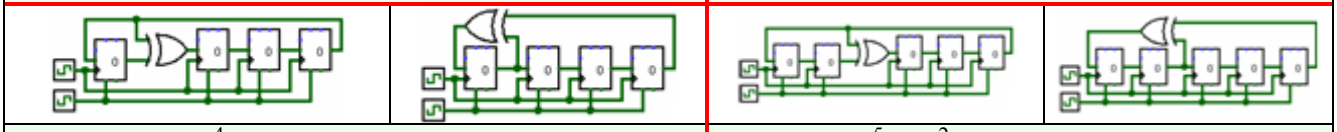
Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

| stupanj (*m*) | dužina *m* niza (*N*) | polinomi | stupanj (*m*) | dužina *m* niza (*N*) | polinomi |
|---|---|---|---|---|---|
| 1 | 1 | $x \oplus 1$ | | | |
| 2 | 3 | $x^2 \oplus x \oplus 1$ | 8 | 255 | $x^8 \oplus x^7 \oplus x^2 \oplus x \oplus 1$ |
| 3 | 7 | $x^3 \oplus x \oplus 1$ | 9 | 511 | $x^9 \oplus x^4 \oplus 1$ |
| 4 | 15 | $x^4 \oplus x \oplus 1$ | 10 | 1023 | $x^{10} \oplus x^3 \oplus 1$ |
| 5 | 31 | $x^5 \oplus x^2 \oplus 1$ | 11 | 2047 | $x^{11} \oplus x^2 \oplus 1$ |
| 6 | 63 | $x^6 \oplus x \oplus 1$ | 12 | 4095 | $x^{12} \oplus x^6 \oplus x^4 \oplus x \oplus 1$ |
| 7 | 127 | $X^7 \oplus x \oplus 1$ | 16 | 65535 | $x^{16} \oplus x^{14} \oplus x^{13} \oplus x^{11} \oplus 1$ |

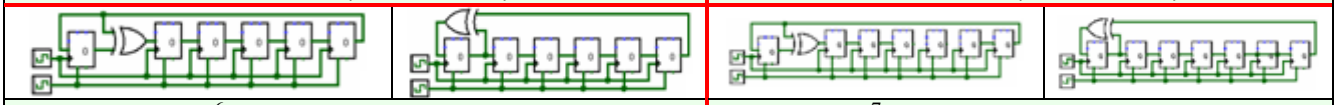
Automatsko generiranje raznih brojeva stanja



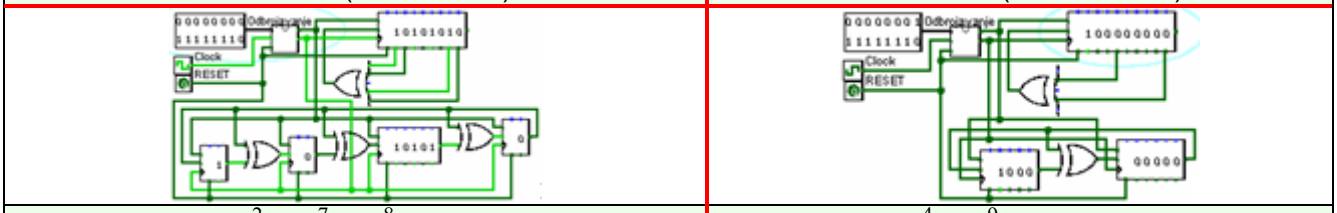
$x^2 \oplus x \oplus 1$ (M niz 3.circ) | $x^3 \oplus x \oplus 1$ (M niz 7.circ)



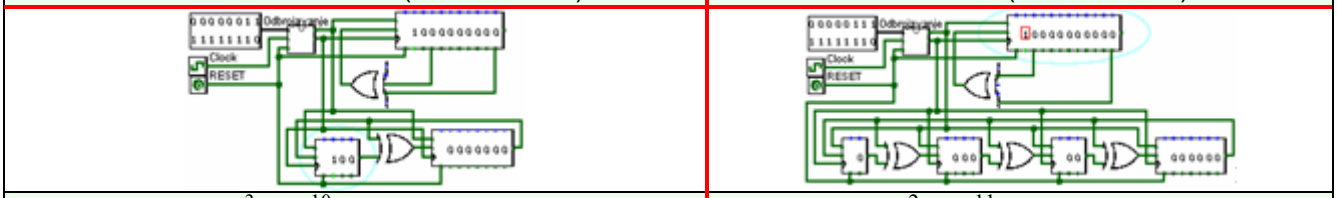
$x^4 \oplus x \oplus 1$ (M niz 15.circ) | $x^5 \oplus x^2 \oplus 1$ (M niz 31.circ)



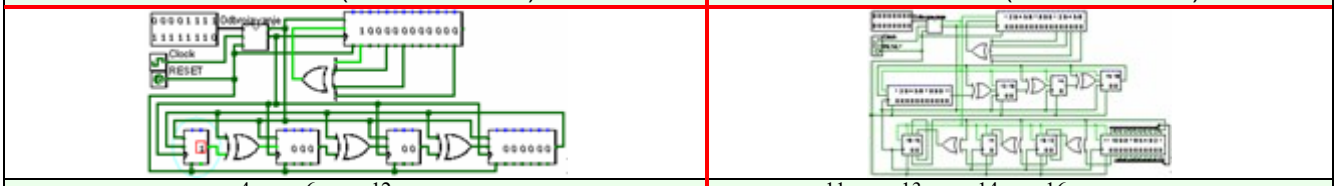
$x^6 \oplus x \oplus 1$ (M niz 63.circ) | $x^7 \oplus x \oplus 1$ (M niz 127.circ)



$1 \oplus x \oplus x^2 \oplus x^7 \oplus x^8$ (M niz 255.circ) | $1 \oplus x^4 \oplus x^9$ (M niz 511.circ)

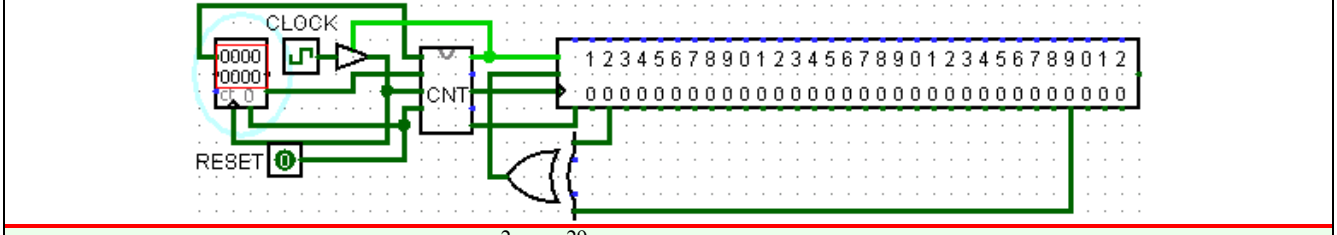


$1 \oplus x^3 \oplus x^{10}$ (M niz 1023.circ) | $1 \oplus x^2 \oplus x^{11}$ (M niz 2047.circ)



$1 \oplus x \oplus x^4 \oplus x^6 \oplus x^{12}$ (M niz 4095.circ) | $1 \oplus x^{11} \oplus x^{13} \oplus x^{14} \oplus x^{16}$ (M niz 65535.circ)

Automatsko generiranje bilo koliko stanja - 536870912 stanja (0x1FFFFFFF)




$1 \oplus x^2 \oplus x^{29}$ (M niz UNIVERSAL 1.circ)

# 5. POGLAVLJE 5 - PRASKOVITE POGREŠKE

## 5.1. 5.1. Definicija praskovite pogreške

Do sada smo razmatrali slučaj pojave jedne pogreške u primljenoj kodnoj riječi. U praksi se može pojaviti više od jedne pogreške u bloku bitova pa trebamo kodove za ispravak višestrukih pogrešaka. U ovome poglavlju bavimo se pojavom nekoliko pogrešaka što je vrlo čest slučaj u praksi.

Uzroci pogrešaka koje se stvaraju u poslanome signalu obično su posljedica vanjskih utjecaja kao npr. munje, iskre iz sustava paljenja u automobilima ili visokonaponski vodovi[64] te iščezavanje signala u atmosferi (*fading*). Sve ove vrste smetnji imaju nešto zajedničko - one kratko traju, nakon čega slijedi relativno dugo razdoblje zatišja (u odnosu na trajanje *interferencije*[65]), a onda se opet dogode. Ako se razmatra učinak ove pojave na ponašanje pogrešaka u prenošenim signalima, možemo primijetiti da se pogreške javljaju u skupinama s razmakom između takve dvije uzastopne skupine.

Svaka skupina pogrešaka odgovara pojavi kratkoga vremenskoga razdoblja zbog prethodno opisanih vrsta smetnji, a period bez pogrešaka odgovara mirnome prijenosnome kanalu. Te skupine nazivaju su *prasak pogrešaka* (*burst of errors*). Pogreške se također javljaju u skupinama riječi koje su pohranjene na *magnetskome* mediju. Oblikovat ćemo kodove koji omogućuju ispravak jednostrukih praskovitih pogrešaka koje se događaju u prijenosu ili pri pohrani kodnih riječi. Razred kodova za ispravak praskovitih pogrešaka, poznat kao Reed-Solomonovi kodovi, koristi se, primjerice, u tehnologiji kompaktnoga diska.

Poseban dio u ovome poglavlju razmatra te kodove. Osnovno načelo u teoriji informacija tvrdi, da što više znamo o ponašanju pogrešaka koje moramo ispraviti, jednostavnija je izgradnja kodova potrebnih za ispravak ovih pogrešaka. U našemu slučaju, na temelju spoznaje da pogreške dolaze u praskovima, kodovi za ispravak pogrešaka su jednostavniji od kodova opće namjene za ispravak višestrukih pogrešaka. Naši kodovi trebali bi se, međutim, koristiti, samo u onima slučajevi u kojima *znamo* da je naš kanal praskovite naravi (na temelju sustavnih mjerenja), inače su naši kodovi beskorisni.

Osnovni parametar što ga se mora razmotriti prilikom oblikovanja bilo kakvih kodova za ispravak praskovitih pogreška, jest *duljina praska* koji se ispravlja. Ova duljina naznačuje veličinu (u bitovima) područja koje omeđuje pogreške. U sljedećim primjerima (slika 5.1), **b** naznačuje ispravan, a **x** pogrešan bit:


*Slika 5.1: Dvije definicije dužine trajanja praskovitih pogrešaka*

Ovi primjeri poslije će se promijeniti. Oni su predočeni samo za prikazati srž problema. Na temelju prije navedenih primjera, već sada može se primijetiti, ako govorimo o prasku duljine *t*, nužno ne znači da je pogrešno *svih t* bitova. To samo znači da su sve pogreške *ograničena* na *uzastopnih t* mjesta gdje, na primjer, samo prvi i posljednji bitovi pogrešni. Ovdje dotičemo vrlo osjetljivu problematiku. Ako su sve pogreške ograničene na *t* uzastopnih mjesta, upravo smo definirali prasak duljine *t*.

Kao primjer promotrimo blok bitova "**bxbxbbb**", gdje *x* označava pogrešan bit. Prema onome što smo iskazali, ovdje imamo prasak duljine 3. Međutim, ovdje također imamo prasak duljine 4 budući da su dvije pogreške ograničena na prva 4 mjesta, od kojih su samo dva pogrešna, jer se nije definiralo da prasak treba započeti ili završiti pogreškom. Također, ovdje imamo prasak dužine 5, 6 ili 7. Dakle pitanje je: Kolika je dužina ovoga praska?

Slično pitanje može se postaviti ako promatramo slučaj gdje je pogrešan samo jedan bit unutar bloka. Možemo reći da imamo praska duljine 1. Međutim, također imamo praskove duljine 2, 3, 4, itd. (gdje je samo jedan bit pogrešan). Također, ponovno promotrimo blok "**bxbxbbb**". Je li ovdje prikazana

[64] *high tension wires* ... visokonaponski vodovi
[65] *interferirati* (lat. *interferre*) fiz. uzajamno djelovanje valova (električnih, svjetlosnih, zvučnih) da se djelovanje svakoga vala pojača, oslabi ili poništi;

jednostruka praskovitost? Možda postoje *dva* praska duljine 1 ili jedan prasak duljine 2 (koji zauzima prva dva bita od kojih je samo jedan pogrešan) i još jedan prasak dužine 1, 2, 3, ili 4.

Namjera prethodne rasprave nije bila zbuniti studenta, nego razbistriti raspravu. Zbunjenost možemo razjasniti navodeći sljedeće. Primimo li vektor s pogreškama, besmisleno se pitati imamo li prasak, ili kolika je duljina praska, ili koliko praskova imamo. Ključ za oblikovanje jednoga koda za ispravak praskovitih pogreška je *odlučiti unaprijed* da se želimo baviti jednostrukim praskovima duljine $t$. To znači da ćemo izgraditi kod tako da, ako se u kodnoj riječ prenesu i pogreške, a one su ograničene na $t$ uzastopnih mjesta, u mogućnosti smo ih ispraviti.

## 5.2. 5.2. Otkrivanje jednostrukih praskovitih pogrešaka

### 5.2.1. 5.2.1. OPĆA RAZMATRANJA

Neka je $\mathbf{v} = [abcdefghijkm]$ vektor čiji elementi zadovoljavaju sljedeće jednadžbe:[66]

| | razmak=4 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ | $j$ | $k$ | $m$ | $n$ | $o$ | razmak=5 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (1) | $a \oplus e \oplus i = 0$ | ⊠ | | | | ⊠ | ▭ | | | ⊠ | | ▭ | | ⊠ | | $0 = a \oplus f \oplus k$ | (1) |
| (2) | $b \oplus f \oplus j = 0$ | | ⊠ | | | ▭ | ▭ | | | | ⊠ | | ▭ | | ⊠ | $0 = b \oplus g \oplus m$ | (2) |
| (3) | $c \oplus g \oplus k = 0$ | | | ⊠ | | | | ⊠ | | ▭ | | ⊠ | | ▭ | | $0 = c \oplus h \oplus n$ | (3) |
| (4) | $d \oplus h \oplus m = 0$ | | | | ⊠ | | | | ⊠ | | | | ⊠ | | ▭ | $0 = d \oplus i \oplus o$ | (4) |

Navedene četiri jednadžbe pariteta što ih elementi vektora $\mathbf{v}$ trebaju zadovoljiti, pokazuju da se $\mathbf{v}$ može ustrojiti dodavanjem četiri paritetna bita informacijskome vektoru duljine 8. U svakoj od gornjih jednadžbi, 3 bita u vektoru $\mathbf{v}$ razdvojena su međusobno za četiri mjesta. Drugim riječima, ako se dva bita pojave u istoj jednadžbi, oni se u $\mathbf{v}$ moraju razdvojiti najmanje za pet mjesta. Pretpostavimo da su se neki bitovi (npr. $c$, $f$ i $m$) u $\mathbf{v}$ obrnuli (vrijednosti 0 i 1 zamijenili su se komplementima postojećih brojeva), gdje su svi preokrenuti bitovi *ograničeni na četiri uzastopna mjesta*. Onda imamo da će svaki preokrenut bit utjecati na različitu jednadžbu, uzrokujući da njezina desna strana ima vrijednost 1.

| | razmak=4 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ | $j$ | $k$ | $m$ | $n$ | $o$ | razmak=5 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (1) | $a \oplus e \oplus i = 0$ | ⊠ | | | | ⊠ | ▭ | | | ⊠ | | ▭ | | ⊠ | | $\mathbf{1} = a \oplus f \oplus k$ | (1) |
| (2) | $b \oplus f \oplus j = \mathbf{1}$ | | ⊠ | | | ▭ | ▭ | | | | ⊠ | | ▭ | | ⊠ | $\mathbf{1} = b \oplus g \oplus m$ | (2) |
| (3) | $c \oplus g \oplus k = \mathbf{1}$ | | | ⊠ | | | | ⊠ | | ▭ | | ⊠ | | ▭ | | $0 = c \oplus h \oplus n$ | (3) |
| (4) | $d \oplus h \oplus m = \mathbf{1}$ | | | | ⊠ | | | | ⊠ | | | | ⊠ | | ▭ | $0 = d \oplus i \oplus o$ | (4) |

Također, budući da se svaki bit u $\mathbf{v}$ pojavljuje u nekoj od jednadžbi (četiri jednadžbe pokrivaju svih 12 bitova), slijedi da je broj jednadžbi čija desna strana ima vrijednost 1, jednak broju preokrenutih bitova. Onda imamo da ako se praskovita pogreška dužine četiri ili manje uvede u $\mathbf{v}$, otkriva se pogreška. Osim toga, broj 1-elemenata u sindromu pogreške, jednak je broj pogrešnih bitova.

*Primjer* Obrnimo vrijednosti $f$, $g$ i $i$. Budući smo izvorno imali da je $b \oplus f \oplus j = 0$, $c \oplus g \oplus k = 0$, $a \oplus b \oplus i = 0$, sada imamo da je: $a \oplus e \oplus i = 1$, $b \oplus f \oplus j = 1$, $c \oplus g \oplus k = 1$. Četvrta jednadžba nije se promijenila, jer se bitovi $d$, $h$, $m$ nisu preokrenuli.

| | razmak=4 | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ | $h$ | $i$ | $j$ | $k$ | $m$ | $n$ | $o$ | razmak=5 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| (1) | $a \oplus e \oplus i = \mathbf{1}$ | ⊠ | | | | ⊠ | ▭ | | | ⊠ | | ▭ | | ⊠ | | $\mathbf{1} = a \oplus f \oplus k$ | (1) |
| (2) | $b \oplus f \oplus j = \mathbf{1}$ | | ⊠ | | | ▭ | ▭ | | | | ⊠ | | ▭ | | ⊠ | $\mathbf{1} = b \oplus g \oplus m$ | (2) |
| (3) | $c \oplus g \oplus k = \mathbf{1}$ | | | ⊠ | | | | ⊠ | | ▭ | | ⊠ | | ▭ | | $0 = c \oplus h \oplus n$ | (3) |
| (4) | $d \oplus h \oplus m = 0$ | | | | ⊠ | | | | ⊠ | | | | ⊠ | | ▭ | $0 = d \oplus i \oplus o$ | (4) |

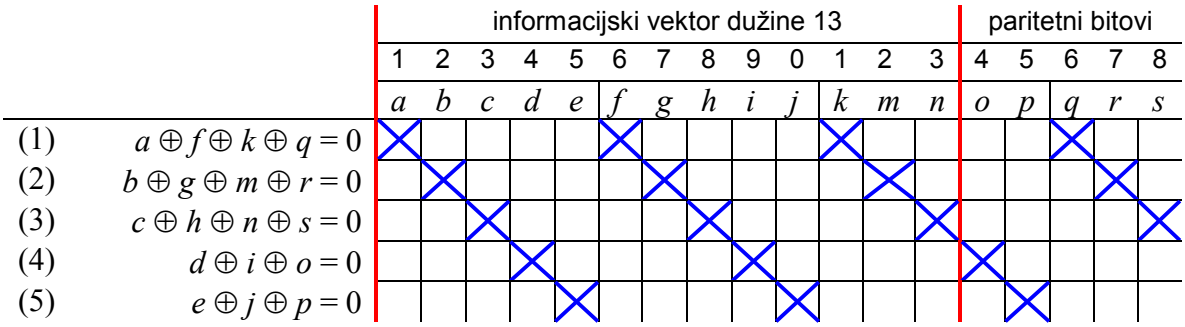
### 5.2.2. 5.2.2. PRASAK PROIZVOLJNE DULJINE

Poopćimo sada prethodnu ideju za slučaj gdje želimo otkriti prasak opće duljine $t$ koji se pojavi u primljenoj kodnoj riječi u kojoj imamo $k$ informacijskih bitova. Informacijskih $k$ bitova kodira se u

[66] Razraditi kao vježbu!

riječi duljine $k+t$ dodavanjem $t$ paritetnih bitova. Zadatak ovih paritetnih bitova je oblikovati riječi koje imaju sljedeće svojstvo: zbroj svih bitova (počevši *prvim* bitom) koji su razdvojeni $t$ mjesta, je 0.

Zbroj svih bitova koji su razdvojeni $t$ mjesta (počevši *drugim* bitom), je 0 i itd.. To se ponavlja do se ne zbroje svi bitovi razdvojeni $t$ mjesta, počevši ($t$–1)-im bitom. Ovaj zbroj jednak je 0. Na primjer, uzmimo slučaj gdje je $k$ = 13, $t$ = 5. Ovdje ćemo pretvoriti informacijski vektor duljine 13 u kodne riječi duljine 18.



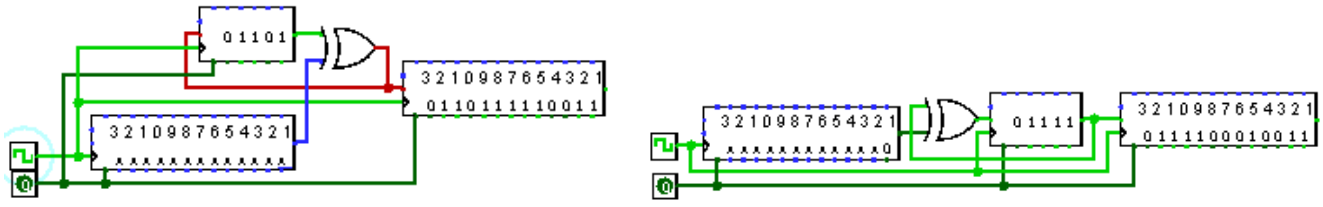

Ova kodna riječ imat će sljedeća svojstva:

1. Zbroj bitova na mjestima #1, 6, 11, 16 jednak je 0.
2. Zbroj bitova na mjestima #2, 7, 12, 17 jednak je 0.
3. Zbroj bitova na mjestima #3, 8, 13, 18 jednak je 0.
4. Zbroj bitova na mjestima #4, 9, 14 jednak je 0.
5. Zbroj bitova na mjestima #5, 10, 15 jednak je 0.

$$\begin{bmatrix} 1.....13:0\,0\,p\,0\,0 \\ 1.....13:0\,0\,0\,p\,0 \\ 1.....13:0\,0\,0\,0\,p \\ 1.....13:p\,0\,0\,0\,0 \\ 1.....13:0\,p\,0\,0\,0 \end{bmatrix}$$

### 5.2.3. 5.2.3. SUSTAVAN KOD CIKLIČKIH SVOJSTAVA

Izborom bitova 14, 15, 16, 17 i 18 kao paritetnih, kod može postati sustavan. Svaki paritetan bit pojavljuje se u različitome zbroju pa se uvijek može odabrati tako da taj zbroj iznosi 0. Imajte na umu da u prethodno navedenome primjeru, duljina kodne riječi (13) nije posljedica praska duljine 5, što je bio slučaj u prvome primjeru. Međutim, zbog prikladnosti i za praktične svrhe, pretpostavljamo od sada pa *nadalje*, da je *duljina kodne riječi posljedica duljine praska*. Također je važno napomenuti da je broj paritetnih bitova koji omogućuju otkrivanje praskovite pogreške duljine $t$, jednak $t$, gdje je *ovaj broj neovisan o duljini informacijskoga vektora*. Prethodni opisi sugeriraju kako napraviti krugove kodiranja/dekodiranja za otkriti niz uzastopnih pogrešaka duljine $t$, a prikazuje ih slika 5.2.




*Slika 5.2: Krugovi za otkrivanje praskovite pogreške*[67]

Paritetni bitovi što ih stvara koder, jednaki su zbroju informacijskih bitova razmaknutih za $t$ mjesta u informacijskome vektoru. Dekoder dodaje te bitove primljenoj poruci, razmaknute za $t$ mjesta. Njihovi sadržaji oblikuju *sindrom pogreške*.

## 5.3. 5.3. Povezanost između uzorka pogreške i cikličkih pomaka sindroma pogreške

*Definicija* **Uzorak praskovite pogreške duljine *t*** binarni je vektor duljine $t$ čiji 1-elementi odgovaraju mjestima gdje postoji pogrešan bit u prasku i gdje prvi pogrešan bit naznačuje položaj #1 u uzorku.

*Primjer* Razmotrimo (12, 8) kod, namijenjen otkrivanju pojave praskovite pogreške duljine 4 ili manje.[68] Označimo primljenu poruku **m** = [$m$, $n$, $q$, ~~$r$, $s$~~, $t$, ~~$u$~~, $v$, $w$, $x$, $y$, $z$]. Ako su $m$, $n$, $r$ pogrešni, onda

[67] Napraviti vježbu!

je praskovit uzorak [1101], jer su pogrešni bitovi na prvome, drugome i četvrtom mjestu, počevši od *m*, čiji je položaj #1. Ako su pogrešni bitovi ~~*r*, *s*, *t*~~, praskovit uzorak je ponovno [1101] počevši od *r* pa je mjesto pogrešnoga bita u prasku opet prvo, drugo i četvrto.

Ako su pogrešni *u* i *w*, onda je praskovit uzorak [1010]. Ako je jedan bit pogrešan, uzorak praska je [1000], jer prvi pogrešan bit (što je samo jedna pogreška) označava početak praska prema našoj definiciji. Imajte na umu da u kodu oblikovanome za otkrivanje/ispravak praska duljine *t* ili *manje*, uzorak praska uvijek je duljine *t* i završava ponekad nulama.

Razmotrimo slučaj u kojemu se pogreške događaju na kraju poruke i ograničene su na manje od *t* mjesta. Na primjer, *y* i *z* u poruci **m** su pogrešni. U tomu slučaju, uzorak praska je 1100, što znači da se umjetno pridružuju dodatne dvije nule kako bi uzorak imao duljinu *t*. Uvijek se držimo pravila da prvi pogrešan bit u primljenoj poruci određuje početnu točku praska. To naravno, znači, da *uzorak praska uvijek počinje jedinicom* "1". Pomakom poruke **m** = [*m*, *n*, *q*, *r*, *s*, *t*, *u*, *v*, *w*, *x*, *y*, *z*] u dekoder oblika prikazanoga na slici 5.2.b, daje sindrom pogreške **s** = [$s_0$, $s_1$, $s_2$, $s_3$], za koji vrijede:

$$s_0 = m \oplus s \oplus w$$

$$s_1 = n \oplus t \oplus x$$

$$s_2 = q \oplus u \oplus y$$

$$s_3 = r \oplus v \oplus z$$

Tablica 5.1 popisuje moguće pogrešne bitove u **m** (spadaju u prasak dužine 4), njihov odgovarajući uzorak praska i učinak pogreške na vrijednosti $s_0$, $s_1$, $s_2$, $s_3$.

*Tablica 5.1 Odnos između pogrešnih bitova u **m**, uzorak praska i sindrom generirane pogreške*

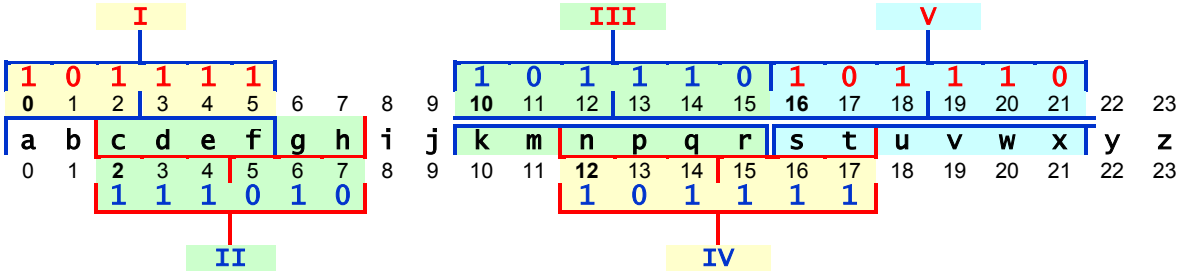
| Pogrešni bitovi u **m** | Uzorak pogreške | $s_0$ | $s_1$ | $s_2$ | $s_3$ |
|---|---|---|---|---|---|
| *m*, *n*, *q*, *r* | [1111] | 1 | 1 | 1 | 1 |
| *q*, *r*, *s*, *t* | [1111] | 1 | 1 | 1 | 1 |
| *q*, *r*, *t* | [1101] | 0 | 1 | 1 | 1 |
| *t*, *w* | [1001] | 1 | 1 | 0 | 0 |
| *u*, *x* | [1001] | 0 | 1 | 1 | 0 |
| *y* | [1000] | 0 | 0 | 1 | 0 |

Tablica pokazuje veliki problem u vezi otkrivanja pogreške praska. *Sindrom pogreške je ciklički pomak uzorka praska*. Daljnje pojedinosti u vezi ove pojave obradit će se u nastavku.

## 5.4. 5.4. Povezanost položaja pogreške praska i veličina cikličkoga pomaka između uzorka praska i sindroma pogreške

*Definicija* **Položaj pogreške praska unutar poruke** indeks je prvoga pogrešnoga bita. Prvo mjesto u primljenoj poruci ima indeks 0.

Slika 5.3 prikazuje primljenu poruku **m'** = [*abcdefghijkmnpqrstuvwxyz*] duljine 24.




Važna napomena: Jedinice naznačuju pogrešne bitove u svakoj skupini (`I, II, ...,V`), a ne vrijednost bita.

*Slika 5.3: Primljena poruka dužine 24 i nekoliko različitih praskovitih pogrešaka*

[68] Napraviti vježbu!

Primljena inačica kodne riječi **m**, je (24, 18) kod[69]. U svrhu otkrivanja pojave praskovite pogreške duljine 6 ili manje, pridružujemo šest paritetnih bitova. Razmatramo slučajeve u kojima se dobije **m'** iz **m** uvođenjem u **m** pet mogućih praskova pogreške, označeni kao I, II, III, IV, V. Položaji ovih blokova podataka naznačeni su na slici, kao i njihov uzorak.

Uzorak označen I, znači da su pogrešni bitovi *a*, *c*, *d*, *e*, *f*. Praskoviti uzorci I i IV su isti, kao što su III i V, ali se javljaju na različitim mjestima. Samo jedan od ovih praskova smije se pojaviti u određenome trenutku. Svrha crtanja njih pet na jednoj slici, jest odvojeno analizirati učinak svakoga od njih. Tablica 5.2 popisuje odnose među praskovitim uzorcima, njihov položaj u primljenoj poruci te nastale sindrome.

*Tablica 5.2 Odnos između uzoraka praskova, njihov položaj i sindromi stvorene pogreške*

| Broj praska | Uzorak | Veličina posmika | Položaj praska | Sindrom pogreške | = |
|---|---|---|---|---|---|
| I | 101111 | 0%6 = 0 | 0 | 101111 | • |
| II | 111010 | 2%6 = 2 | 2 | 101110 | |
| III | 101110 | 10%6 = 4 | 10 | 111010 | |
| IV | 101111 | 12%6 = 0 | 12 | 101111 | • |
| V | 101110 | 16%6 = 4 | 16 | 111010 | |

Ovdje opet uočavamo činjenicu koja se već prikazala u tablici 5.1. Sindrom je ciklički pomak uzorka praska, uključujući slučaj u kojemu je posmik 0, što znači da je sindrom jednak praskovitome uzorku. Za prasak I, ciklički pomak je 0. Za prasak II ciklički pomak je dva mjesta u lijevo, što znači da se sindrom mora ciklički pomaknuti dva mjesta u lijevo kako bi se poklopio s uzorkom praska. Za prasak III moramo ciklički pomaknuti sindrom pogreške četiri mjesta u lijevo kako bi se dobio uzorak praska. Za prasak IV ciklički pomak je nula, a za prasak V, ciklički pomak je četiri. Ove brojeve daje nam izračun u tablici 5.3.

*Tablica 5.3 Sažetak odnosa mjesta praska i broja posmika sindroma pogreške*

| Praskovit uzorak | I | II | III | IV | V |
|---|---|---|---|---|---|
| Položaj praska | 0 | 2 | 10 | 12 | 16 |
| Izravan odnos | 0%6 = 0 | 2%6 = 2 | 10%6 = 4 | 12%6 = 0 | 16%6 = 4 |
| Broj posmika | 0 | 2 | 4 | 0 | 4 |

U tablici 5.3 sažimamo mjesto početka svakoga od pet praskova kao i koliki pomak u lijevo treba napraviti sindrom pogreške da bi se podudarao s uzorkom praska.

Postoji izravan odnos između brojeva u *drugome* i *trećemu* retku tablice 5.3. Brojevi u trećemu retku dobiju se diobom 6 i uzimanjem ostatka odgovarajućih brojeva iz drugoga retka. Nakon ove operacije, 0 u drugome retku daje ostatak 0 u četvrtome retku, 2 daje ostatak 2 (nakon diobe 2/6 ostatak je 2), za 10 dobije se ostatak 4, za 12 je ostatak 0, a 16 daje ostatak 4. Algoritam izračuna dat će se poslije.

*Označavanje* Ostatak dobiven nakon diobe broja *m* brojem *n*, obilježava se kao ***m*** mod ***n.***

*Primjer* Dijeljenjem 16 sa 6 daje 4 kao ostatak. Korištenjem gornje oznake kažemo da je 16 mod 6 = 4 ili 16 je kongruentno 4 modulo 6, a piše se: 16 ≡ 4 (mod 6). Znak "≡" čita se "kongruentno"[70]

Još primjera: 100 mod 9 = 1; 20 mod 5 = 0; 27 mod 7 = 6; 38 mod 8 = 6.

*Zaključak 5.1* Neka je **c** kod za otkrivanje pogrešaka praska. Neka *t* označava duljinu praska kojega treba otkriti i neka je duljina kodne riječi djeljiva s *t*. Neka *x* označava položaj pogreške praska unutar primljene poruke. Ako **p** označava uzorak praska, a **s** označava sindrom pogreške što proizlazi iz praska, onda se **p** dobije cikličkim pomakom u lijevo za *x* mod *t* mjesta.

---

[69] Napraviti vježbu!

[70] *kongruencija* … (lat. *congruere*) podudarati se; 1. skladnost, suglasnost, sukladnost, istovjetnost, podudaranje i u obliku i u veličini; podudarnost; 2. *gram.* sročnost; slaganje rečeničkih dijelova; *kongruentan*, -tna, -tno - suglasan, sročan, sukladan, istovjetan, podudaran (i u obliku i u veličini), *kongruirati*, -gruiram - biti u kongruenciji, slagati se, sricati se

Još jednom objasnit ćemo Zaključak 5.1 razmatranjem slučaja $n = 100$, $t = 10$. Neka prasak (duljine 10) započne u poruci (duljine 100) na mjestu #87 ($x \equiv$ #87). Ako je **p** uzorak ovoga praska, a **s** je dobiven sindrom pogreške (**p** i **s** su dugački 10 bitova), onda se **p** dobije cikličkim pomakom za **s** mjesta u lijevo 7 puta (87 mod 10 = 7).

### *5.5. 5.5. Uvjeti pod kojima je moguće otkriti točan praskovit uzorak*

U tablici 5.4 navedeni su svi mogući sindromi pogrešaka koji odgovaraju trima posebnima uzorcima praskova duljine 5.

*Tablica 5.4 Neki uzorci praskova dužine 5 u kojima su pogreške ograničene na prva tri mjesta i svi mogući odgovarajući sindromi pogrešaka*

| uzorak praska | moguć sindrom pogreške |
|---|---|
| 10100 | 10100 |
| ↓ | 01010 |
| ↓ | 00101 |
| ↓ | 10010 |
| ↓ | 01001 |
| | |
| 11000 | 11000 |
| ↓ | 01100 |
| ↓ | 00110 |
| ↓ | 00011 |
| ↓ | 10001 |
| | |
| 10000 | 10000 |
| ↓ | 01000 |
| ↓ | 00100 |
| ↓ | 00010 |
| ↓ | 00001 |

Svojstvo ovih uzoraka jest činjenica da su pogreške ograničene na prva 3 mjesta. Mogući sindromi pogrešaka su svi mogući ciklički pomaci praskovitoga uzorka. Napomenimo sada vrlo važnu činjenicu[71]. *Od pet cikličkih pomaka uzorka, postoji samo jedan uzorak što započinje jednom* 1, *a završava s dvije* 0. Ovo je istinito za sve slučajeve i može se dokazati. *Ovaj posmik je posmik* #0 tj., *on sam je uzorak*. To znači da je moguće odrediti točan oblik uzorka praska **p** *duljine* 3, pomoću koda namijenjenoga otkrivanju praska *duljine 5*. To se radi cikličkim pomakom sindroma pogreške (duljine 5) dok sindrom pogreške ne započne jedinicom i ne završi s dvije nule.

Takvo mjesto jedinstveno je i njegova prva tri bita jednaka su uzorku praska **p**. Na temelju Zaključka 5.1, ako duljina poruke nije veća od 5, a $x$ je mjesto prvoga pogrešnoga bita u primljenoj poruci, onda se postupak određivanja uzorka praska **p** također dobije kao $x$ mod 5. Jednostavno izračunamo koliko puta moramo napraviti ciklički pomak sindroma u lijevo dok se ne dobije jedinica na početku i dvije 0 na kraju. Ovaj rezultat može se poopćiti kako slijedi:

*Zaključak 5.2* Neka je $C$ kod za otkrivanje pogrešaka praska. Neka $u$ označavaju duljinu praska koji se otkriva i neka je duljina kodne riječi djeljiva s $u$. Ako duljina praska koji se stvarno pojavljuje u odaslanoj poruci ne prelazi $[(u+1)/2]$, moguće je otkriti točan uzorak praska. Također, moguće je izračunati $x$ mod $u$, gdje $x$ označava položaj praska.

Da bi se ispravile pogreške i poznajući uzorak praska, jedina stvar koju moramo znati je $x$. Uvjeti navedeni u Zaključku 5.2 omogućuju nam poznavanje $x$ mod $u$. U nastavku se prikazuje što još trebamo, kako bi u potpunosti odredili $x$.

[71] VIF ... vrlo važna činjenica

# 11. *Laboratorijska vježba 10: Otkrivanje i ispravak praskovite pogreške*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

5.2. Otkrivanje jednostrukih praskovitih pogrešaka

5.2.1. Opća razmatranja

## 5.6.1. 5.6.1. KINESKI TEOREM O OSTATKU I NJEGOVA PRIMJENA U ISPRAVCIMA PRASKOVITIH POGREŠAKA

### 5.6.1.1. 5.6.1.1. Temelji modulo aritmetike

Rimski car Julije Cezar u komunikaciji sa svojim prijateljima koristio se tajnopisom (šifriranjem[72]). U njemu bi se slova otvorenoga teksta zamijenila slovima koja su se u abecedi nalazila 3 mjesta dalje od njih:

$$A \Rightarrow D,\ B \Rightarrow E,\ C \Rightarrow F,\ \ldots.$$

U primjerima, koristit ćemo međunarodnu abecedu od 26 slova. Cezarovim tajnopisom nazivaju i se tajnopisi istoga oblika, pomaka različitoga od 3. Za točno definirati Cezarov tajnopis, uvodi se prirodan preslik *jedan na jedan* (bijekcija) između slova abecede (A ÷ Z) i cijelih brojeva (0 ÷ 25). Skup $Z_{26}$ označit ćemo kao:

$$Z_{26} = \{0, 1, 2, \ldots, 25\}$$

i pretpostavit ćemo da su na njemu definirane operacije *zbrajanja*, *oduzimanja* i *množenja* na isti način kao i na skupu cijelih brojeva, ali tako da se rezultat, ukoliko je izvan skupa

$$Z_{26} = \{0, 1, 2, \ldots, 25\},$$

na kraju zamijeni njegovim *ostatkom pri dijeljenju* s 26. Za te namjene, koristit ćemo oznake:

*zbrajanje* ($+_{26}$)

$$a +_{26} b \text{ ili } (a + b) \bmod 26,$$

*oduzimanje* ($-_{26}$)

$$a -_{26} b \text{ ili } (a - b) \bmod 26,$$

*množenje* ($\cdot_{26}$).

$$a \cdot_{26} b \text{ ili } (a \cdot b) \bmod 26.$$

Npr.:

$$(10 + 20) \bmod 26 = +30 - 26 = 4,$$

$$(10 - 20) \bmod 26 = -10 + 26 = 16.$$

Skup $Z_{26}$, uz operacije $+_{26}$ i $\cdot_{26}$, zadovoljava aksiome matematičke strukture koja se zove *prsten*. To znači da su operacije *zbrajanja* i *množenja*

- zatvorene u $Z_{26}$,

  rezultat je ponovo iz $Z_{26}$,

- komutacijske,

$$a +_{26} b = b +_{26} a,$$

$$a \cdot_{26} b = b \cdot_{26} a, \text{ i}$$

- asocijacijske,

$$(a +_{26} b) +_{26} c = a +_{26} (b +_{26} c),$$

$$(a \cdot_{26} b) \cdot_{26} c = a \cdot_{26} (b \cdot_{26} c).$$

Vrijedi razdioba množenja u odnosu na zbrajanje

$$(a +_{26} b) \cdot_{26} c = (a \cdot_{26} c) +_{26} (b \cdot_{26} c).$$

[72] *šifrirati* … (fr. *chiffrer*) 1. Pisati tajnim pismom; 2. trg. cijenu robe označiti tajnim znacima umjesto običnim brojevima koje svatko razumije.

Broj 0 je *neutralan element* za ***zbrajanje***

$$a +_{26} 0 = 0 +_{26} a = a$$

te svaki element $a$ ima suprotan element (*adicijska recipročna vrijednost*) $-a$.

Za $a \neq 0$ to je broj $26-a$, jer vrijedi:

$$a +_{26} (26 - a) = (26 - a) +_{26} a = 26 \bmod 26 = 0.$$

Nadalje, broj 1 je *neutralan element* za ***množenje***

$$a \cdot_{26} 1 = 1 \cdot_{26} a = a,$$

no *samo neki* elementi $a$ imaju *multiplikacijski suprotan element* $a^{-1}$, tj. element za koji vrijedi:

$$a \cdot_{26} a^{-1} = a^{-1} \cdot_{26} a = 1.$$

Uvedimo još jednu oznaku:

Ako dva cijela broja $a$ i $b$ daju isti ostatak pri dijeljenju s 26, to ćemo zapisati kao:

$$a \equiv b \pmod{26}$$

i izgovarati kao:

"$a$ i $b$ su *kongruentni* modulo 26".

Za proizvoljan prirodan broj $m$, slično se definiraju: skup $Z_m$ i operacije na njemu. Primijetimo da zamjena slova brojevima još nije nužna. Mogli smo sve definirati izrazima slova i njihovim pomicanjem unutar abecede te objasniti što se događa ako se preskoči zadnje slovo.

*Cezarov tajnopis* definira se na sljedeći način:[73] Neka je $\mathcal{P} = C = \mathcal{K} = Z_{26}$. Za $0 \leq K \leq 25$ definiramo

$$e_K(x) = (x + K) \bmod 26,\ d_K(y) = (y - K) \bmod 26.$$

Kao što smo već objasnili, tajnopis se definirao na $Z_{26}$ budući da koristimo 26 slova pa imamo sljedeći preslik, koji za svako slovo abecede daje njegovu "brojčanu jednakost":

| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |

U Cezarovome tajnopisu (šifri), osnovni elementi (simboli) otvorenoga teksta su slova odnosno njihove brojčane istoznačnice (*ekvivalenti*), a ključ $K$ određuje koliko ćemo mjesta (u desno/lijevo) pomicati slova pri tajnopisu. Očito je $d_K(e_K(x)) = x$, kao što se zahtijeva u definiciji kripto-sustava[74]. Za $K = 3$ dobije se izvoran Cezarov tajnopis.

*5.6.1.2.* *5.6.1.2. Primjer 5.1. Otkrivanje (dekriptiranje) značenja riječi PWNUYTLWFKNOF dobivene Cezarovim tajnopisom.*

***Rješenje*:**

Budući da je prostor ključeva jako malen (ima ih 26) zadatak možemo riješiti "grubom silom", tj. tako da ispitamo sve moguće ključeve, sve dok ne dođemo do nekoga smislenoga teksta. Za $d_0$, $d_1$, $d_2$, … dobivamo redom:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P | W | N | U | Y | T | L | W | F | K | N | O | F |
| O | V | M | T | X | S | K | V | E | J | M | N | E |
| N | U | L | S | W | R | J | U | D | I | L | M | D |
| M | T | K | R | V | Q | I | T | C | H | K | L | C |
| L | S | J | Q | U | P | H | S | B | G | J | K | B |
| K | R | I | P | T | O | G | R | A | F | I | J | A |

Dakle, ključ je $K = 5$, a skriven tekst je KRIPTOGRAFIJA.

Da bismo dobili barem malo sigurniji tajnopis, možemo promatrati funkcije za tajnopis koje će uključivati više od jednoga parametra. Najjednostavnija takva funkcija je *afina funkcija pravca*, $e(x) =$

[73] *Kripto-sustav* je uređena petorka $(\mathcal{P}, C, \mathcal{K}, \mathcal{E}, \mathcal{D})$.

[74] *kripto* … (grč. *krypto*, *kryptos*) skrivam, krijem predmetak u složenicama sa značenjem: skriven, tajan

$ax + b$[75]. No, tu se pojavljuje nov problem, jer takva funkcija na skupu $Z_{26}$ ne mora imati *suprotan element* (nije nužno injekcijski preslik[76]). Zato parametar $a$ nije proizvoljan, već je *relativno prost* broj uz modulo 26.

#### 5.6.1.3. 5.6.1.3. Najveća zajednička mjera

***Najveća zajednička mjera*** GCD (*greatest common divisor*)[77], dvaju prirodnih brojeva $n$ i $m$ najveći je broj $j$, takav da su $n$ i $m$ višekratnici od $j$. Euklid je predložio jednostavan algoritam za računanje GCD($n$, $m$), gdje je $n > m$, a on se temelji na konceptu poznatome kao *kineski teorem o ostatku*. Osnovna ideja algoritma je da se više puta izvede **modulo** računanje uzastopnih parova niza koji počinje ($n$, $m$, …), dok se ne dosegne nula. Zadnji ne nulti broj u ovome nizu između $n$ i $m$ je GCD. Na primjer, za $n$ = 80844 i $m$ = 25320, niz je kako slijedi:

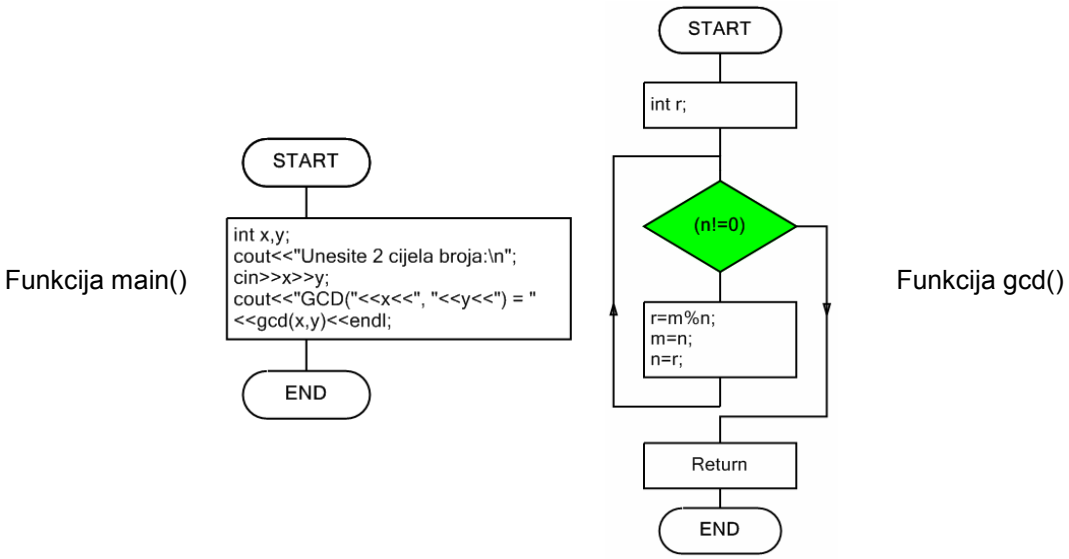
```
80844 mod 25320 = 4884 = 80844 - (⌊80844/25320⌋ · 25320)[78]
25320 mod 4884 = 900 = 25320 - (⌊25320/4884⌋ · 4884)
4884 mod 900 = 384 = 4884 - (⌊4884/900⌋ · 900)
900 mod 384 = 132 = 900 - (⌊900/384⌋ · 384)
384 mod 132 = 120 = 384 - (⌊384/132⌋ · 132)
132 mod 120 = 12 ⇒ GCD = 132 - (⌊132/120⌋ · 120)
120 mod 12 = 0
```

Dakle, GCD za brojeve 80844 i 25320 je **12**.

Iskaz Euklidova algoritma korištenjem primjera 1599 i 650:

```
1599 = 650*2 + 299
650 = 299*2 + 52
299 = 52*5 + 39
52 = 39*1 + 13
39 = 13*3 + 0
```

Slika 5.4 prikazuje dijagram toka algoritma za izračun najvećega zajedničkoga djelitelja GCD (*greatest common divisor*) dvaju brojeva $a$ i $b$ na mjestima A i B.




*Slika 5.4: Dijagram toka algoritma (Euclid's algorithm) za izračun najvećega zajedničkoga djelitelja GCD (greatest common divisor) dvaju brojeva a i b na mjestima nazvanima A i B.*

Algoritam se ostvaruje uzastopnim oduzimanjem u dvije petlje: IF provjera, B ≥ A daje "da" (ili istinu) (točnije *broj b* na mjestu B je veći ili jednak broju *a* na mjestu A) THEN algoritam pridružuje B ← *b*–

---

[75] *Afina* funkcija - funkcija $F(x) = ax + b$, gdje su $a$, $b$ realni brojevi. Naziva se i linearnom funkcijom. Naziv je 1748. god. uveo Euler prema latinskoj riječi *affinitas*, što znači srodstvo po mužu ili ženi.

[76] Vidi poglavlje "10.4.3. Injekcija, surjekcija, bijekcija"

[77] Vidi LCM

[78] Par oznaka "$\lfloor a/b \rfloor$", znače cjelobrojnu vrijednost diobe dvaju brojeva.

*a* (što znači da broj *b* zamjenjuje staru vrijednost *b*). Slično tome, IF A > B, THEN (pridružuje) A ← *a*–*b*. Postupak završava ako je (sadržaj) B jednak 0, čime se određuje GDC u A.

*12. Najveći zajednički djelitelj GCD*

```
#include <iostream>
using namespace std;
int gcd(int m,int n){
int r; // obznana ostatka
while(n!=0){
r=m%n;
m=n;
n=r;
}
return m;
}

int main(){
int x,y;
cout<<"Unesite 2 cijela broja:\n";
cin>>x>>y;
cout<<"GCD("<<x<<", "<<y<<") = "
<<gcd(x,y)<<endl;
}
```

```
C:\windows\system32\cmd.exe
Unesite 2 cijela broja:
8 3
GCD(8, 3) = 1
Press any key to continue . . .
```

#### 5.6.1.4. 5.6.1.4. Kineski teorem o ostatku

Neka su *p* i *q* dva ***relativno*** *prosta broja* (znači da je najveći broj kojime su oba djeljiva jednak 1)[79]. Ako je *x* ograničen na raspon između 0 i *pq*–1, onda, znajući vrijednosti *x* mod *p* i *x* mod *q*, moguće je jedinstveno odrediti *x*.

Teorem navodi i postupak za određivanje *x* iz *x* mod *p* i *x* mod *q*.[80] Također treba napomenuti da teorem ima općenitiji oblik. Zbog jednostavnosti, ovdje smo prihvatili poseban slučaj.

*Primjer* Brojevi 3 i 8 su relativno prosti. Obzirom da su *x* mod 3 = 2 i *x* mod 8 = 4, jedinstveno rješenje za *x* je, *x* = 20.

$x\%3 = 2 \Rightarrow x \equiv 2 \pmod 3 \Rightarrow x \bmod 3 = 2 \Rightarrow x - 2 = y \cdot 3$ (znak '≡' čita se kongruentno)

$x\%8 = 4 \Rightarrow x \equiv 4 \pmod 8 \Rightarrow x \bmod 8 = 4 \Rightarrow x - 4 = z \cdot 8$

$$3y + 2 = 8z + 4$$
$$3y = 8z + 2$$
$$y = \frac{8z+2}{3}$$

Postupkom *ustrajnoga ponavljanja* (*iteration*) traži se rješenje kao višekratnik nazivnika:

Za $z = 0 \Rightarrow y = \frac{8 \cdot 0 + 2}{3} = \frac{2}{3} \Rightarrow$ (nije rješenje)

[79] Dva broja su "relativno prosta", ako nemaju zajedničkih faktora osim 1 (drugim riječima, ne možete ih ravnomjerno podijeliti nekom zajedničkom vrijednosti). Primjer: 7 i 20 su *relativno* prosti (nemaju zajednički faktor), ali 6 i 20 nisu relativno prosti, jer svakoga od njih možete podijeliti s 2 (2 je zajednički faktor).

[80] CRT ... Kineski teorem o ostatcima (*Chinese Remainder Theorem*)

Za $z=1 \Rightarrow y=\frac{8\cdot 1+2}{3}=\frac{10}{3} \Rightarrow$ (nije rješenje)

Za $z=2 \Rightarrow y=\frac{8\cdot 2+2}{3}=\frac{18}{3}=6 \Rightarrow$ (je rješenje!) $\Rightarrow y=6$

$a*y+b = c*z+d$

$3y+2=8z+4$

$8z+4=3y+2 \Rightarrow 8z=3y-2$

$z=\frac{3y-2}{8}$

Za $y=0 \Rightarrow z=\frac{3y-2}{8}=\frac{3\cdot 0-2}{8}=\frac{-2}{8} \Rightarrow$ (nije rješenje)

Za $y=1 \Rightarrow z=\frac{3\cdot 1-2}{8}=\frac{1}{3} \Rightarrow$ (nije rješenje)

Za $y=2 \Rightarrow z=\frac{3\cdot 2-2}{8}=\frac{4}{8} \Rightarrow$ (nije rješenje)

Za $y=3 \Rightarrow z=\frac{3\cdot 3-2}{8}=\frac{7}{8} \Rightarrow$ (nije rješenje)

Za $y=4 \Rightarrow z=\frac{3\cdot 4-2}{8}=\frac{10}{8} \Rightarrow$ (nije rješenje)

Za $y=5 \Rightarrow z=\frac{3\cdot 5-2}{8}=\frac{13}{8} \Rightarrow$ (nije rješenje)

Za $y=6 \Rightarrow z=\frac{3\cdot 6-2}{8}=\frac{16}{8}=2 \Rightarrow$ (je rješenje!) $z=2$

$x-2=y\cdot 3 \Rightarrow x=3y+2=3\cdot 6+2=18+2=20$

$x-4=z\cdot 8 \Rightarrow x=8z+4=8\cdot 2+4=16+4=20$

$\Rightarrow x=20$

Programsko rješenje:

## 13. *Modulo matematika za 2 kongruentna izraza*

```
//1. Ispitati SVE raspoložive primjere!
//2. napraviti funkciju za proizvoljan broj
jednadžbi
//3. ispitati jesu li odabrani brojevi (a i
c) relativno prosti brojevi (RPB)
//4. napraviti funkciju za pisanje u jednoj
crti
#include <iostream>
using namespace std;
//a*y+b=c*z+d
int rpb(int,int,int);
int upis();
int gcd(int,int);
int izracun(int,int,int,int);
int a,b,c,d,x; //x%a=b x%c=d

int main(){
int y,z;
upis();
gcd(a,c);
int i(0);
//i=rpb(a,c,i);
//if(i==1)return(0);
```

```
C:\windows\system32\cmd.exe
x % 3 ostatak je: 2
x % 8 ostatak je: 4
x=20
Press any key to continue . . .
```

*13. Modulo matematika za 2 kongruentna izraza*

```
y=izracun(a,b,c,d);
//cout<<y<<endl;
z=izracun(c,d,a,b);
//cout<<z<<endl;
if(a*y+b==c*z+d)cout<<"x="<<a*y+b<<endl;
}

int upis(){
cout<<"x % ";
cin>>a;
system("cls");
cout<<"x % "<<a<<" ostatak je: ";
cin>>b;
cout<<"x % ";
cin>>c;
system("cls");
cout<<"x % "<<a<<" ostatak je: "<<b<<endl;
cout<<"x % "<<c<<" ostatak je: ";
cin>>d;
return 0;
}

int izracun(int e,int f,int g,int h){
int u(0);
float v;
for(u=0;(g*u+h-f)%e!=0;u++){
//a*y+b=c*z+d
v=(g*u+h-f)/e;
}
//cout<<"g*u+h-f="<<(g*u+h-f)%e<<"
v="<<(g*u+h-f)/e<<endl;
v=(g*u+h-f)/e;
return (v);
}

int rpb(int x,int y, int i){
if(x%y!=0&&y%x!=0){
cout<<x<<" i "<<y<<" nisu djeljivi!\n";
return(1);
}
return(0);
}

int gcd(int m, int n){
int r; // obznana ostatka
while(n!=0){
r = m % n;
m = n;
n = r;
}
return m;
}
```

*Nekoliko primjera*: Neka su: $x$ mod 5 = 1 i $x$ mod 3 = 0. Rješenje je $x = 6$.

$$y_1 = x\%5 = 1 \qquad y_1 = (x : 5) + 1$$
$$y_2 = x\%3 = 0 \qquad y_2 = (x : 3) + 0$$

Neka su: $x$ mod 5 = 2 i $x$ mod 7 = 5. Rješenje je $x = 12$.

$$y_1 = x\%5 = 2 \qquad y_1 = (x : 5) + 2$$
$$y_2 = x\%7 = 5 \qquad y_2 = (x : 7) + 5$$

Neka su: $x$ mod 5 = 2 i $x$ mod 7 = 4. Rješenje je $x = 32$.

$$y_1 = x\%5 = 2 \qquad y_1 = (x : 5) + 2$$
$$y_2 = x\%7 = 4 \qquad y_2 = (x : 7) + 4$$

Zaključak 5.2 pokazuje nam kako možemo dobiti $x$ mod $u$ unutar primljene poruke, gdje je $x$ mjesto praska duljine $t$, za $u = 2t–1$.

Obzirom na kineski teorem ostatka CRT (*Chinese Remainder Theorem*), potpuno određivanje $x$ moguće je ako je poznata druga vrijednost oblika $x$ mod $v$. Ako su $u$ i $v$ relativno prosti (najveći broj kojime su oba djeljiva jednak je 1), onda jedinstveno možemo riješiti $x$, sve dok duljina poruke ne prelazi $u \cdot v$. Kako je indeks prvoga mjesta u poruci jednak #0, dužina $u \cdot v$ osigurava da niti jedan indeks položaja ne premašuje vrijednost $(u \cdot v)–1$, čime se osigurava da $x$ ima jedinstveno rješenje. Da bi imali još jednu vrijednost oblika $x$ mod $v$, generiramo kodnu riječ koja će zadovoljiti daljnji skup paritetnih jednadžbi. Za omogućiti otkrivanje praska pogrešaka duljine $u$, znači da je zbroj svih bitova u kodnoj riječ, koja je razmaknuta za $u$ mjesta, počevši bitovima #0, 1, …, $u$–1, jednak 0. Ovaj skup nadalje pokazuje da *je zbroj svih bitova koji su razmaknuti za v mjesta*, *počevši bitovima* #0, 1, …, $v$–1, *jednak* 0. U praksi, vrijednost $v$ odabire se da tako da bude sam $t$ (gdje je $u = 2t–1$).

*Primjer* za $u = 5$ i $v = 3$ zahtijevamo da bitovi kodne riječi [*abcdefghijkmnpq*] zadovoljavaju sljedeće jednadžbe:

$$a \oplus f \oplus k = 0$$
$$b \oplus g \oplus m = 0$$
$$c \oplus h \oplus n = 0$$
$$d \oplus i \oplus p = 0$$
$$e \oplus j \oplus q = 0$$

$$x\%3 = (y \cdot 3) + 2$$
$$x\%8 = (z \cdot 8) + 4$$


kodna riječ također treba zadovoljiti jednadžbe:

$$a \oplus d \oplus g \oplus j \oplus n = 0$$
$$b \oplus e \oplus h \oplus k \oplus p = 0$$
$$c \oplus f \oplus i \oplus m \oplus q = 0$$

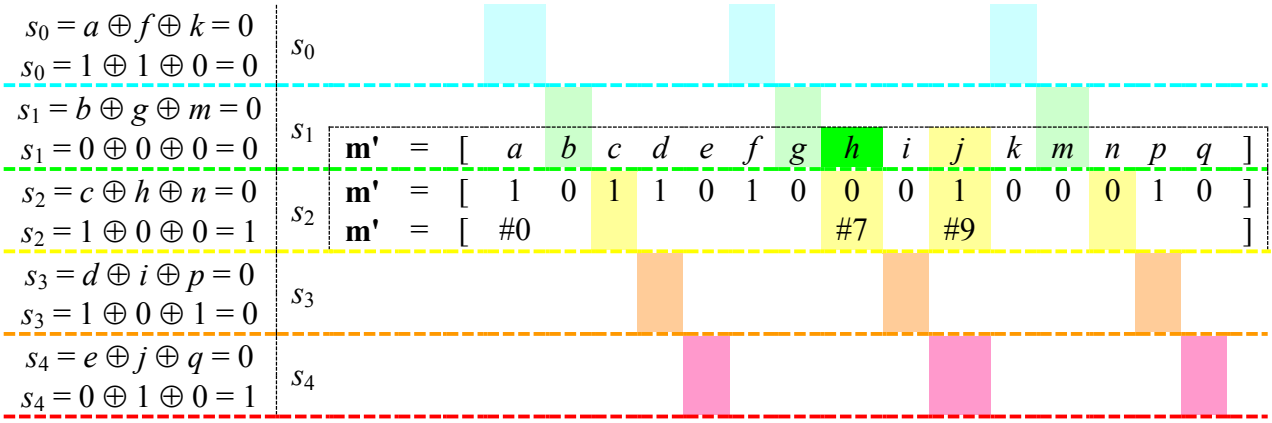
## 12. *Laboratorijska vježba 11: BCH kod*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

[81] G od *Galois*

### 5.7.1. 5.7.1 CJELOVIT POSTUPAK ISPRAVKE POGREŠAKA

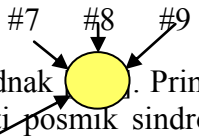
Sada pokazujemo cjelovit postupak ispravke pogrešaka.[82] Zbog lakšega objašnjenja oslonit ćemo se na sljedeću raspravu o točno određenome slučaju gdje je $u = 5$ i $t = 3$. Neka je primljena poruka **m'** = [101101000100010]. Njena poslana inačica **m** zadovoljava prethodno navedenih osam jednadžbi. Zatim se u **m** uvodi praskovita pogreška duljine ne veće od 3, oblikujući **m'**. Neka $x$ označava položaj praskovite pogreške u **m'**. Na temelju Zaključka 5.2, iz sindroma pogreške možemo odrediti uzorak praska, kao $x$ mod 5. Prijemnik započinje dekodiranje izračunom sindroma **s** = $[s_0, s_1, s_2, s_3, s_4]$. Bitovi sindroma dobiju se zbrajanjem tih bitova iz **m'** koji su razdvojeni 5 mjesta:




Prijemnik sada mora uzeti sindrom pogreške [00101] i napraviti ciklički pomak u lijevo dok se na početku uzorka ne pojavi **1**, a uzorak ne završi s dvije nule (**00**)[83]. Ovo se dobije pomakom sindroma ciklički za dva mjesta u lijevo, a rezultat je uzorak [10100]. Praskovit uzorak je 101, a $x$ mod 5 = 2. Sada računamo sindrom **z** = $[z_0, z_1, z_2]$. Bitovi sindroma dobiju se zbrajanjem tih bitova iz **m'** koji su razmaknuti za 3 mjesta:

$z_0 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$
$z_1 = 0 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 1$
$z_2 = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$

**m'** = [1 0 1 1 0 1 0 0 0 1 0 0 0 1 0]
**m'** = [*a b c d e f g h i j k m n p q*]
**m'** = [#0 #7 #9 ]




Iz gornjega rezultata poznato je da je praskovit uzorak jednak [ ]. Primijenimo sada Zaključak 5.1. Prema ovome zaključku, iznos za koji moramo napraviti posmik sindroma [110] u lijevo, kako bi dobili uzorak [101] jednak je $x$ mod 3. Pomak [110] ciklički jednom u lijevo daje [101], što znači da je $x$ mod 3 = 1.

Dva sindroma pogreške sada imaju 2 jednadžbe: $x$ mod 5 = 2 i $x$ mod 3 = 1. Jedinstveno rješenje za $x$ je [ ]. Imamo, dakle, određen $x$ iz dva sindroma pogrešaka, gdje se praskovit uzorak već odredio iz prvoga sindroma pogreške.

Poznajući [ ] možemo ispraviti pogreške. One su na mjestima 7 i 9 u **m'** (gdje brojanje započinje s lijeve strane i prvi bit je #0).[84] Za prikazanu tehniku ispravke praska duljine 3 može se općenito ustvrditi: prasak duljine $t$ može se ispraviti unutar kodne riječi duljine $u \cdot t$ gdje je $u = 2t–1$. To je moguće ako kodna riječ ima strukturu koja omogućuje otkrivanje neovisnih praskova duljine $u$ i $t$.

Otkrivanje praska duljine $u$ (u kojemu su pogreške ograničena na $t$ mjesta) omogućuje otkrivanje praskovitoga uzorka i određivanje $x$ mod $u$, gdje je $x$ položaj praska. To se postiže pomoću Zaključka 5.2. Također smo primijenili nezavisnu shemu namijenjenu otkrivanju praska duljine $t$. Budući da već znamo praskovit uzorak, primjenjujemo Zaključak 5.1 i određujemo $x$ mod $t$. Iz $x$ mod $u$ i $x$ mod $t$, možemo odrediti $x$. Određivanjem položaja praskovitoga uzorka, omogućuje nam potpun ispravak.

[82] C++ zadatak!!!!!
[83] Vidi tablicu 5.4 i Zaključak 5.2
[84] Kako ovo za dobiti matematički??????

### 5.7.2. 5.7.2. SKLOPOVSKE TEHNIKE ZA ODREĐIVANJE PRASKOVITIH UZORAKA I NJIHOVIH POLOŽAJA U SINDROMIMA POGREŠAKA

Pokazujemo jednostavnu sklopovsku tehniku za određivanje uzorka praska i njegovoga položaja *x*. Načela koja se ovdje primjenjuju temelje se na prethodnim raspravama. Neka **s** označava sindrom pogreške duljine *u*, a **z** označava sindrom pogreške duljine *t*. Oba sindroma generiraju se iz primljene poruke. Znamo da je broj mjesta za koje se ovi sindromi moraju pomaknuti ciklički u lijevo kako bi se dobili uzorci praska *x* mod *u* odnosno *x* mod *t*. Na temelju definicije iz modulo aritmetike, slijedi da ako ciklički pomičemo svakoga od sindroma u lijevo *za x mjesta*, pomak **z** bit će jednak uzorku praska, a pomak **s** imat će uzorak praska u svojih *t* mjesta lijevo.

Na temelju ovoga načela i bez ova dva sindroma, možemo ustrojiti jednostavan način određivanje praskovitoga uzorka i njegovoga položaja. Jednostavno napravimo ciklički pomak u lijevo dok lijevih *t* bitova pomaknutoga **s** ne bude jednako pomaknutome **z**, a oba započinju s 1. Onda je praskovit uzorak, pomak **z**, a broj pomaka koji dovode do ove situacije, položaj je praska *x*. Kao primjer promotrimo slučaj gdje je $u = 5$ i $t = 3$. Pretpostavimo da je **s** = [00101], a **z** = [110]. U nastavku su navedeni oblici uzastopnih cikličkih pomaka u lijevo za **s** i **z**.

| pomak[85] | **s** | **z** |
|---|---|---|
| 0 | 00101 | 110 |
| 1 | 01010 | 101 |
| 2 | 10100 | 011 |
| 3 | 01001 | 110 |
| 4 | 10010 | 101 |
| 5 | 00101 | 011 |
| 6 | 01010 | 110 |
| 7 | 10100 | 101 |

Nakon 7 pomaka, stigli smo u opisanu situaciju. Uzorak praska za ovaj slučaj je onda [101], a njegovo mjesto unutar primljene poruke je 7.

### 5.7.3. 5.7.3. OBJAŠNJENJE OPASKI

Možemo zaključiti ovo pod-poglavlje navodeći dvije važne napomene:

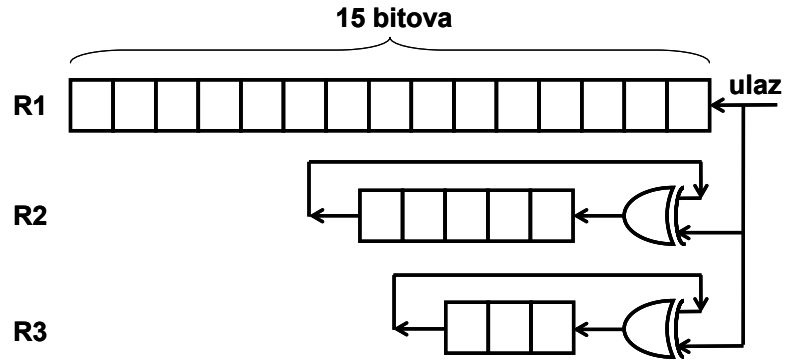
1. Utvrdilo se da kodna riječ treba neovisno zadovoljiti uvjete koji omogućuju otkrivanje praskova duljinâ *u* i *t*, gdje je $u = 2t–1$. Vrijednost $2t–1$ je *minimalna* vrijednost koju bi *u* trebao imati. Bilo koja vrijednost veća od toga omogućit će također, ispravak praska duljine *t*, pod uvjetom da duljina kodne riječi ne prelazi $u \cdot t$. Također, *u* i *t* trebaju biti relativno prosti (*primality*)[86]. Zbog učinkovitosti razmatranja, pretpostavljamo ovdje, ali i poslije, slučaj gdje je *u* jednak svojoj donjoj granici.
2. Prethodno obrađena shema ispravke praska vrijedi samo u slučajevima u kojima uzorak praska duljine *t*, nije periodički (tj., nema dio koji se ponavlja). Ako je uzorak periodički, onda ima nekoliko cikličkih pomaka koji vode istome uzorku. Uzmite, na primjer, slučaj gdje je uzorak "sve 1". On je jednak za bilo koji ciklički pomak. Tako imamo slučaj gdje opisana tehnika za određivanje *x* mod *t* neće dati jedinstven rezultat, jer se nakon različitih pomaka dobije isti rezultat.

Kako bi prevladali ove poteškoće, možemo ograničiti vrijednost *t* da bude prost broj. To će osigurati da uzorak nije periodički, osim za slučaj "sve 1", u kojemu slučaju to stvarno ne možemo ispraviti. Imajte na umu, međutim, da ovdje opisana obrada kodova za ispravak praskovitih pogrešaka, namijenjena je isključivo uvođenju osnovnih načela. Nema praktičnoga koda koji se temelji isključivo na opisanoj tehnici. U profinjenijim shemama, prethodno opisane poteškoće, lako se nadvladaju.

[85] C++ … brojevi koraka!
[86] *primality* ... riječ ne postoji u rječniku (možda znači: *prije svega*)

## 5.8. 5.8. Sklop za ispravak praskovitih pogrešaka – dekoder

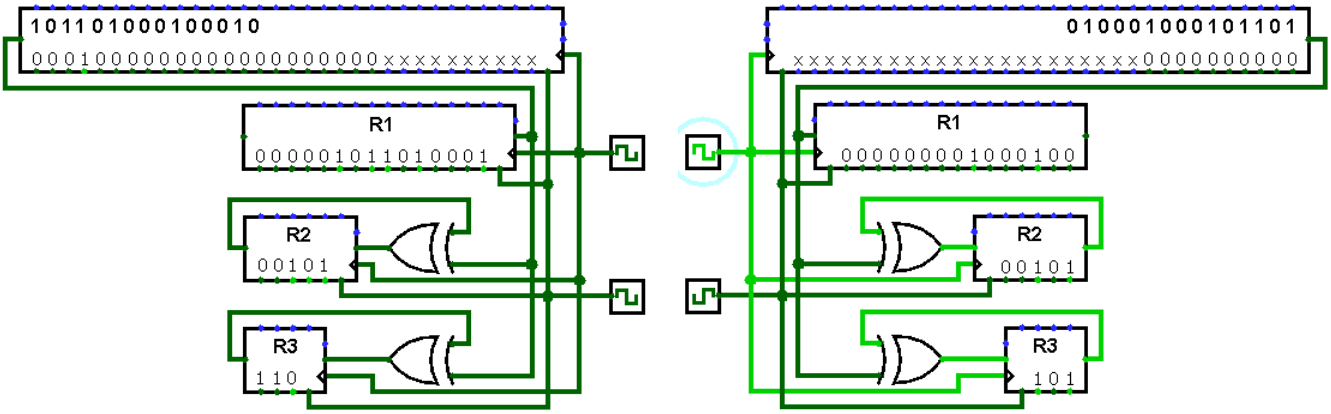
Dekoder, koji se opisuje u nastavku, omogućuje automatski ispravak praskovite pogreške duljine $t$, a temelji se na načelima *kineskoga teorema ostatka*. Postupak uključuje određivanje uzorka praska i njegov položaj unutar primljene poruke. Slika 5.5 prikazuje takav dekoder za slučaj $u = 5$, $t = 3$.




*Slika 5.5: Ispravak jednostruke praskovite pogreške duljine t = 3 u bloku duljine $2t^2–t = 15$*

### 5.8.1. 5.8.1. OTKRIVANJE PRASKOVITE POGREŠKE

Dekoder se sastoji od tri nezavisna registra: R1, R2 i R3 (slika 5.6).




*Slika 5.6: Sklop za otkrivanje praskovite pogreške*

Primljen signal duljine $3 \cdot 5 = 15$, pomiče se istodobno u sva tri registra, gdje registar R1 pohranjuje poruku. Registri R2 i R3 predstavljaju standardne sindrom-generatore dužine 5 odnosno 3. Za objašnjenje, uzmimo poruku **m'** = [101101000100010] koju smo prethodno razmatrali i napravimo posmik u krug. Bit koji se prvi posmiče u dekoder, prvi je lijevi bit u **m'** (a vrijednost mu je "1").

Konačan sadržaj R3 je [110], a konačan sadržaj R2 je [00101][87]. Da bi ispravili **m'**, moramo odrediti praskovit uzorak u dekoderu i njegov položaj $x$. Sada primjenjujemo tehnike opisane u poglavlju 5.6.3. Posmičemo bitove i u R2 i u R3 sve dok sadržaj lijeva tri stanja R2 ne bude *jednak* onome stanju u R3, gdje je lijevi bit jednak 1. Ovdje treba dodati još jedan sklop za otkrivanje (usporedbu) opisane jednakosti.[88]

Dakle, sadržaj R2 predstavlja praskovitu pogrešku, a broj posmika je mjesto prvoga pogrešnoga bita u **m'**. Imajte na umu da pri izradi slike 5.5 nismo koristili dogovor o LFSR s desnim posmikom pa je ulaz na desnoj strani. To je učinjeno kako bi krug odgovarao prethodnim objašnjenjima. To će se poslije "odraziti" na uobičajene dogovore.

### 5.8.2. 5.8.2. ISPRAVAK OTKRIVENE PRASKOVITE POGREŠKE

Pogreške u **m'** mogu se automatski ispraviti ako se **m'** posmikne, bit po bit, iz registra R1 zajedno s cikličkim pomakom u druga dva registra. **Broj** *posmika*, potreban za dosegnuti opisanu situaciju (u

[87] Simulacijski primjer na slici ove vrijednosti prikazuje u obrnutome redoslijedu!
[88] Zadati kao seminarski rad!

kojoj je sadržaj ~~lijeva~~ **desna** tri stanja R2 jednak sadržaju R3, a ~~lijevi~~ **desni** bit je 1), jednak je **položaju *x*** prvoga pogrešnoga bita u **m'**.

Imajte na umu da, ako se bitovima u **m'** napravi posmik istom brzinom kao posmik u R2 i R3, onda je *x*-ti bit u **m'** *upravo onaj bit koji se pomakne kada se otkrije opisana situacija*. Dok R2 u tomu trenutku sadrži uzorak praska, pogreška se cjelovito ispravlja dodavanjem sadržaja R2, bit po bit, trima bitovima odstranjenima iz R1.

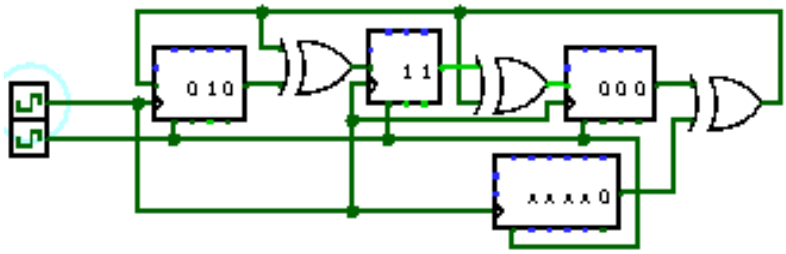
## 5.9. 5.9. Sklop za ispravak praskovite pogreške - koder

Sada razmatramo sklop za *kodiranje* koji odgovara *dekoderu* na slici 5.5. Sindrom-generator sastoji se od dvaju registara spojenih paralelno, a načini izgradnje LFSR za kodiranje podrobno su opisani u poglavlju 4.4. Povratne veze našega LFSR za kodiranje mogu se odrediti jednadžbom:

informacijski vektor × [matrica s 4 ciklička pomaka vektor-generatora] = povratne veze LFSR kodera

$$\mathbf{v}_3 = [1001] \cdot \begin{bmatrix} 100001000 \\ 010000100 \\ 001000010 \\ 000100001 \end{bmatrix} = [100101001] \cong [x^8 + x^5 + x^3 + x^0].$$

Lijevih osam bitova rezultirajućega vektora $\mathbf{v}_3$ = [100101001] naznačuju povratne veze LFSR kodera. Ovaj koder prikazuje slika 5.7.

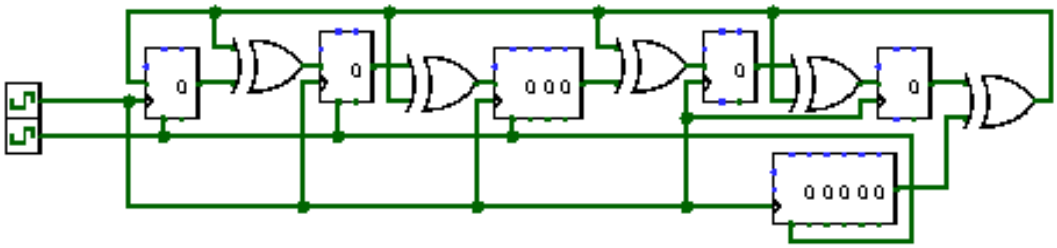



*Slika 5.7:Koder koji odgovara dekoderu na slici 5.5*

Imajte na umu da smo pri opisu gornjega postupka bili vrlo oprezni. Rekli smo da dobiven koder, *odgovara* dekoderu na slici 5.5. Nismo tvrdili da je to - koder. Pod time podrazumijevamo da postoji drugi LFSR, kraći od ovoga gore i on je koder našega koda. Da bi se razumjelo zašto, vratimo se vektorima $\mathbf{v}_1$, $\mathbf{v}_2$ i $\mathbf{v}_3$ iz poglavlja 4.4.

Jedini zahtjev bio je da se $\mathbf{v}_3$ sastoji od linearnih pomaka $\mathbf{v}_1$ i $\mathbf{v}_2$. Jedan od načina dobivanja takvoga vektora $\mathbf{v}_3$, opisao se kao množenje vektora i matrice.

Ovo nije jedini način dobivanja vektora koji se sastoji od linearnih pomaka $\mathbf{v}_1$ i $\mathbf{v}_2$. U ovome posebnome slučaju, imamo da su: $\mathbf{v}_1$ = [1001] i $\mathbf{v}_2$ = [100001000]. To se može potvrditi provjerom da se vektor $\mathbf{v}_3$ = [111001110] sastoji od zbroja linearni pomaka $\mathbf{v}_1$ i $\mathbf{v}_2$. Povratne veze LFSR koje odgovaraju $\mathbf{v}_3$, naznačuju njihovih sedam lijevih bitova. Ovaj LFSR prikazuje slika 5.8.




*Slika 5.8: Još jedan moguć koder* $(x^0 + x + x^2 + x^5 + x^6 + x^7)$

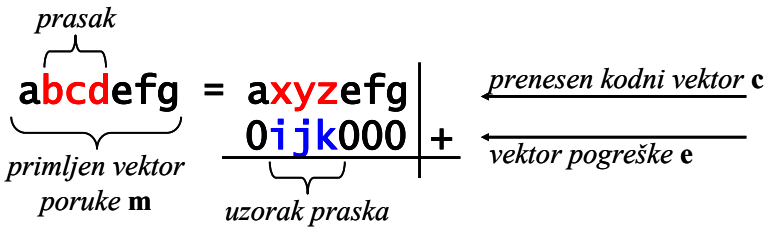
Kod za ispravak pogrešaka čiji dekoder prikazuje slika 5.5, ima kodne riječi duljine 15. Kineski teorem o ostatku, govori nam da se prasak pogrešaka ne može ispraviti, ako je kodna riječ duža od 15. Dva kodera na slikama 5.7 i 5.8, pogodna za naš kod, imaju različite duljine. Ako duljinu registra za

kodiranje određuje broj paritetnih bitova, onda je kod stvoren koderom na slici 5.7 dimenzije (15, 7), a kod stvoren prvim registrom na slici 5.5 je dimenzije (15, 8).

Drugi kod je učinkovitiji, jer on kodira dodatan informacijski bit, nudeći istu mogućnost ispravke pogreške. Također, vrijednost *Pu* - koja definira periodičnost LFSR povezanu stanjem [1000 … 0] - je 27 za prvi koder, a 15 za drugi. Na temelju Tvrdnje 4.3, drugi koder je ciklički. Prvi koder nije ciklički, što se može dokazati uzimanjem, na primjer, kodne riječi [000000100101001]. Posmik ove kodne riječi ciklički za jedno mjesto u desno, daje vektor koji nije kodna riječ.

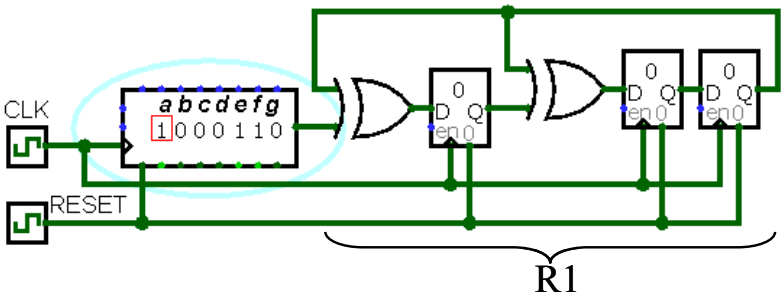
## *5.10. 5.10. Korištenje cikličkoga Hammingova koda za otkrivanje praskovitih pogrešaka*

Slika 5.9 prikazuje komunikacijski scenarij u kojemu **m** = [*a b c d e f g*] predstavlja primljenu poruku čija je poslana inačica kodna riječ **c** = [*a x y z e f g*].




*Slika 5.9: Osnovni komunikacijski scenarij koji uključuje praskovitu pogrešku*

Poruka **m** sadrži praskovitu pogrešku duljine 3, ograničenu na mjesta 2, 3 i 4 (računajući s lijeva). Vektor **m** onda se sastoji od zbroja **c** i vektora pogreške **e**, gdje **e** sadrži uzorak praska (*i j k*) na mjestima 2, 3 i 4, a ostatak njegovih elemenata su 0. Sada promotrimo slučaj gdje je **c** kodna riječ koda, čija matrica pariteta je $\mathbf{H}^T$ (poglavlje 3). Sindrom-generator ovoga koda prikazuje slika 5.10, a to je ponavljanje prikaza slike 3.7.




*Slika 5.10:* $R(x) = 1 + x + x^3$ *(ponovno nacrtana slika 3.7)*

Zbog **m** = **c** + **e** i zbog linearnosti R1, slijedi da je sadržaj R1, nakon posmika poruke **m** u njega, jednak sadržaju dobivenome posmikom samo **e** u njega (dok je pribrojnik **c**, dodana kodna riječ koja vodi sadržaj u stanje "sve 0"). Razmotrite sada učinak posmika vektora pogreške **e** u R1. Najprije smo razmatrali samo posmik **e**, uz uvjet da će se naša konačna analiza temeljiti na najmanje sedam posmika. Zapravo radimo posmik **m** u R1, a učinak od **c** uočava se tek nakon sedam posmika. Nakon 6 posmika, sadržaj R1 praskovit je uzorak (*i j k*). Nakon sedmoga posmik, kada se **e** potpuno pomaknuo u R1, sadržaj se mijenja pa više nema praskovitoga uzorka.

Ovaj sadržaj, sada je *sindrom pogreške*. Zbog linearnosti registra, ako njegovi sadržaji nisu nule, on nikada neće imati sadržaj "sve 0", bez obzira na to koliko puta napravimo posmik. Nakon što se **e** u potpunosti pomaknuo, konačan sadržaj LFSR različit je od nule budući da je različit od nule i nakon šestoga posmika. Drugim riječima, postojanje praskovite pogreške duljine 3 ili manje uvijek će stvarati ne-nulti sindrom, što omogućuje otkrivanje pogrešaka. Ovo razmatranje neovisno je od činjenice da naš poseban LFSR ima maksimalnu periodičnost, a to nas vodi do sljedećega općega zaključka.

*Zaključak 5.3* Bilo koji kod što ga stvori LFSR duljine *n,* može se koristiti za otkrivanje pojave praskovite pogreške u primljenoj kodnoj riječi, duljine *n* ili manje.

Pitanje koje se sljedeće razmatra jest mogućnost osmišljavanja LFSR za otkrivanje praskovite pogreške, koji ima i svojstvo da kodne riječi što ih je stvorio, imaju također i Hammingovu težinu.

Ovo će omogućiti otkrivanje *neparnoga* broja pogrešaka (što se trivijalno[89] prikazalo u prvome poglavlju), uz otkrivanje praskovitih pogrešaka. Zbog toga što je LFSR usvojen kao međunarodni standard, kao što će se prikazati poslije, ovdje ga se posebno razmatra.[90]

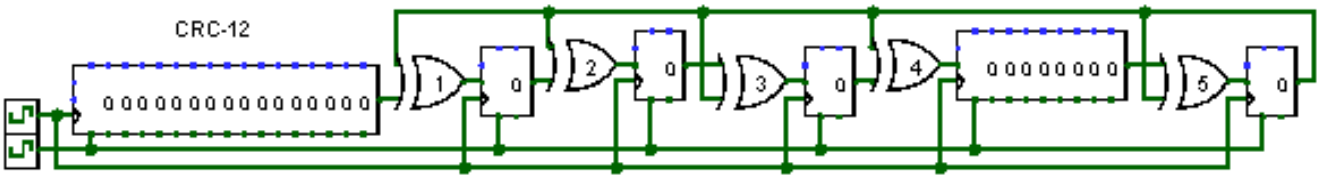
Pokazali smo u 4. poglavlju da ako LFSR generira kod duljine *n*, onda prvih *n* bitova prvoga retka generator-matrice koda**, G**, odgovaraju povratnim vezama LFSR-generatora. Iza ovih *n* bitova slijedi 1, a ostatak bitova u retku su 0. Onda slijedi da ako je broj stanja u LFSR-generatoru, neparan, onda prvi redak **G** ima paran broj jedinica.[91]

Vratimo se sada Tvrdnji 4.1. Ona tvrdi da se sve kodne riječi nekoga koda sastoje od zbroja linearnih posmika prvoga retka generator-matrice koda, **G**. Sve ove promjene imaju istu Hammingovu težinu kao prvi redak i sve imaju isti paritet. Ako sve imaju paritet 0, kao u našemu slučaju, onda bilo koji njihov zbroj (tj., bilo koje kodne riječi) također imaju paritet 0. Prema tomu, ako kod *C* kojega generira LFSR, ima svojstvo da je broj stanja gdje povratna veza puni registre, neparan, onda sve kodne riječi **c** imaju paritet 0.

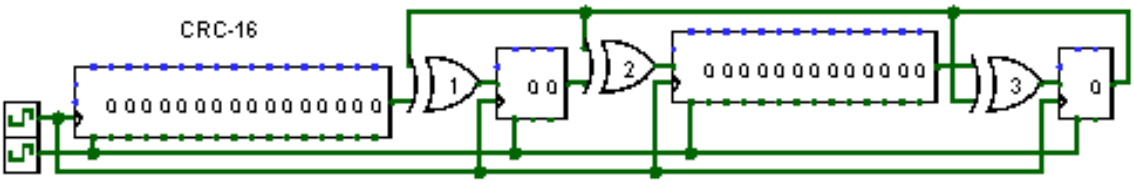
*Zaključak 5.4* Uvijek je moguće otkriti postojanje neparnoga broja pogrešaka koje se pojave u kodnoj riječi koda *C*, jer LFSR-generator, puneći registre povratnom veza, ima neparan broj stanja.

Specifični LFSR-generatori kodova za otkrivanje praskovitih pogrešaka, prihvaćeni su kao međunarodni standardi i definirani su za generatore $G(x)$ od 8, 12, 16 i 32 bita. Standard, $G_{\text{CRC-32}}(x)$, primjenjuje se u brojnim IEEE protokolima na razini veze. Projekt 802 LAN i *Ethernet* koriste generator $G(x)$ = 10000010011000001000111010110111. Serijski priključak USB (*Universal Serial Bus*) koristi CRC s $G(x)$ = 11000000000000101.

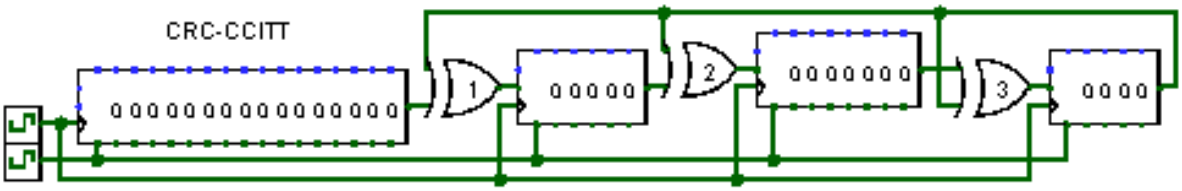
Šest LFSR (CRC -8, -10, -12, -16, -32, -CCITT), od kojih tri prikazuje slika 5.11, imaju neparan broj ~~stupnjeva~~ (EXOR vrata) povezanih na zajedničku povratnu vezu.




(a) *CRC-12* $R(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$




(b) *CRC-16* $R(x) = x^{16} + x^{15} + x^2 + 1$




(c) *CRC-CCITT* $R(x) = x^{16} + x^{12} + x^5 + 1$

(d) [*CRC-32* $R(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^2 + x + 1$]

*Slika 5.11: Tri* LFSR *prihvaćena kao međunarodni standardi*

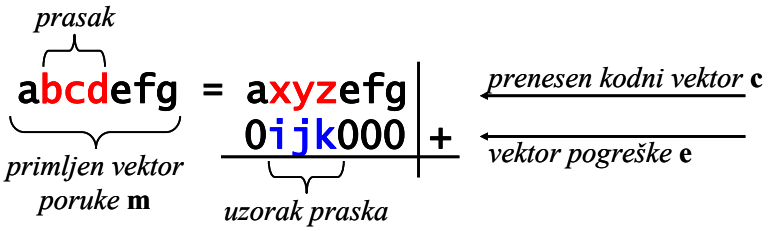
Njihova sposobnost otkrivanja pogrešaka može se odrediti iz zaključaka 5.3 i 5.4.

---

[89] *trivijalan* … (lat. *trivialis*), opće poznat, otrcan, običan, suviše poznat
[90] VIF!!!!
[91] Ibidem!!!

## 5.11. 5.11. Korištenje cikličkoga Hammingova koda za ispravak praskovitih pogrešaka na temelju poznavanja uzorka praska

Opet promotrimo sliku 5.9 i prateći, raspravimo je.




*Slika 5.12: Zbroj prenesenoga vektora i vektora pogreške*

Tvrdi se da je posmik **m** u R1, jednak posmiku samoga **e** u R1. Dok u prethodnome dijelu nismo koristili činjenicu da R1 ima najveću periodičnost (ova svojstvo nije bitno za analize otkrivanja praskovitih pogrešaka), ovdje ćemo koristiti to svojstvo, za analizu sposobnosti ispravka praskovite pogreške koda kojega stvara R1.

Ako se **e** posmiče u R1, pokazalo se da sadržaji R1 nakon 6. posmika, predstavljaju praskovit uzorak (*i j k*). Zbog periodičnosti R1, *ako ćemo se zadržati na sedmome posmiku, vratit će nam se uzorak praska*. Nećemo dobiti ovaj uzorak prije ovoga dodatnoga 7. posmika sve dok LFSR ima različite sadržaje tijekom sedam posmika, ako početni sadržaj nije bio "sve 0".

Općenito, ako nekako *znamo uzorak praska*, ali *ne znamo njegov položaj unutar* **m** (čija primljena inačica je kodna riječ našega koda), primjenjuje se sljedeći postupak.

Nakon učitavanja **m** u R1 (7 posmika), nastavimo dalje posmik sve dok R1 sadrži uzorak praska. Ako se ***dodatno posmikne za*** *k*, onda prasak počinje na mjestu 7–*k*, računajući s lijeva. Ako je prasak ograničen na mjesta 0, 1, 2, shema još uvijek vrijedi (objasnite zašto[92]). Gore naveden postupak zapravo je poopćenje postupka ispravke jednostruke pogreške cikličkoga Hammingova koda, što se obradilo već u 3. poglavlju. U tomu slučaju, znamo da je "praskovit uzorak" [100]. Tvrdimo da spoznaja o *bilo* kakvomu uzorku praska (odgovarajuće duljine) omogućuje ispravak pogrešaka.

Razmotrimo sada slučaj poruke **p** duljine 7*k*, za neki cio broj *k*, koji ima svojstvo da nakon posmika u registar R1 na slici 5.10, konačan sadržaj registra bude "sve 0". Neka se **q** može dobiti uvođenjem praskovite pogreške duljine 3 ili manje u **p**, gdje prasak započinje od mjesta *i*. Opet pretpostavimo da znamo uzorak praska. Na temelju periodičnosti R1 i na temelju prethodne rasprave (poruka **m** duljine 7) možemo zaključiti sljedeće: ako se **q** posmiče u R1, a R1 se dalje posmiče za *j* mjesta sve dok ne sadrži uzorak praska, onda je *i* mod 7 = 7–*j*. Ovo promatranje čini temelj koda što ga se uvodi u nastavku.

Za potrebe udžbenika, sada ponavljamo ista razmatranja kao i prije, gdje umjesto razmatranja rada LFSR, obrađujemo paritetnu matricu koda. Matrica pariteta našega specifičnoga koda je $\mathbf{H}^T$. Neka je $\mathbf{H'}_j$, matrica dobivena cikličkim pomakom redaka $\mathbf{H}^T$ za naviše *j* mjesta. Kako je naš kod ciklički (tj., ciklički pomak kodne riječi također je kodna riječ), imamo da je $\mathbf{c} \cdot \mathbf{H'}_j = 0$. Onda slijedi da $\mathbf{m} \cdot \mathbf{H'}_j$ daje zbroj redaka $\mathbf{H'}_j$ čija veličina određuje položaj pogrešaka u **m**. Imajte na umu da, iako smo napravili posmik redaka $\mathbf{H}^T$, a ne kodne riječ **c**, umnožak $\mathbf{c} \cdot \mathbf{H'}_j$ daje isti rezultat koji bi se dobio množenjem cikličkoga pomaka **c** s $\mathbf{H}^T$.

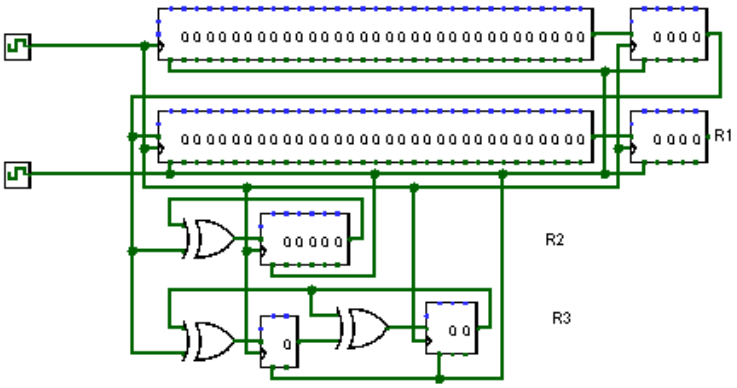
Neka je praskovita pogreška ograničena na mjesta #*i*, *i*+1, *i*+2 u **m**, gdje se bitovi računaju počevši s lijeva (prvi bit je #0). Zatim $\mathbf{m} \cdot \mathbf{H'}_{(7-i)}$ daje uzorak praska. To proizlazi iz činjenice da $\mathbf{H}^T$ započinje jediničnom matricom dimenzije (3×3). Onda $\mathbf{m} \cdot \mathbf{H'}_{(7-i)}$ ima tu jediničnu matricu na mjestima #*i*, *i*+1, *i*+2 pa $\mathbf{m} \cdot \mathbf{H'}_{(7-i)}$ zapravo, kao rezultat, daje uzorak praska pomnožen jediničnom matricom, a rezultat je uzorak praska. Imajte na umu, ako je *i* = 0, tj., pogreške su ograničena na prva 3 mjesta u **m** i ako je

[92] Objasniti zašto?

$\mathbf{H'}_{(7-i)} = \mathbf{H'}_7 = \mathbf{H}^T$, naš argument vrijedi i u ovome slučaju. Naša shema za ispravak pogrešaka sada je jasna. Ako znamo uzorak praska, onda množimo **m** cikličkim pomakom $\mathbf{H}^T$ dok ne dobijemo taj *uzorak praska*. *Mjesto pogreške* onda se može otkriti iz *broja posmika*.

## 5.12. 5.12. Fire kod

Razmatramo dekoder prikazan na slici 5.13.

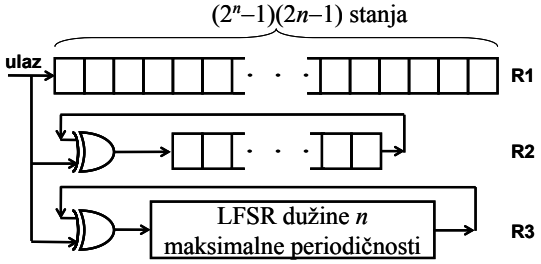



*Slika 5.13: Predložen dekoder*

Uzmimo (35, 27) kod[93] čije kodne riječi imaju svojstvo da, nakon što se napuni ovaj dekoder, R2 i R3 sadrže samo nule. Imajte na umu da je R3 sindrom-generator Hammingova koda prikazan na slici 5.10. Svaka kodna riječ našega koda zadovoljava sljedeći uvjet: zbroj svih bitova koji su razmaknuti 5 mjesta, počevši mjestom #0, #1, #2, #3, #4, daje rezultat 0.[94] Zbog ovoga općega svojstva, R2 sadrži 0 ako se u sustav učita kodna riječ. Činjenica da R3 također sadrži 0, kada se posmik potpuno završi, znači sljedeće: izgradnju matrice $\mathbf{M}^T$ ponavljanjem matrice $\mathbf{H}^T$ (iz poglavlja 3.1) jedne ispod druge pet puta. Matrica $\mathbf{M}^T$ onda ima tri stupca i trideset pet redaka. Svaka kodna riječ **c** našega koda onda ima svojstvo $\mathbf{c} \cdot \mathbf{M}^T = \mathbf{0}$.

$$\mathbf{M}^T = \begin{bmatrix} \mathbf{H}^T \\ \mathbf{H}^T \\ \mathbf{H}^T \\ \mathbf{H}^T \\ \mathbf{H}^T \end{bmatrix}$$

Svaka kodna riječ našega novoga koda zadovoljava osam nezavisnih provjera pariteta. To je omogućeno na odašiljačkoj strani pridruživanjem osam paritetnih bitova danome informacijskome vektoru. Naš kod omogućuje ispravak jednostrukih praskovitih pogrešaka duljine 3 u bloku duljine 35. Da bi se to razumjelo, prvo uočimo da takvom praskovitom pogreškom, konačan sadržaj R2 omogućuje određivanje uzorka praska, kao i $x$ mod 5, gdje je $x$ položaj praska. Opravdanje za ovaj dokaz jednako je našemu objašnjenju kruga na slici 5.5. Nakon izvođenja uzorka praska, možemo primijeniti tehniku predstavljenu u poglavlju 5.10. I dalje posmičemo R3 dok ne naiđemo na naš poznat uzorak praska. To će dati $x$ mod 7. Kineski teorem o ostatku onda jamči da se $x$ može odrediti. Dekoder općega Fire koda za $2^n-1$ i $2n-1$ je *relativno prost*, a prikazuje ga slika 5.14.



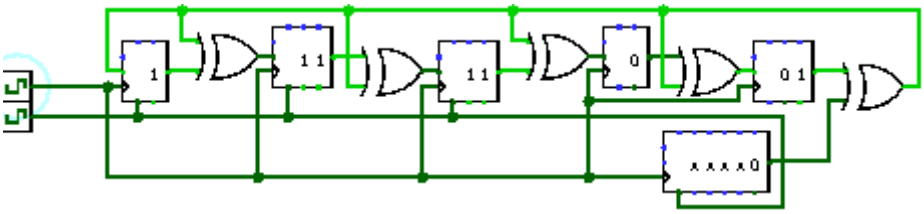

*Slika 5.14: Opći dekoder Fire koda*

[93] C++!!!
[94] Napraviti C++ zadatak!

To je poopćenje isto kao i ono koje prikazuje slika 5.13, a njegova valjanost temelji se na istome razmatranju. Oblikovanje krugova kodiranja što odgovaraju našim dekoderima, opet se temelji na detaljima navedenima u poglavlju 4.4. Sada se razmatra određen poseban koder Fire koda čiji dekoder prikazuje slika 5.13. Sindrom-generator sastoji se od dvaju registara: registra R2 dužine 5 i registra R3 dužine 3, spojenih paralelno. Priključci povratne veze LFSR kodera, odgovaraju našemu dekoderu, a može ih se odrediti sljedećim postupkom:

$$[100001] \cdot \begin{bmatrix} 110100000 \\ 011010000 \\ 001101000 \\ 000110100 \\ 000011010 \\ 000001101 \end{bmatrix} = [110101101] \cong (1+x+x^3+x^5+x^6+x^8)$$

Lijevih osam bitova rezultirajućega vektora [110101101] naznačuje povratne veze LFSR kodera. Ovaj koder prikazuje slika 5.15.




*Slika 5.15: Koder što odgovara dekoderu na slici 5.13*

Dva skupa paritetnih jednadžbi kojima upravljaju sindrom-generatori R2 i R3 na slici 5.13, potpuno su neovisna, za razliku od slučaja skupa jednadžbi kojima upravljaju R2 i R3 na slici 5.5. To znači da se koder našega Fire koda ne može ostvariti pomoću LFSR koji je kraći od onoga prikazanoga na slici 5.15.

# 13. *Laboratorijska vježba 12: Fire kod*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

- Hammingov (7, 4, 3) kod. Ima 16 kodnih riječi duljine 7. Može ga se koristiti za slanje $2^7$ = 128 poruka i ispravak 1 pogreške.
- Golayev (23, 12, 7) kod. Ima 4096 kodnih riječi. Može ga se koristiti za prijenos 83.888.608 poruka i ispravak 3 pogreške.
- Kvadratni ostatak (*Quadratic residue*) (47, 24, 11) kod. Ima 16.777.216 kodnih riječi. Može ga se koristiti za prijenos 140.737.488.355.238 poruka i ispravak 5 pogrešaka.

Golayeve kodove $G_{24}$ i $G_{23}$ koristili su Voyager I i Voyager II za prijenos slike u boji od Jupitera i Saturna. Generator-matrica ima oblik $\mathbf{G}_{24}$ i predstavlja (24, 12, 8) kod. Težine svih kodnih riječi su višekratnici od 4. Matrica $\mathbf{G}_{23}$ dobije se iz $\mathbf{G}_{24}$ brisanjem posljednjih simbola svake kodne riječi u $\mathbf{G}_{24}$. Kod $G_{23}$ je (23, 12, 7) kod. Generator matrica Golayeva koda, $\mathbf{G}_{24}$, ima jednostavnu građu. Prvih 12 stupaca oblikuju jediničnu matricu $\mathbf{I}_{12}$, a 13. stupac ima same jedinice. Retci u ostalih 11 stupaca cikličke su permutacije prvoga retka koje sadrže 1 na onim mjestima koji su kvadrati modulo 11, tj. na mjestima 0, 1, 3, 4, 5 i 9.

### 5.13.1. 5.13.1. BCH KODOVI

Bose-Chadhuti-Hocquenghem (BCH) kodovi predstavljaju poopćenje Hammingovih kodova koji omogućuju ispravak višestrukih pogrešaka. Oni su *snažan razred cikličkih kodova* koji omogućuju velik izbor duljine blokova, brzinu kodova, veličinu abecede i sposobnosti ispravke pogrešaka. Tablica 5.5 popisuje neke generatore koda $G(x)$[95] koji se obično koriste za izgradnju BCH ($n$, $k$, $t$) kodova za različite vrijednosti $n$ (=*ukupan broj bitova=informacijski+paritetni*), $k$ (=*broj informacijskih bitova*) i $t$ (=*broj mogućih ispravaka pogreški*), do duljine bloka od 255.

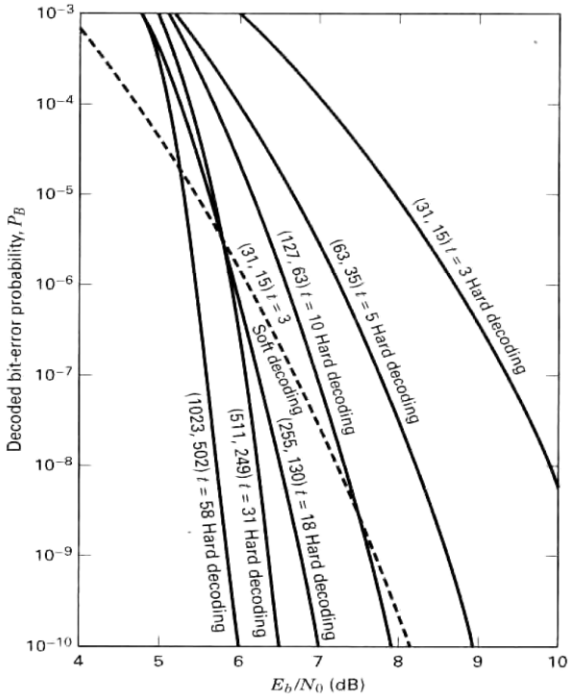
*Tablica 5.5 Prvih desetak polinom-generatora primitivnih BCH kodova (brojevi u stupcu $G(x)$ predstavljaju oktalni prikaz binarnih polinoma zbog kratkoće pisanja) (napraviti kao zadatak i nacrtati LFSR)*

| bitova | | | oktalno | binarno | polinom-generator |
|---|---|---|---|---|---|
| $n$ ukupno | $k$ informacijski | $t$ ispravke | $G(x)$ | | |
| 7 | 4 | 1 | 13 | 1011 | $x^3+x+1$ |
| 15 | 11 | 1 | 23 | 10011 | $x^4+x+1$ |
| | 7 | 2 | 721 | 111010001 | $x^8+x^7+x^6+1$ |
| | 5 | 3 | 2467 | 10100110111 | $x^{10}+x^8+x^5+x^4+x^2+x+1$ |
| 31 | 26 | 1 | 45 | 100101 | $x^5+x^2+1$ |
| | 21 | 2 | 3551 | 11101101001 | $x^{11}+x^{10}+x^9+x^7+x^6+x^4+1$ |
| | 16 | 3 | 107657 | 1000111110101111 | … |
| | 11 | 5 | 5423325 | 10110001001101010101 | … |
| | 6 | 7 | 313365047 | 11001011011110101000100111 | … |
| 63 | 57 | 1 | 103 | 1000011 | $x^6+x+1$ |
| | 51 | 2 | 12471 | 1010100111001 | $x^{12}+x^{10}+x^8+x^5+x^4+x^3+1$ |
| ⋮ | … | … | … | … | … |

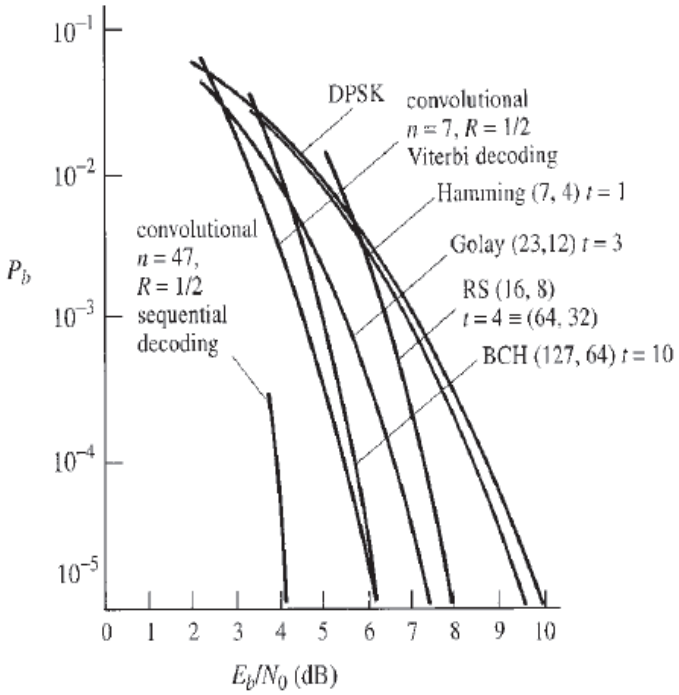
Koeficijenti $G(x)$ prikazuju se kao oktalni brojevi raspoređeni tako da kada se pretvore u binarne znamenke, znamenka sasvim desno odgovara koeficijentu $G(x)$ nultoga stupnja. Lako se može provjeriti svojstvo cikličkoga koda da je polinom-generator stupnja $n–k$. Za blokove duljine nekoliko stotina bitova, BCH kodovi nadmašuju sve ostale blok-kodova iste duljine i brzine koda. Najčešće korišteni BCH kodovi koriste binarnu abecedu i duljinu bloka kodne riječi od $n = 2^m–1$, gdje je $m$ = 3, 4, ….

[95] G od *Galois*

Naziv tablice 5.5 naznačuje da su prikazani generatori BCH kodova nazivaju *primitivni kodovi*. Pojam "primitivan" je teorijski koncept broja koji zahtijeva algebarski razvoj. Relativno širok maksimalan dobitak kodiranja u odnosu na brzinu koda za određen $n$, javlja se otprilike između brzina kodiranja 1/3 i 3/4. Gaussov kanal značajno narušavaju svojstva kodova pri velikim i malim brzinama. Slika 5.16 predstavlja izračunatu učinkovitost BCH koda pomoću koherentno demodulirane BPSK uz dekodiranje i tvrdom i mekom odlukom.




a) Za koherentno demoduliranu BPSK u AWGN kanalu korištenjem BCH kodova.[96]




b) Za DPSK signale u AWGN kanalu, brzine 1/2, pri dekodiranju tvrdom odlukom

*Slika 5.16: $P_B$ u odnosu na $E_b/N_0$*

Dekodiranje mekom odlukom obično se ne koristi za blok-kodove zbog svoje složenosti. Međutim, kada god se provodi, ono nudi približan dobitak kodiranja od 2 dB u odnosu na dekodiranje tvrdom odlukom. Uz zadanu brzinu koda, vjerojatnost pogreške dekodiranja poboljšat će se povećanjem duljine bloka $n$. Dakle, za određenu brzinu koda, zanimljivo je razmotriti duljinu bloka koja će se zahtijevati za svojstva dekodiranja tvrdom odlukom da bi se sučelila svojstvima dekodiranja mekom odlukom.

Na slici 5.16, prikazani BCH kodovi imaju brzine koda približno 1/2. Iz slike čini se da za određenu brzinu koda i dekodiranje tvrdom odlukom, BCH kod duljine $8\times n$ ili dulji, ima bolja svojstva (za $P_B$ od oko $10^{-6}$ ili manje) od onih pri dekodiranju mekom odlukom za BCH kod duljine $n$. Jedna posebna pod-vrsta BCH kodova (otkriće koje je prethodilo BCH kodovima), posebno je korisna za ne-binarni skup, a zovu se Reed-Solomonov kodovi.

## 5.14. 5.14. Ispravak pogrešaka jednoga znaka (Reed-Solomonovi kodovi)

### 5.14.1. 5.14.1. OTKRIVANJE POGREŠKE ZNAKA

Ponekad se poruke prenose ili pohranjuju kao blokovi znakova. Poruka **m** = [100, 001, 011, 101, 110, 111, 010], što će se koristiti u sljedećim primjerima, čine blok od sedam znakova, a svaki je duljine 3. Neki kodovi posebno su oblikovani za ispravak pogrešaka koje se javljaju u primljenim znakovima. Ovi kodovi postali su popularni prije 40-tak godina, zbog njihove primjene u tehnologiji kompaktnoga diska. Ovdje se informacije pohranjuju u obliku blokova bajtova koji su vrlo osjetljivi na pogreške.

---

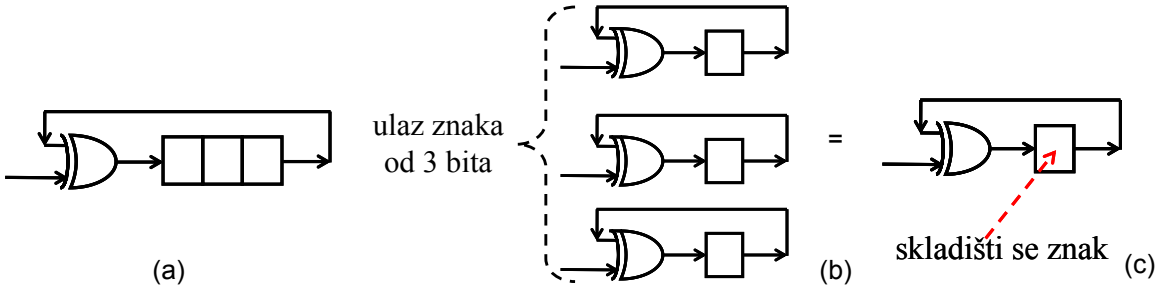
[96] Preuzeto iz L.J. Weng "*Usporedbe svojstava BCH kodova za meko i tvrdo dekodiranje*" Proc. Int. Conf. Commun., 1979, Sl.3, str. 2555© 1979 IEEE.

[98] *komercijalan* … (lat. *commercialis*) trgovački, obrtni, privredni, prometni; društveni; komercijalna ulica trgovačka ulica; komercijalne igre društvene igre; komercijalni sustav državne privredne politike koji povlašćuju trgovinu, osobito na račun poljoprivrede; komercijalni traktat trgovinski ugovor

Specifičan kod, odabran za uporabu u tehnologiji kompaktnoga diska, je RS (Reed-Solomon) kod. Slučaj koji se razmatra u ovome tekstu je onaj u kojemu smo pretpostavili da je unutar bloka pogrešan samo jedan znak. RS kod zapravo je kod za ispravak praskovitih pogrešaka, budući da ispravljamo višestruke pogreške bitova unutar zadanoga okvira jednoga znaka. Onda duljina znaka predstavlja duljinu praska kojega treba ispraviti.

*Definicija* Neka je **c** poslan znak i neka je **c'** njegova primljena inačica. Uzorak **e** = **c** + **c'** zove se *uzorak pogreške znaka*.

Iako su pogreške unesene u **c**, samo poseban oblik praska, morali smo uvesti navedenu definiciju kako bi se napravila razlika između *uzorka pogreške znaka* i *uzorka praskovite pogreške*. Dok ovaj drugi, prema definiciji, mora započeti jedinicom, za prvi to nužno nije neophodno. Primjerice, ako je **c** = [110], a **c'** = [111], onda je uzorak pogreške znaka [001]. Ispitajmo poruku **m** zadanu prije, a koja se sastoji od 21-oga bita. Primjena tehnike ispravke praska pogrešaka za ispravak pogrešaka znakova ovdje je puno lakša nego primjena takve tehnike za ispravak bilo kojega praska duljine 3. Znamo da uzorak pogreške znakova počinje bitom #3*p*, a završava bitom #3*p*+2, za neki cio broj *p*. Slika 5.17.a prikazuje krug za otkrivanje praskovite pogreške (za prasak duljine 3). Slika 5.17.b prikazuje krug za otkrivanje pogreške znakova.




*Slika 5.17: Otkrivanje praskovitih pogrešaka; (b) otkrivanje pogreške znakova*

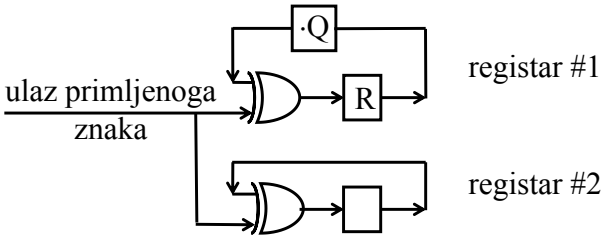
Cio znak duljine 3 uvodi se u krug tijekom svakoga posmika, gdje se obavlja XOR operacija između pohranjenoga znaka i znaka koji se uvodi. Krug na desnoj strani opisuje isto ponašanje i predstavlja dogovor koji se koristi u ovome tekstu. Lako se može provjeriti da će nakon posmika poruke **m** (navedene gore), u krug na slici 5.17.b, registar imati sadržaj "sve 0". Neka je **m'** primljena inačica **m**. Iz materijala predstavljenoga prije u ovome poglavlju, slijedi da ako je znak pogrešan (tj., imamo uzorak pogrešaka sveden na tri uzastopna mjesta, počevši mjestom #3*p*), onda su ovi sadržaji, *uzorak pogrešnoga znaka.*

*Primjer* **m'** = [100, 001, 011, 101, 110, 111, 010].

Vektor **m'** dobio se uvođenjem uzorka pogrešnoga znaka [001] u peti znak (brojeći s lijeva). Nakon posmika **m'** u jedan od krugova koje prikazuje slika 5.17, konačan sadržaj bit će [001]. Otkrivanje praskovite pogreške duljine *t* u primljenoj poruci, omogućuje uvođenje *t* paritetnih bitova u poslanu poruku. Iz toga slijedi da u slučaju jednoga pogrešnoga znaka, *uvođenje paritetnoga znaka u poslan blok znakova, omogućuje određivanje uzorka pogrešnoga znaka*.

### 5.14.2. 5.14.2. ISPRAVAK NEISPRAVNOGA ZNAKA

Kodna riječ u RS kodu je blok što se sastoji od $2^n$–1 znakova duljine *n*. Postoje dva paritetna znaka i $2^n$–3 informacijska znaka u kodnoj riječi za ispravak jedne pogreške RS koda (čime se omogućuje ispravak jednoga pogrešnoga znaka). Kod je sustavan. Zbog sažetosti, kada god se spomene RS kod od sada pa nadalje, točnije mislimo na kod za ispravak jedne pogreške. Dekoder našega RS koda temelji se na *sindrom-generatoru* na slici 5.18.




*Slika 5.18: Sindrom-generator RS koda za ispravak jedne pogreške*

Registri prikazani na slici 5.18, skladište cijele znakove. Odašiljački blok znakova ima svojstvo da ako ga se posmikne u dva sindrom-generatora, oba će na kraju imati sadržaj "sve 0". Rad registra #1 objašnjava se na sljedeći način. Postoji LFSR maksimalne periodičnosti duljine $n$ povezan RS kodom. Prisjetimo se da $n$ označava duljinu znakova, gdje se prenošen blok sastoji od $2^n$–1 znakova. Označimo ovaj registar kao R, gdje $\mathbf{R}_i$ predstavlja njegov sadržaj nakon $i$ posmika, počevši početnim sadržajem $\mathbf{R}_0$ = [1000 … 0]. Matrica **Q** pridružena je R. Ova matrica jedna je od onih koje su se opisale u poglavlju 4.1 (svojstvo c). Sastoji se od $n$ redaka, a to su sadržaji registra R, počinje s $\mathbf{R}_1$ i završava s $\mathbf{R}_n$. Pokazalo se da je $\mathbf{R}_i \cdot \mathbf{Q} = \mathbf{R}_{i+j}$, za bilo koji $0 \le i \le 2^n$–2.

Promatranjem registra #1, možemo uočiti da se njegov sadržaj množi s **Q** prije nego što ga se zbroji (XOR) dolaznim znakom. To jest, (novi sadržaji) = (stari sadržaj) · **Q** + (dolazni znak). Zato što je bilo koji sadržaj registra bio ili "sve 0" ili jednak nekome $\mathbf{R}_i$, nalazimo da množenjem **Q** daje **0** u prvom slučaju, a $\mathbf{R}_{i+1}$ u drugome. U svrhu objašnjenja, tablica 5.6 popisuje uzastopne sadržaje registra #1 za slučajeve u kojima se poruke **m** i **m'**, navedene u prethodnim primjerima, pomiču u njega.

*Tablica 5.6 Uzastopni sadržaji registra #1 na slici 5.18*

| | Ulazni blok **m** | Ulazni blok **m'** | | Ulazni blok **m** | Ulazni blok **m'** |
|---|---|---|---|---|---|
| 1 | 010 | 010 | 5 | 101 | 110 |
| 2 | 110 | 110 | 6 | 101 | 010 |
| 3 | 101 | 100 | 7 | 000 | 101 |
| 4 | 001 | 111 | | | |

Ovdje je $n$ = 3, a registar R je onaj prikazan na slici 3.1. Pridružena matrica **Q** je:

$$\mathbf{Q} = \begin{bmatrix} 010 \\ 001 \\ 110 \end{bmatrix}$$

Pri razmatranju registra #2 na slici 5.18, prepoznalo ga se, kao jednakoga onome na slici 5.17. On zbraja primljene znakove. Iz prethodnih rasprava znamo, ako zbroj znakova poslanoga bloka predstavlja znak "sve 0", a došlo je do pogreške jednoga znaka za vrijeme prijenosa, onda zbroj primljenih znakova daje uzorak pogreške znaka. U točno određenome slučaju odaslane poruke **m** i primljene poruke **m'** koje smo prije obrađivali, zbroj znakova je [000] odnosno [001].

Sada ćemo objasniti ispravak pogreške našega RS koda. Poslana poruka **m** i njena primljena inačica **m'** imaju ukupno $2^n$–2 znaka i mogu se razlikovati u jednome znaku. Neka je **e** = **m** + **m'**. Blok pogreške **e** sastoji se od $2^n$–2 znaka koji su "sve 0" i jednoga znaka koji je jednak uzorku pogreške. Kako je naš kod linearan, a **m** je kodna riječ, sadržaj sindrom-generatora na slici 5.18 nakon posmika **m'** u njih, isti je kao da se posmiče sam **e**.

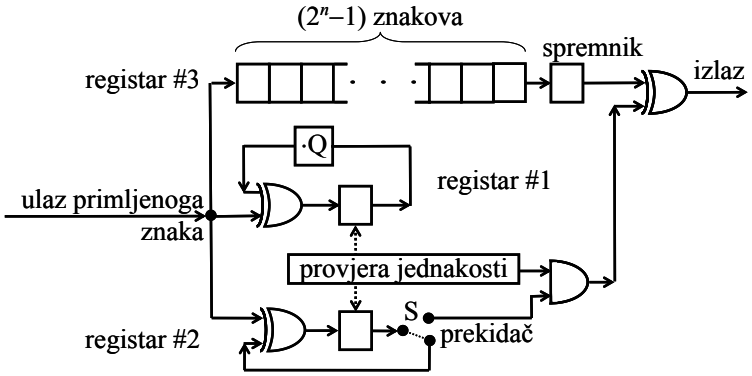
Neka se znakovi iz **m**, **m'** i **e** mogu indeksirati, gdje se znak na lijevoj strani indeksira kao #0. Neka je pogrešan znak #$i$. Neka je uzorak pogreške znaka neki $R_j$, gdje $R$ predstavlja LFSR maksimalne periodičnosti povezan našim kodom (na temelju kojih se oblikovala matrice **Q**). Sada analiziramo učinak posmika **e** u registar #1 na slici 5.18. Počinjemo posmikom prvoga znaka sasvim desno. Sadržaj registra je 0 sve dok se uzorak pogreška posmiče u njega. U ovome trenutku, sadržaj registra je $R_j$.

Registar se sada posmikne dodatnih $i$ puta, sve dok se znak sasvim lijevo ne posmikne u njega. Za vrijeme ovih $i$ posmika, ulazni znakovi su uvijek 0, a *krug stvara uzastopan sadržaj R* (što je posljedica uzastopnih množenja s **Q**). Konačan sadržaj registra je, dakle, $R_{j+i}$. U ovome trenutku,

sadržaj registra #2 je uzorak pogreške znaka $R_j$ (ovaj učinak već se objasnio). Dakle, *jedan registar daje $R_{j+i}$, a drugi daje $R_j$ = uzorak pogreške.*

Cjelovitome postupku ispravke jednoga pogrešnoga znaka, svojstveno je određivanje *uzorka pogreške* i *položaja pogrešnoga znaka.* Prethodno opisan postupak izravno daje uzorak pogreške. Određivanje mjesta pogreške znači određivanje indeksa *i*. Budući da znamo $R_j$ (to je uzorak pogreške) i znamo $R_{j+i}$, moguće je odrediti *i*, napuniti $R_j$ u registar R i napraviti posmik dok se ne dobije uzorak poznat nam kao $R_{j+i}$. Brojenjem posmika dobijemo *i*.

Indeks *i* određuje se još jednim načinom - posmikom registra R, počevši sadržajem $R_{j+i}$, sve dok se ne dobije sadržaj $R_j$. Broj posmika u ovomu slučaju je $2^n–i$. On se temelji na promatranju da je $(j+i) + (2^n–1–i) = j+2^n–1$. Međutim, obzirom na periodičnost R, njegov sadržaj nakon $j+2^n–1$ posmika je $R_j$. Imajte na umu da je sada $2^n–2–i$ mjesto pogrešnoga znaka u **e**, *računajući s desne strane*. To se objašnjava uočavanjem da je bit sasvim desno, bit #($2^n–2$), računajući s lijeva, a bit je #0 računajući s desne strane. Sada smo spremni objasniti strujni krug ispravka pogreške prikazan na slici 5.19.




*Slika 5.19: Cjelovit RS dekoder za ispravak jednoga znaka*

Prijemni blok koji se sastoji od $2^n–1$ znakova, posmiče se u tri registra. Blok se pohranjuje u registru #3, a dva registra (#1 i #2) u donjemu dijelu na slici 5.18, su sindrom-generatori. Tijekom toga posmika, prekidač S zatvara povratnu vezu prema registru #2. Nakon što ga se primi, blok se potpuno pomakne u registar, prekidač S spaja sadržaj registra #2 na ulazu jednih I vrata. U slučaju ako pogreške nema, ovaj sadržaj je "sve 0", a u slučaju ako ona postoji, sadržaj je upravo uzorak pogreške znaka. Imajte na umu da čitav znak ulazi na AND vrata: tj., ova vrata imaju *n*+1 ulaza za pojedine bitove. Broj *n* su znakovi iz registra plus jedan dodatan bit iz kruga "provjere jednakosti (*check for equality*)".

Sada ćemo nastaviti posmik u registre #3 i #1. Dok znakovi ne napuste krug, registar #1 djeluje kao registar R, stvarajući uzastopne sadržaje R što počinju s $R_{j+i}$. Nakon $2^n–1–i$ posmika, sadržaj registra #1 je $R_i$. Registri #2 i #1 onda imaju isti sadržaj pa je izlaz iz kruga "provjere jednakosti" jednak 1. Sadržaj registra #2, koji predstavlja uzorak pogreške, posmiče se u XOR vrata te se râdi operacija XOR nad znakom koji je trenutno pohranjen u spremniku. Odbrojavanje objašnjava da je *ovaj znak pogrešan znak* pa ga operacija XOR ispravlja. Za slučaj u kojemu ne postoji pogreška, izlaz iz I vrata uvijek je 0, a primljena poruka pomiče se nepromijenjena iz registra #3.

*Primjer* Promotrimo slučaj prije obrađenih poruka **m** i **m'**. Nakon što **m'** napravi posmik u krug na slici 5.19, sadržaji registara #1 i #2 su [101] odnosno [001] (vidi tablicu 5.5 i primjer koji joj prethodi). Posmikom u registar #1, njegovi uzastopni sadržaji su [100], [010], [001]. Nakon tri posmika dobivamo jednadžbe između sadržaja dvaju registara. U ovome trenutku pogrešan znak pomaknuo se u spremnik. Pri uvođenju **m'**, opazili smo da se pogrešan znak nalazi tri mjesta brojeći s desne strane. Sada primjenjujemo XOR operaciju nad uzorkom pogreške i ispravljamo ga. Imajte na umu da je broj paritetnih znakova u **m**, potrebnih za ispravak pogreške jednoga znaka, jednak zbroju duljinâ sindrom-generatorâ, a taj zbroj je dva. To jest, od $2^n–1$ znakova u **m**, ima $2^n–3$ informacijska znaka i dva paritetna znaka.

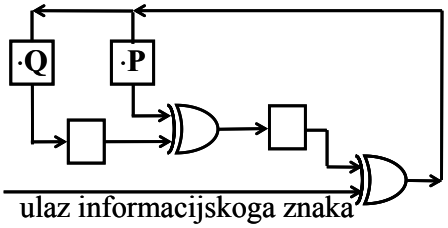
Zaključujemo ovo poglavlje uvođenjem sklopa za kodiranje našega RS koda. Ovdje moramo izgraditi krug koji istovremeno zadovoljava dva nezavisna paritetna ograničenja nametnuta od strane dvaju sindrom-generatora na slici 5.18. Ovaj problem već se obradio u poglavlju 4.4. Ponavljamo ondje

naveden rezultat. Povratne veze LFSR koje odgovaraju dvama LFSR-ima spojenima paralelno, pronađu se koristeći operaciju $\mathbf{v} \cdot \mathbf{m}$, u kojoj **v** predstavlja povratne veze jednoga informacijskoga znaka u LFSR. Retci **m** sastoje se od linearnih posmika vektora koji odgovara povratnoj vezi drugoga LFSR.

U slučaju RS koda, LFSR rukuje cjelokupnim znakovima pa *povratne veze nisu obilježene* s 0 *i* 1, *nego matricama*. Na primjer, povratna veza registra #2, koja se pojavljuje kao vektor [11], zapravo je vektor [**II**], gdje je **I** jediničnu matrica dimenzije $n \times n$. Prema istoj logici, povratne veze registra #1 su [**QI**]. Kako bi pronašli jedan LFSR koji odgovara paralelnome radu tih dvaju registara, onda obavljamo operaciju:

$$[\mathbf{I}\cdot\mathbf{I}] \cdot \begin{bmatrix} \mathbf{Q\,I\,0} \\ \mathbf{0\,Q\,I} \end{bmatrix} = [\mathbf{QPI}] \text{ gdje } \mathbf{P} \text{ označava matricu } \mathbf{I} + \mathbf{Q}.$$

Koder, čije povratne veze odgovaraju strukturi matrice [**QPI**], prikazuje slika 5.20.




*Slika 5.20: RS koder za cjelovit ispravak jednoga znaka*

Kao što se prije navelo, broj informacijskih znakova koji ulaze u koder je $2^n$–3. Konačan sadržaj slijedećega LFSR oblikovat će paritetne znakove pridružene informacijskim znakovima, tvoreći prenesen kodiran blok.

# 14. *Laboratorijska vježba 13: Kodiranje i preplitanje primijenjeno u kompakt diskovima digitalnoga audio sustava*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

13.1. Kodiranje i preplitanje primijenjeno u kompakt diskovima digitalnome audio sustavu

13.1.1. CIRC kodiranje

13.1.1.1. Skraćivanje RS koda

13.1.2. CIRC dekodiranje

13.1.3. Interpolacija i utišavanje

# 6. POGLAVLJE 6 - KONVOLUCIJSKI KODOVI

## 6.1. 6.1. Uvod

U *blok-kodovima*, svaka *m*-torka informacijskih simbola, preko preslika 1:1 (injekcije) definira, kontrolnu skupinu od $n - m$ simbola u kodnoj riječi sustavnoga, odnosno definira kompletnu kodnu riječ (*n*-torku) nesustavnoga koda. Ovakav preslik odgovara sustavu *bez memorije* tako da je izlaz iz sustava (kodna riječ) određen isključivo trenutnim ulazom (*m*-torkom informacijskih simbola).

Za razliku od ovoga, *konvolucijski kod* predstavlja izlaz iz *sustava s memorijom*. Kodna riječ, tj. *n*-torka, ne ovisi dakle samo o trenutnoj ulaznoj *m*-torci, nego i o prethodnim ulaznim *m*-torkama. Na taj način ne postoji jednostavan preslik 1:1 između ulazne skupine od *m* informacijskih simbola i izlazne skupine od *n* kodnih simbola, nego se struktura koda proteže na cio kodiran niz.

Zbog korištenja memorije, konvolucijski kodovi su matematički složeniji. Međutim, praktička rješenja ograničavaju se na linearne, vremenski nepromjenjive sustave, pa se takvi kodovi mogu opisati relacijom *konvolucije*. Posebno za binarne sustave, konvolucija se može ostvariti jednostavnim posmičnim registrima SR (*shift registers*) u sjedinjenju sa zbrajalima po modulu 2.

Zanimljivo je također naglasiti, da su praktička rješenja konvolucijskih kodova ograničena na male vrijednosti *n* i *m* (do 4), za razliku od rješenja u blok-kodovima gdje su *n* i *m* reda veličine 100 i više.

## 6.2. 6.2. Osnovni pojmovi

Obradili smo blok-kodove u kojima se poslana informacija uvijek sastoji od kodnih riječi *unaprijed određene dužine*. Konvolucijski kodovi koriste se u slučajevima ako se informacija prenosi *neprekinutim* nizom bitova *nedefinirane* duljine. Za proučavanje učinka pogrešaka, *tok* bitova razbije se na *prethodno određenim mjestima* gdje se umeću *paritetni bitovi*, kao što prikazuje slika 6.1.

| | |
|---|---|
| Informacija | … *a b c d e f g h i j k* … |
| Primljen signal | … *a b c* $P_0$ *d e* $P_1$ *f g* $P_2$ *h i j* $P_3$ *k* … |

*Slika 6.1: Umetanje paritetnih bitova u poslanu informaciju*

Paritetni bitovi ne postavljaju se nužno na međusobno jednakim udaljenostima, a to prikazuje i slika. Svaki paritetan bit, jednak je zbroju složaja nekoliko prethodnih bitova. Kroz cio postupak kodiranja, paritetni bitovi izgrađeni su na čvrsto propisan način.

Na primjer: $P_0$ jednak je zbroju svoja prethodna tri bita. $P_1$ jednak je zbroju dvaju bitova poslanih 2 i 3 trenutka prije. $P_2$ jednak je zbroju svoja prethodna dva bita. $P_3$ može predstavljati zbroj svojih triju prethodnih bitova, itd.

Postoje brojni konvolucijski kodovi, namijenjeni rješavanju slučajno raspoređenih pogrešaka, prasku pogrešaka, ili oboma. Pogreške se pojavljuju u praskovima, ali između praskova, dopušta se pojava ograničenoga broja pojedinačnih pogrešaka. Za prikaz ispravke praskovitih pogrešaka, koristi se konvolucijski kod poznat i kao kod *dekodiranja difuzijskim pragom* (*diffuse threshold decoding*). Ovaj kod (vidi Laboratorijsku vježbu 14) ima sljedeća zanimljiva svojstva:

1. To je jedan od najpopularnijih, danas korištenih kodova.;
2. Načela na kojima radi, uobičajena su za neke druge kodove. Razumijevanje tih načela, pružit će studentu čvrstu podlogu za daljnja proučavanja.;
3. Kod je jednostavan za razumijevanje.

Parametri jednostrukih blok-kodova za ispravak praskovitih pogrešaka su: duljina praska *t* i duljina bloka *n*. U bloku duljine *n*, možemo ispraviti do *t* pogrešaka, uz uvjet da su pogreške ograničene na, ne više od *t* mjesta u prasku unutar bloka. Blok-kodovi za ispravak praskovitih pogrešaka koje smo proučili, imaju ograničenje. U svakome bloku dopuštaju se samo *jednostruke praskovite* pogreške.

Za konvolucijske kodove, pojam "*blok*" ne postoji. Za njih se zahtijeva da, između dvaju uzastopnih praskova, mora postojati područje bez pogrešaka ili *područje oporavka* (*recovery region*), jer se sustav mora oporavi od učinka praskovitih pogrešaka prije no što bude u stanju obraditi sljedeći prasak.

Minimalna veličina područja oporavka ovisi o veličini praska što ga se ispravlja. U slučaju koda za dekodiranje raspršenim (*diffuse*) pragom, minimalna veličina područja oporavka je $3t + 2$, gdje je $t$ maksimalna duljina praskovite pogreške koju se ispravlja.

## 6.3. 6.3. Parametri konvolucijskih kodova

Izlazna kodna riječ iz konvolucijskoga kodera definira se trenutnom informacijskom $m$-torkom, ali i prethodnim $m$-torkama. Koliko prethodnih $m$-torki ima utjecaj na trenutnu kodnu riječ ovisi o broju stupnjeva D bistabila u koderu, odnosno o veličini memorije sustava $q + 1$. Za svaku novo učitanu $m$-torku simbola na ulazu, koder daje na izlazu odgovarajuću $n$-torku, tako da se *koeficijent prijenosa* definira kao i u blok-kodovima $\rho = m/n$.

Konvolucijskim kodovima obično se pridružuju tri parametra ($n$, $k$, $m$).

$n$ … broj izlaznih bitova

$k$ … broj ulaznih bitova

$m$ … broj memorijskih registara

Iznos $k/n$ naziva se *brzina koda*. To je mjera učinkovitosti koda. Obično su $k$ i $n$ parametri u rasponu od 1 do 8, $m$ od 2 do 10, a brzina koda od 1/8 do 7/8, osim aplikacije napravljenih za svemirski prostor gdje je brzina koda malena (koristi se 1/100 ili čak i veći omjer).

Pored toga što $k$ definira minimalnu dužinu kodne riječi, kod se može ograničiti i na maksimalan broj bitova $L$, čime se potpuno definiraju tzv. ($k$, $L$) kodovi ograničene dužine niza RLL (*run-length-limited*) kodovi. Često proizvođači integriranih sklopova (*chips*) za konvolucijske kodove navode parametre koda ($n$, $k$, $L$). Veličina $L$ označava *duljinu ograničenja* (*constraint length*) koda i definira se izrazom:

$$L = k(m - 1), \tag{6.1}$$

a broj stanja konvolucijskoga kod(er)a jednak je $2^{k(L-1)}$. Duljinu ograničenja $L$, predstavlja *broj bitova* u *memoriji kodera* koji utječu na *stvaranje* $n$ izlaznih bitova. Duljina ograničenja $L$ također se označava velikim slovom $K$, što se može pobrkati s malim slovima $k$, a on predstavlja broj ulaznih bitova. U nekim knjigama $K$ je jednak umnošku $k \times m$.

Često, u trgovačkim (komercijalnim[98]) detaljnim opisima (specifikacijama[99]), kodovi su određeni s ($r$, $K$), gdje je $r$ brzina koda $k/n$, a $K$ je *duljina ograničenja*. Duljina ograničenja $K$, također je jednaka $L$–1, kao što se definira i u ovoj skripti. U nastavku, konvolucijski kodovi označit će se kao ($n$, $k$, $m$), a ne kao ($r$, $K$) kodovi.

### 6.3.1. 6.3.1. PARAMETRI KODA I STRUKTURA KONVOLUCIJSKOGA KODA

#### 6.3.1.1. 6.3.1.1. Posmični registar

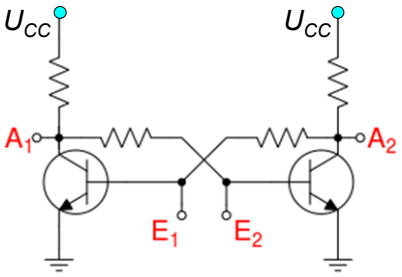
Flip-flop sklop dijeli se na: *jednostavan* (transparentan[100] ili asinkron) *sinkrono okidan*. Jednostavan flip-flop, obično se naziva *sprega* (*latch*). Riječ *sprega* uglavnom se koristi za naznačiti sklopove skladištenja, dok se *okidani* uređaji opisuju kao flip-flop. Sprega je osjetljiva na razinu, a flip-flop osjetljiv je na rub (uspon ili pad). Ako se spregnu, postaju transparentni, a flip-flop mijenja izlaz samo jednom vrstom (porastom ili padom) brida okidnoga takta. Tradicijski, flip-flop sklop temelji se na bipolarnim tranzistorima (slika 6.2).

---

[98] *komercijalan* … (lat. *commercialis*) trgovački, obrtni, privredni, prometni; društveni; komercijalna ulica trgovačka ulica; komercijalne igre društvene igre; komercijalni sustav državne privredne politike koji povlašćuju trgovinu, osobito na račun poljoprivrede; komercijalni traktat trgovinski ugovor

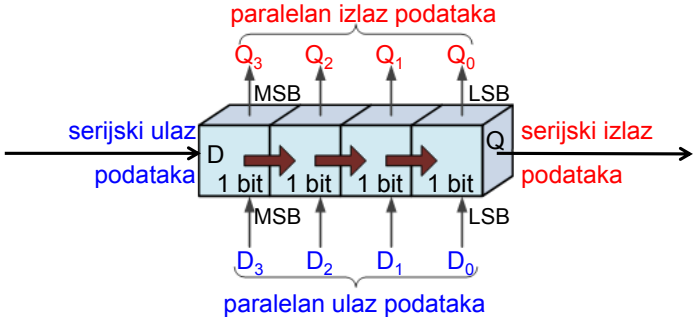
[99] *specifikacija* … (lat. *specificatio*) nabrajanje pojedinosti, podroban (detaljan) opis, popis svih pojedinačnih predmeta koji spadaju zajedno; posebna oznaka

[100] *transparentan* … (lat. *trans-parere*) proziran, bistar, jasan




*Slika 6.2: Flip-flop sklop napravljen bipolarnim tranzistorima*

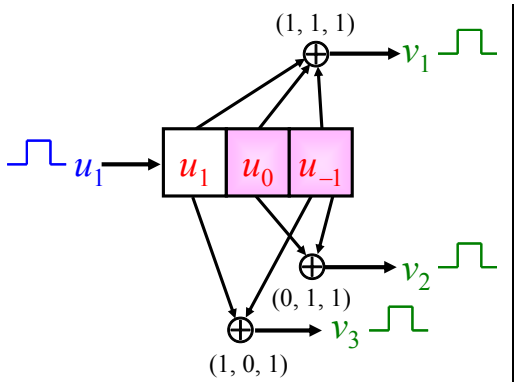
Posmični registar još je jedna vrsta strujnoga kruga serijske logika koja se koristiti za skladištenje ili prijenos podataka u obliku binarnih brojeva "posmikom" podataka, u svakome taktu za jedan. Odatle i dolazi naziv "posmični registar" (slika 6.3).




*Slika 6.3: Posmični registar ostvaren nizom od 4 bistabila*

On se u osnovi sastoji od nekoliko podatkovno spregnutih bistabila D vrste "od jednoga bita". Skup D bistabila (po jedan za svaki bit), spaja se serijski tako da izlaz iz jedne podatkovne sprege postaje ulaz u sljedeću spregu i tako dalje.

Strukturu konvolucijskoga kodera lako je nacrtati iz njegovih parametara. Prvo nacrtamo *m* okvira koji predstavljaju *m* registara. Zatim nacrtamo *n* zbrajala modulo 2 (EXOR) za predstaviti *n* izlaznih bitova. Sada spojimo registre na zbrajala prema izrazu *polinom-generatora* kao što prikazuje slika 6.4.




$$v_1 = (u_1 + u_0 + u_{-1}) \bmod 2$$

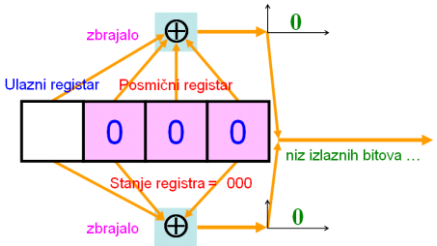
$$v_2 = (u_0 + u_{-1}) \bmod 2$$

$$v_3 = (u_1 + u_{-1}) \bmod 2$$

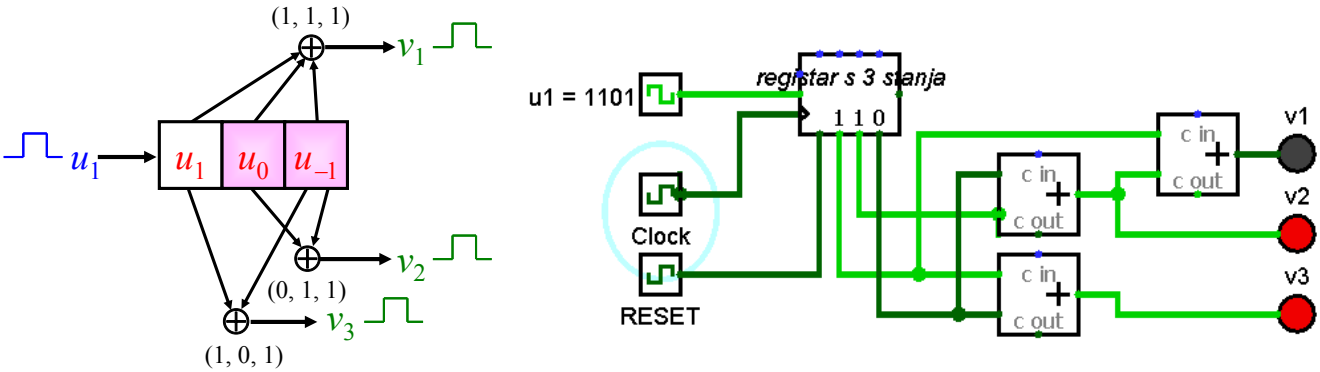
$$L = k(m - 1) = 2$$

*Slika 6.4: Ovaj (3, 1, 3) konvolucijski kod ima 3 registara, 1 ulazni bit i 3 izlazna bita.*

Zoran prikaz rada konvolucijskoga kodera sadrži sljedeća animacija:

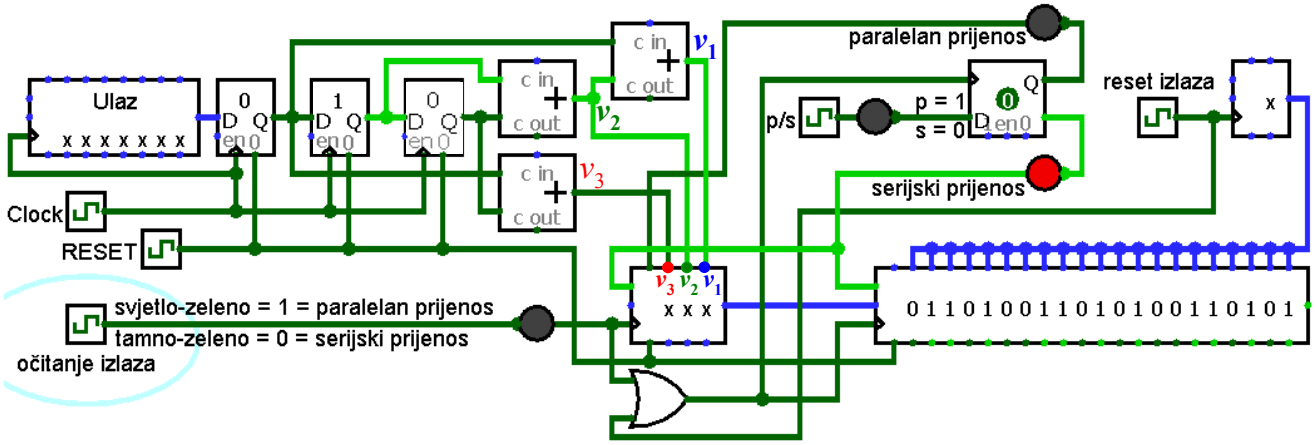



Na slikama 6.5 i 6.6 prikazuje se kod brzine 1/3.




a) izvedbena shema b) izvedba simulacijskim programom LOGISIM-win-2.7.1

*Slika 6.5: Kodiranje (2, 1, 3) koderom*




*Slika 6.6: Kodiranje (2, 1, 3) koda simulacijskim programom LOGISIM-win-2.7.1*

Svaki *ulazni* bit kodira se u 3 *izlazna* bita. *Duljina ograničenja koda* je 2. Tri izlazna bita proizvode 3 zbrajala modulo-2 zbrajanjem odgovarajućih bitova u memorijskim registrima. *Obrazac*, koji će se bitovi zbrojiti da bi proizveli izlazni bit, zove se *polinom-generator* (*g*) za taj izlazni bit. Na primjer, prvi izlazni bit ima polinom-generator (1, 1, 1). Izlazni bit 2 ima polinom-generator (0, 1, 1), a treći izlazni bit ima polinom (1, 0, 1). Izlazni bitovi samo su zbroj EXOR ovih bitova.

$$v_1 = (u_1 \oplus u_0 \oplus u_{-1}) \text{ mod-2}$$

$$v_2 = (u_0 \oplus u_{-1}) \text{ mod-2}$$

$$v_3 = (u_1 \oplus u_{-1}) \text{ mod-2}$$

*Polinomi* rezultiraju *kodom* prema svojoj *jedinstvenoj kakvoći zaštite od pogreške*. Jedan (3, 1, 4) kod može imati sasvim različita svojstva u odnosu na neki drugi kod, ovisno o odabranim polinomima.

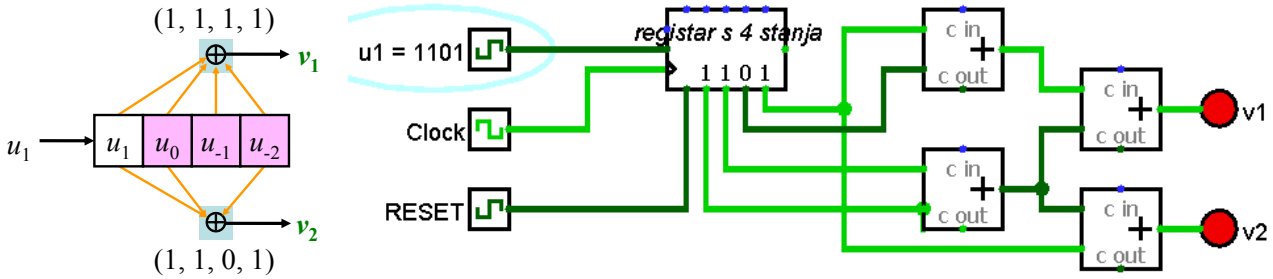
### *6.3.1.2. 6.3.1.2. Kako se odabiru polinomi?*

Postoji mnogo izbora za polinome koda bilo kojega reda *m*. Svi oni ne moraju proizvesti nizove, dobro zaštićene od pogrešaka. Knjiga Petersena i Weldona sadrži cjelovit popis tih polinoma. Dobri polinomi koji se nalaze u ovome popisu, pronađeni su računalnom simulacijom. Popis dobrih polinoma za kodove brzine 1/2 dan je u tablici 6.1.

*Tablica 6.1 Polinom-generatori koje je pronašao Busgang za dobre "kodove" brzine 1/2*

| Duljina ograničenja | Zbrajalo G1 | Zbrajalo G2 | Duljina ograničenja | Zbrajalo G1 | Zbrajalo G2 |
|---|---|---|---|---|---|
| 3 | 110 | 111 | 7 | 110101 | 110101 |
| 4 | 1101 | 1110 | 8 | 110111 | 1110011 |
| 5 | 11010 | 11101 | 9 | 110111 | 111001101 |
| 6 | 110101 | 111011 | 10 | 110111001 | 1110011001 |

Znamo da postoje stanja našega uma pa to preslikamo i na kodere. Jedan dan smo depresivni, a već sljedećega dana možda sretni, a to su različita stanja u kojima se možemo nalaziti. Naš izlaz ovisi o našim stanjima uma, a to se odražava "izrazom lica" pa možemo kazati da i koderi rade na ovaj način. Ono što im je izlaz, ovisi o tome kakvo je stanje njihova "uma". Naša stanja su složena, ali stanja kodera samo je niz bitova. Profinjeni koderi imaju duge ograničene duljine, a neki drugi jednostavno imaju kratke, pokazujući broj stanja u kojima se mogu nalaziti. Kod (2, 1, 4) na slici 6.7 ima ograničenu duljinu 3.




*Slika 6.7: Stanja kodera predstavljaju sadržaji registara*

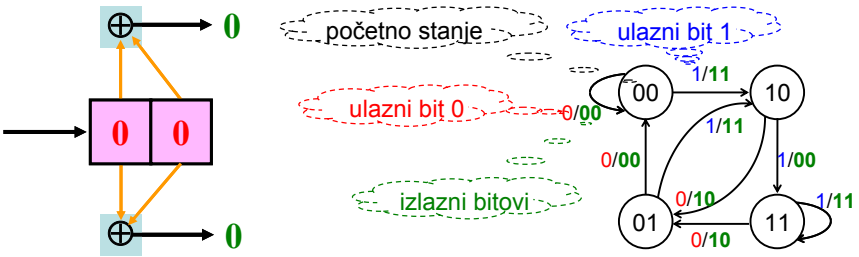
Zasjenjeni registri sadrže ove bitove. Ne-zasjenjen registar sadrži dolazni bit (ali i on je u ovome primjeru uključen u stvaranje izlaznoga niza). Teorijski, to znači da 3 bita ili 8 različitih složaja ovih bitova može postojati u registarskim memorijama. Ovih 8 različitih složaja određuju kakav ćemo izlaz dobiti za kodirane nizove $v_1$ i $v_2$. Složaji bitova u zasjenjenim registrima zovu se *stanja koda* i definiraju se kao

$$\text{broj stanja} = 2^L \tag{6.2}$$

gdje je $L$ … ograničena duljina koda i jednaka je

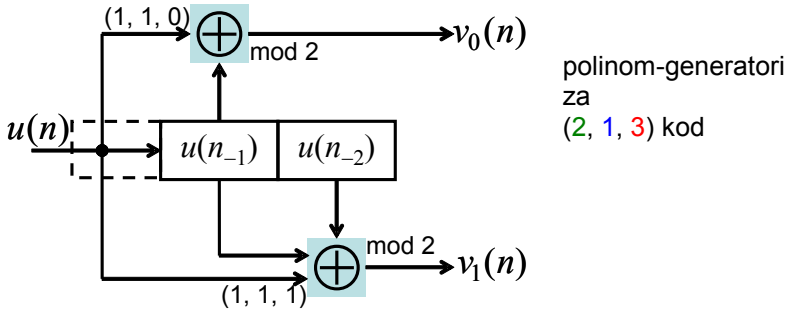
$$L = k(m - 1). \tag{6.3}$$

Zamislimo stanje kao svojevrstan početni uvjet. Izlazni bit ovisi o tome početnome stanju koje se *mijenja u svakome vremenskom odsječku*. Ispitajmo stanja (2, 1, 4) koda prikazanoga na slici 6.7. Za svaki ulazni bit, ovaj kod šalje 2 bita na izlaz. To je kod brzine 1/2. Njegova duljina ograničena je na 3. Ukupan broj *stanja* iznosi 8. Osam stanja ovoga (2, 1, 4) koda su: 000, 001, 010, 011, 100, 101, 110, 111. Slika 6.8 opisuje oznake na dijagramu stanja za koder od 2 stanja, za (2, 1, 2) kod.




*Slika 6.8: Opis oznaka na dijagramu stanja za (2, 1, 2) kod.*

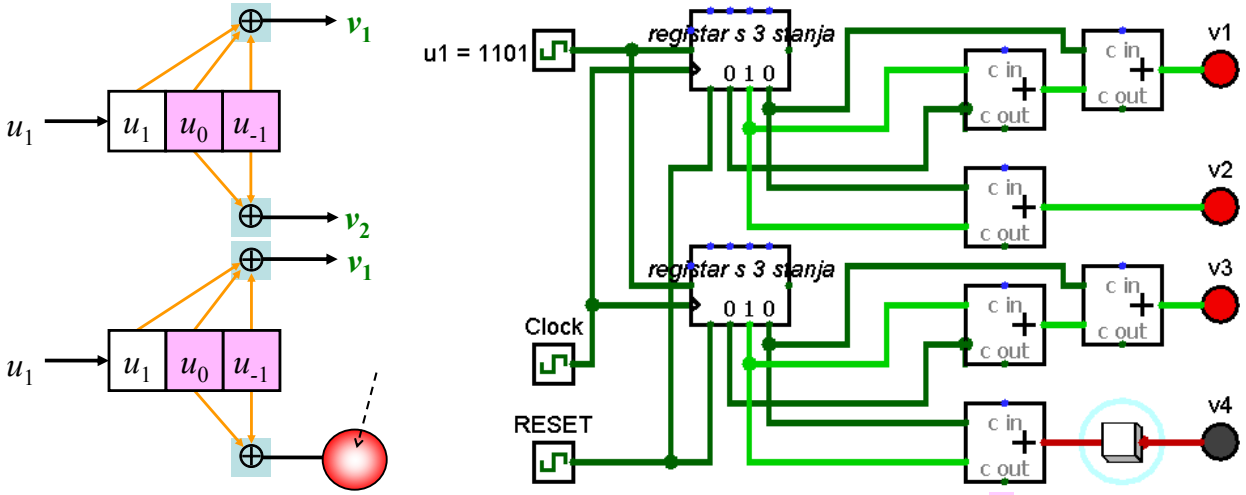
Slika 6.9 dodatno opisuje oznake na dijagramu stanja (*polinom-generatori*).




*Slika 6.9: Pogled na blok-dijagram konvolucijskoga kodiranja (2, 1, 3) s posmičnim registrima.*

[102] *punctured* … prekinut, (ras)puknut, probušen, okljaštren, podrezan

U posebnome slučaju kada je $k = 1$, kodovi brzina 1/2, 1/3, 1/4, 1/5, 1/7, ponekad se nazivaju *matični kodovi*. Možemo sjediniti pojedine ulazne kodne bitove za proizvesti *probušene kodove* (*punctured codes*)[102] koji nam daju brzina koda, različitu od $1/n$. Korištenjem zajedno 2 dvo-brzinska koda od 1/2 kao što prikazuje slika 6.10, a zatim ne slanjem jednoga od izlaznih bitova, možemo pretvoriti ovu brzinu iz 1/2 u brzinu 2/3 koda, 2 dolazna i 3 odlazna bita (a ne 4, koliko ih je raspoloživo).

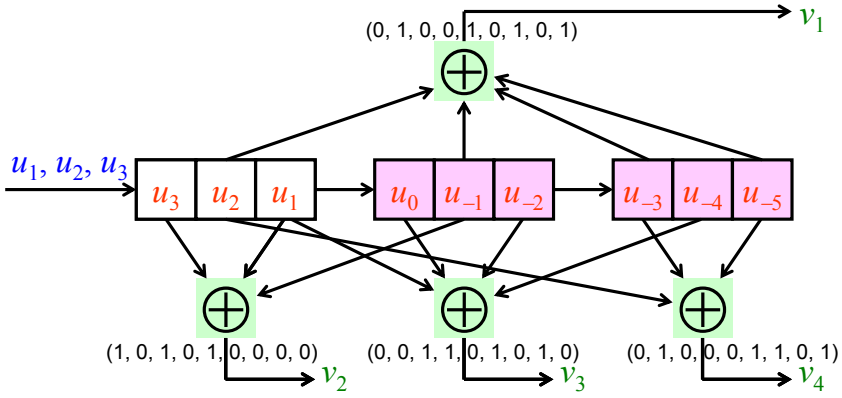



*Slika 6.10: Dva (2, 1, 3) konvolucijska koda proizvode 4 izlazna bita. Bit broj $v_2$ je "probušen", tako da je složaj, učinkovit (3, 2, 3) kod.*

Ovaj koncept zove se *bušenje* (*puncturing*). Na prijemnoj strani, dodatni bitovi (što ne utječu na mjeru dekodiranja) umeću se na odgovarajuća mjesta prije dekodiranja. Ova tehnika omogućuje proizvesti kodove različitih brzina korištenjem samo jednoga jednostavnoga sklopa. Iako možemo izravno ustrojiti kod brzine 2/3, prednost "podrezanoga" koda je da se brzine mogu dinamički (programski) mijenjati, ovisno o stanju kanala, kao što je npr. utjecaj atmosferskih smetnji i sl. Čvrsta provedba, iako je jednostavnija, ne dopušta takvu dinamiku.

### 6.3.1.5. 6.3.1.5. Struktura koda za $k > 1$

Naizmjence možemo stvoriti kodove gdje je $k > 1$ bita kao što je npr. (4, 3, 3) kod (slika 6.11).




[duljina ograničenja je: $L = k\,(m-1) = 3 \cdot (3-1) = 6$]

*Slika 6.11: Ovaj (4, 3, 3) konvolucijski kod ima $3 \times 3 = 9$ memorijskih registara, 3 ulazna i 4 izlazna bita. Osjenčani registri sadrže "stare" bitove predstavljajući trenutno stanje.*

Ovaj kod ima 3 ulazna i 4 izlazna bita. Broj registara je 3. Duljina registara ograničena je na $3 \times 2 = 6$ stanja. Kod ima 64 stanja. Ovomu kodu trebaju polinom generatori 9-toga reda. Zasjenjeni kvadrati predstavljaju ograničenu duljinu.

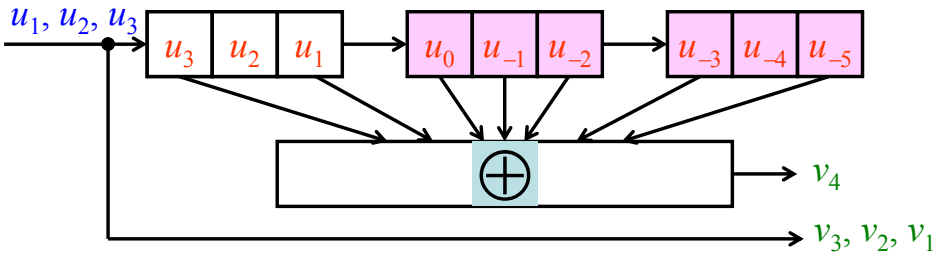
Postupak za izradu strukture ($n$, $k$, $m$), gdje je $k$ broj veći od 1, je kao što slijedi. Prvo nacrtamo $k$ skupova od $m$ kvadrata. Zatim nacrtamo $n$ zbrajala. Sada spojimo $n$ zbrajala na memorijske registre

[102] *punctured* … prekinut, (ras)puknut, probušen, okljaštren, podrezan

koristeći koeficijente odgovarajućega polinoma *n*-toga (*k*×*m*) stupnja. Ono što ćemo dobiti je struktura (4, 3, 3) koda kao ona što ju prikazuje slika 6.11.

#### 6.3.1.6. 6.3.1.6. Sustavan i nesustavan kod

Poseban oblik konvolucijskoga koda u kojemu izlazni bitovi sadrže *lako prepoznatljiv niz ulaznih bitova* naziva se *sustavan oblik.* Sustavnu inačicu gornjega (4, 3, 3) koda prikazuje slika 6.12.




*Slika 6.12: Sustavna inačica konvolucijskoga (4, 3, 3) koda.*

Sustavna inačica ima *isti broj memorijskih registara*, *3 ulazna* i *4 izlazna bita*. Izlazni bitovi sastoje se od *3 izvorna bita* i *4-toga* "*paritetnoga*" *bita*. Od 4 izlazna bita, 3 su ista kao i 3 ulazna bita. Četvrti bit, vrsta je paritetnoga bita kojega tvori sastav triju bitova koristeći polinom-generator. Radije se koriste sustavni u odnosu na ne-sustavne kodove, jer su pregledni. Oni također zahtijevaju manje opreme za kodiranje. Drugo važno svojstvo sustavnih kodova jest da nisu "katastrofični[103]", što znači da se pogreške ne mogu prostirati na katastrofičan način. Sva ta svojstva čine ih veoma poželjnima. Sustavni kodovi također se koriste u *modulaciji kodirane rešetke* TCM (*Trellis Coded Modulation*). Svojstva zaštite od pogrešaka sustavnih kodova, međutim ista su kao ona za ne-sustavne kodove.

### 6.3.2. 6.3.2. KODIRANJE DOLAZNOGA NIZA

Izlazni niz **v**, može se izračunati *konvolucijom* (*) ulaznoga niza **u** i impulsnoga odziva *g*. To se može izraziti kao

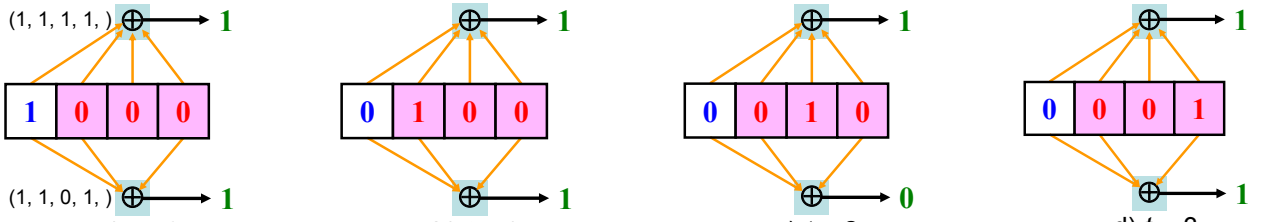
$$\mathbf{v} = \mathbf{u} * g \tag{6.4}$$

ili u općenitijem obliku

$$v_l^j = \sum_{i=0}^{m} u_{l-i} g_i^j \tag{6.5}$$

gdje je $v_l^j$ izlazni bit *l* kodera *j*, $u_{l-i}$ je ulazni bit, a $g_i^j$ je *i*-ti izraz u polinomu *j*.

Kodirajmo niz od dva bita 1 i 0 (2, 1, 4) kodom te pogledajmo kako postupak radi. Prvo ćemo propustiti jedan bit "1" kroz ovaj koder kao što prikazuje slika 6.13.




*Slika 6.13: Prolaz kroz koder niza bitova od samo jedne 1. Jedan bit stvara izlaz od osam bitova.*

[103] *katastrofa* … (grč. *katastrofe*) preokret, prevrat, obrat, odlučan trenutak; u drami: odlučan događaj koji dovodi do razrješenja zapleta (usp. *epistaza* i *katastaza*); svaki odlučujući, posebice nesretan obrat; težak (ili nesretan) događaj, propast

a. U trenutku $t = 0$, vidimo da je početno stanje kodera "sve 0" (bitovi u desnome dijelu $L = 3$ registarskih mjesta). Ulazni bit 1 uzrokuje pojavu dvaju bitova 11 na izlazu. Kako smo to izračunali? Zbroj svih bitova u registrima po modulu 2 za prvi bit (gornji izlaz) i zbroj modulo 2 triju bitova za drugi izlazni bit (donji izlaz) prema koeficijentima polinoma.

b. U trenutku $t = 1$, ulazni bit 1 pomiče se desno u registar. Ulazni registar sada je prazan i puni se bitom 0 za ispiranje. Koder je sada u stanju 100. Izlazni bitovi sada su opet 11 prema istome izračunu.

c. Ulazni bit 1 pomiče se opet u desno. Sada je koder u stanju 010 pa se drugi bit za ispiranje seli u ulazni registar. Izlazni bitovi su sada 10.

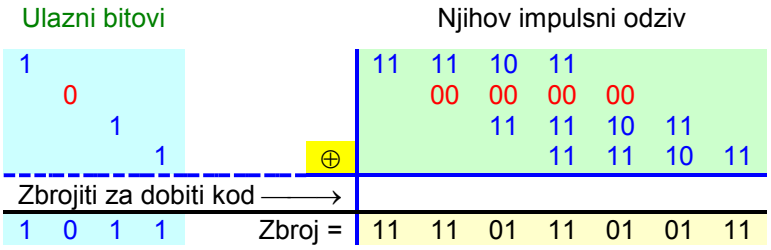
d. U trenutku $t = 3$, ulazni bit seli se u posljednji registar i stanje ulaza je 001. Sada su izlazni bitovi 11.

e. U trenutku $t = 4$, ulazni bit 1 potpuno je prošao kroz koder i koder se "isprao" u stanje "sve 0" te je spreman za prijem sljedećega niza.

### 6.3.3. 6.3.3. IMPULSNI ODZIV KODERA

Imajte na umu da jedan bit proizvodi izlaz od 8 bitova, iako je nominalna brzina koda 1/2. Ovo pokazuje da je za male nizove prekoračenje znatno veće od nominalne brzine, što inače vrijedi samo za duge nizove. Ako učinimo istu stvar bitom vrijednosti 0, dobit ćemo niz od 8 bitova "sve 0". Ono što smo upravo proizveli zove se *impulsni odziv* ovoga kodera, a to je niz [11 11 10 11]. Slično, odziv na ulazni bit 0 ima impulsni odziv [00 00 00 00] (nije prikazan, ali je rezultat očigledan).

Konvolucija[104] ulaznoga niza kodnim polinomom, proizvodi ova dva izlazna niza, što je razlog zašto se ti kodovi zovu *konvolucijski kodovi.* Prema načelu *linearne superpozicije*, sada možemo proizvesti kodiran niz iz navedenih dvaju impulsnih odziva kao što slijedi. Pretpostavimo da imamo ulazni niz [1011] i želimo znati što će postati kodiran niz. Možemo *izračunati izlaz* samo dodavanjem *pomaknute inačice pojedinih impulsnih odziva* (slika 6.14).




*Slika 6.14: Jedinični impulsni odzivi kodera na 0 odnosno 1*

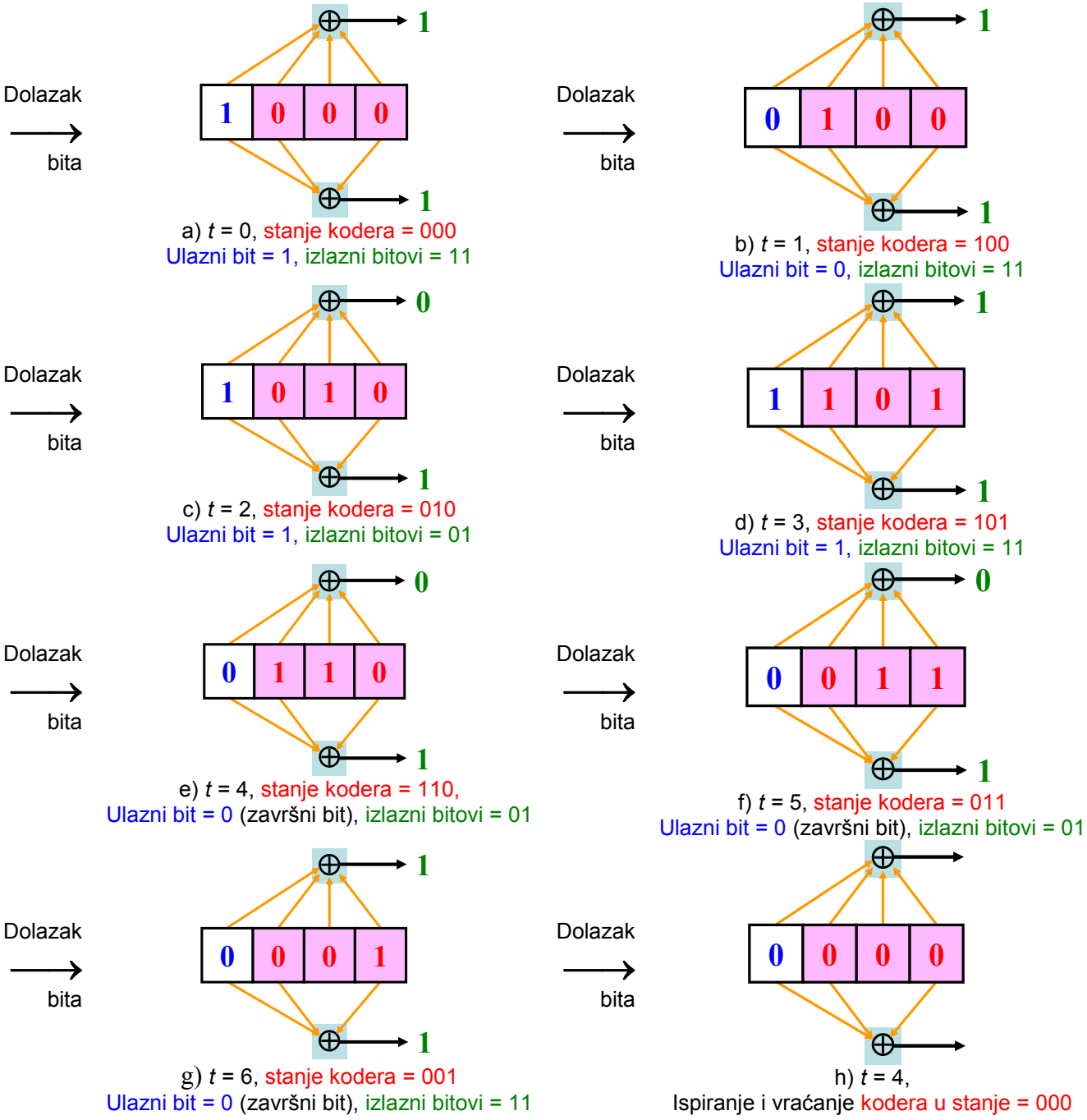
Dobili smo odziv na niz [1011] dodavanjem pomaknutih inačica odziva na 1 odnosno na 0. Rezultat kodiranja, u svakome vremenskome odsječku, prikazuje tablica 6.2.

*Tablica 6.2 - Izlazni bitovi i bitovi kodera za (2, 1, 4) kod - ulazni bitovi su: [1011000]*

| Vrijeme | Ulazni bit | Izlazni bitovi | Stanje registara |
|---|---|---|---|
| 0 | 1 | 11 | 000 |
| 1 | 0 | 11 | 100 |
| 2 | 1 | 01 | 010 |
| 3 | 1 | 11 | 101 |
| 4 | 0 | 01 | 110 |
| 5 | 0 | 01 | 011 |
| 6 | 0 | 11 | 001 |

Na slici 6.15, pomoću kodera za provjeru, ručno smo postavili niz [1011] i dobili smo isti odziv.

[104] Konvolucija je integral što opisuje veličinu preklapanja jedne krivulje $g$ drugom krivuljom $f$ dok se jedna pomiče preko druge (Vidi http://mathworld.wolfram.com/Convolution.html).




*Slika 6.15: Kodiranje (2, 1, 4) kodom*

Ovo pokazuje da je konvolucijski model ispravan. Kodirani niz je [**11 11 01 11 01 01 11**].

# 15. *Laboratorijska vježba 14: Kodiranje, dekodiranje i ispravak pogrešaka difuzijskim pragom*

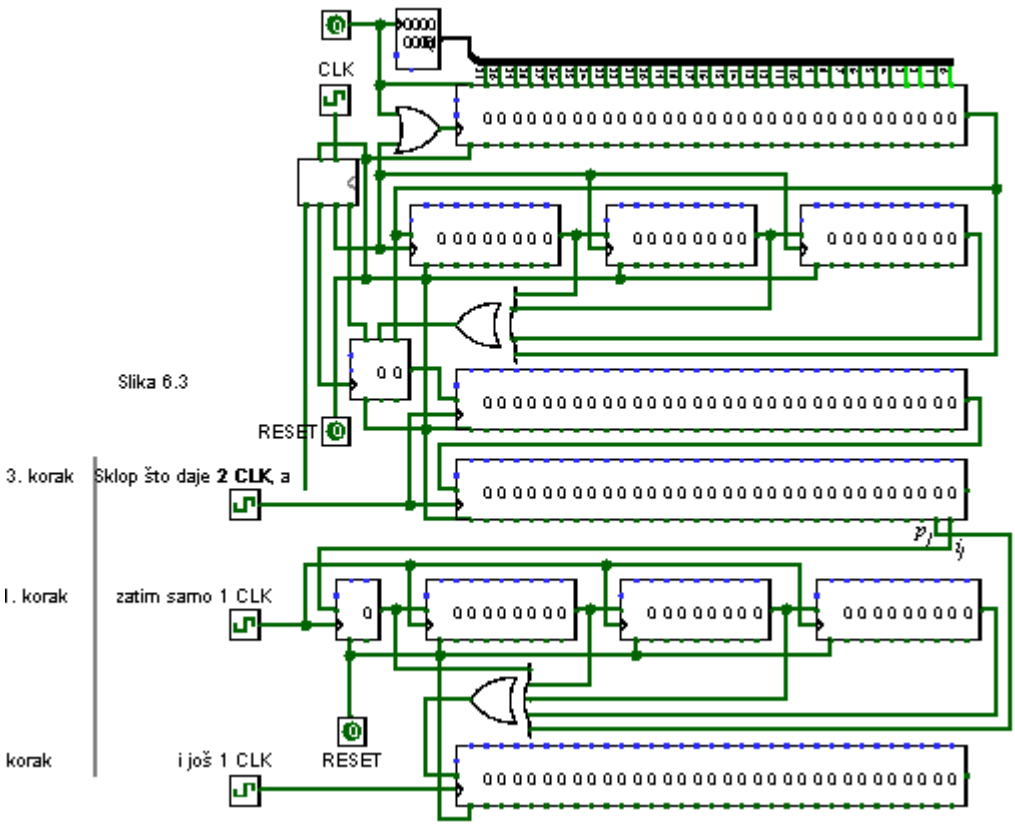
Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

14. LABORATORIJSKA VJEŽBA 14: KODIRANJE I DEKODIRANJE KONVOLUCIJSKIH KODOVA DIFUZIJSKIM PRAGOM

14.1. Postupak kodiranja

14.2. Postupci dekodiranja i ispravke pogrešaka

Slika 6.2




Slika 6.3

## 6.4. 6.4. Oblikovanje kodera

### 6.4.1. 6.4.1. PREGLEDNA TABLICA

Dvije metode u prethodnome poglavlju pokazuju matematički što se događa u koderu. Sklop kodera je puno jednostavniji, jer koder nije matematika. Koder za konvolucijski kod koristi *preglednu tablicu* za kodiranje. Pregledna tablica sastoji se od četiri stavke.

1. Ulazni bit.
2. Stanje kodera. To je jedno od 8 mogućih stanja za primjer (2, 1, 4) koda
3. Izlazni bitovi. Za (2, 1, 4) kod izlazi su 2 bita, a izbori su 00, 01, 10, 11.
4. *Stanje izlaza*, To je *ulazno stanje* za sljedeći bit.

Za (2, 1, 4) kod zadan polinomima na slici 6.15, stvorena je sljedeća pregledna tablica 6.3.

*Tablica 6.3 Pregledna tablica za koder (2, 1, 4) koda*

| ulazni bit | | prije posmika | prijelaz | poslije posmika | | izlazni bitovi |
|---|---|---|---|---|---|---|
| i1 | | s1s2s3 | iz … u | s1s2s3 | | o1o2 |
| 0 | | 0 0 0 | ⟶ | 0 0 0 | | 00 |
| 1 | | 0 0 0 | ⟶ | 1 0 0 | | 11 |
| 0 | | 0 0 1 | ⟶ | 0 0 0 | | 11 |
| 1 | | 0 0 1 | ⟶ | 1 0 0 | | 00 |
| 0 | | 0 1 0 | ⟶ | 0 0 1 | | 10 |
| 1 | | 0 1 0 | ⟶ | 1 0 1 | | 01 |
| 0 | | 0 1 1 | ⟶ | 0 0 1 | | 01 |
| 1 | | 0 1 1 | ⟶ | 1 0 1 | | 10 |
| 0 | | 1 0 0 | ⟶ | 0 1 0 | | 11 |
| 1 | | 1 0 0 | ⟶ | 1 1 0 | | 00 |
| 0 | | 1 0 1 | ⟶ | 0 1 0 | | 00 |
| 1 | | 1 0 1 | ⟶ | 1 1 0 | | 11 |
| 0 | | 1 1 0 | ⟶ | 0 1 1 | | 01 |
| 1 | | 1 1 0 | ⟶ | 1 1 1 | | 10 |
| 0 | | 1 1 1 | ⟶ | 0 1 1 | | 10 |
| 1 | | 1 1 1 | ⟶ | 1 1 1 | | 01 |

Ova pregledna tablica u potpunosti opisuje (2, 1, 4) kod. Ona je različita za svaki kod, ovisno o parametrima i korištenim polinomima.

Na WEB stranici: http://www.invocom.et.put.poznan.pl/~invocom/C/P1-7/en/P1-7/index.htm

prikazuje se niz animacije kako se stvaraju konvolucijski kodovi, a namijenjene su boljemu razumijevanju proučavane problematike konvolucijskoga kodiranja.

Postoje tri grafička načina analize kodera za bolje razumijevanje njegovoga rada. To su:

1. Dijagram stanja

Stanje kodera definira se kao sadržaj stanja njegovih registara. Dijagram stanja sadrži iste informacije kao i pregledna tablica, a prikaz je grafički. Prikladno je ispratiti animacijski dijagram stanja.pps.

2. Dijagram stabla

Za konvolucijske kodove, dijagram stabla prikazuje sve moguće informacije i kodirane nizove. Zadavanjem ulaznoga niza, kod(ira)ni nizovi mogu se izravno pročitati iz grana stabla. Idući dublje u grane stabla, dijagram stabla pokazuje protok vremena. U svakome vremenskome trenutku, izravno se prikazuju svaki (i svi) prijelazi između stanja.

3. Dijagram rešetke.

Stvaran kodiran niz može se prikazati putanjama između stanja u dijagramu rešetke. Os *x* prikazuje diskretno vrijeme, a os *y* prikazuje sva moguća stanja. Ovakav dijagram upućuje nas kako dekodirati primljen kodiran niz. Ako se primljen niz (zbog pogrešaka) ne uklapa u dijagram, onda pri dekodiranju moramo odabrati putanju koja je najbliža primljenome nizu te tako odabrati ispravljen niz.

### 6.4.2. 6.4.2. DIJAGRAM STANJA

Dijagram stanja za (2, 1, 4) kod prikazuje slika 6.16.

Pune (plave) crte označavaju dolazak 1.

Crtkane (crvene) crte označavaju dolazak 0.
Svaka kružnica predstavlja stanje.

U bilo kojemu trenutku, koder boravi u jednome od tih stanja.

Crte prema stanju i od stanja pokazuju stanja prijelaza koja su moguća kako bitovi pristižu.

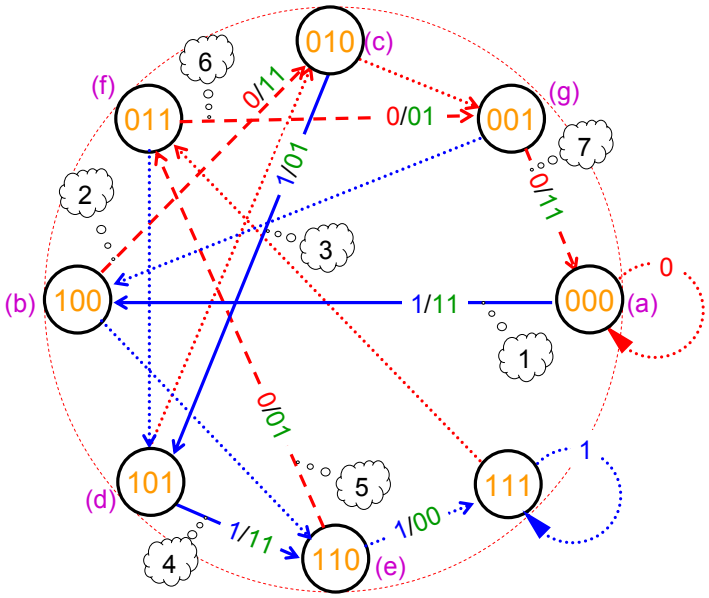
U svakome vremenskome trenutku može doći ili bit 1 ili bit 0.

Svaki od tih dvaju događaja omogućuje koderu skok u različito stanje.

Dijagram stanja nema vrijeme kao dimenziju i stoga stremi intuicijskome razmatranju.

Za svako stanje, izlazni bitovi prikazani su uz crtu

Strelice označavaju prijelaz u novo stanje




(…\All about Digital Modulation\Dijagram stanja.ppt)

*Slika 6.16: Dijagram stanja (2, 1, 4) koda*

Svaka kružnica predstavlja stanje. U bilo kojemu trenutku, koder boravi u jednome od tih stanja. Kako bitovi pristižu, crte prema stanju i od stanja, pokazuju moguće stanje prijelaza. Samo dva događaja moguća su u svakome trenutku, dolazak bita 1 ili dolazak bita 0. Svaki od tih dvaju događaja omogućuje koderu skok u novo stanje. Dijagram stanja nema vremensku dimenziju pa stremi intuicijskome razmatranju.

Usporedimo gornji dijagram stanja i preglednu tablicu kodera. Osim što je prikaz grafički, dijagram stanja sadrži iste informacije kao i pregledna tablica. Pune crte pokazuju dolazak 0, a isprekidane crte prikazuju dolazak 1. Izlazni bitovi u svakome slučaju, prikazani su uz crtu, a strelica označava prijelaz iz staroga stanja u novo stanje. Još jednom vratimo se ideji stanja. O tome gdje ste (stanje) određuje se na koliko načina možete putovati (izlaz). Neka stanja kodera dopuštaju izlaze 11 i 00, a neka dopuštaju samo stanje 01 i stanje 10. Nema stanja koje dopušta sve četiri mogućnosti.[105]

Kako možemo kodirati niz [1011] koristeći dijagram stanja (vidi i prati sliku 6.16)?

- Počnimo stanjem [000] (*a*). Dolazak bita 1 daje izlaz 11 i vodi nas u stanje [100] (*b*).
- Dolazak idućega bita 0, šalje na izlaz bitove 11 i vodi nas u stanju [010] (*c*).
- Dolazak idućega bita 1, šalje na izlaz bitove 01 i vodi nas u stanju [101] (*d*).
- Dolazak posljednjega bita 1 šalje na izlaz bitove 11 i vodi nas u stanje [110] (*e*). Tako sada imamo na izlazu prikupljen niz bitova [11 11 01 11]. No, ovo nije kraj. Moramo dovesti koder natrag u stanje "sve 0".
- Idući dolazni bit je 0 pa se iz stanja [110], ide u stanje [011], a na izlazu se pojavljuju bitovi 01 i pridružuju nizu [11 11 01 11] s desne strane, dakle, izlaz je [11 11 01 11 01].
- Iz stanja [011] (*f*) ide se u stanja [001] (*g*), a na izlazu se pojavljuju bitovi 11 i pridružuju se gornjemu nizu s desna, a
- zatim 0 na ulazu vodi prema stanju 000 (*a*) i konačnome izlaznom nizu: [11 11 01 11 01 01 11].

Ovo je isti rezultat kao i onaj što smo ga dobili zbrajanjem pojedinačnih *impulsnih odziva* za bitove [1011000].

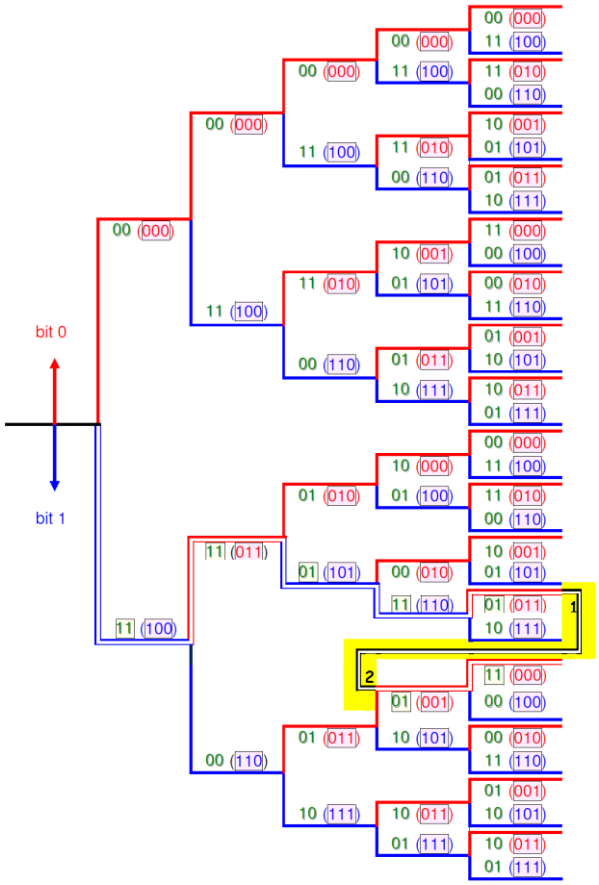
[105] Objasniti ovu rečenicu!?!

### 6.4.3. 6.4.3. DIJAGRAM STABLA

Slika 6.17 prikazuje dijagram stabla za (2, 1, 4) kod.

Crvene okomite crte znače dolazak bita 0.
Plave okomite crte označavaju dolazak bita 1.
Na vodoravnim crtama, prva 2 bita su izlazni.
Broj unutar zagrada je stanje registra
Prva grana (lijevo na slici), označava dolazak 0 ili 1
Za početno stanje pretpostavlja se da je [000].
Ako se primi 0, penjemo se.
Ako se primi 1, spuštamo se.
Neka je kodni niz [1011] kao i prije.
Na grani 1 (lijevo), spuštamo se (dvostruka crta).
Izlaz je 11 (brojke zelene boje), a stanje je [100].
Sada se primila 0 pa se penjemo (dvostruka crta).
Izlazni bitovi su 11, a stanje je sada [011].
Sljedeći dolazni bit je 1.
Spuštamo se, pa smo dobili izlaz 01
Sada je stanje izlaza [101].
Sljedeći dolazni bit je 1.
Spuštamo se i opet su izlazni bitovi 11.
Od ove točke, u odzivu na ulazni bit 0
izlaz je 01, a stanje kodera je [011].
Što ako je niz duži,
što ako ponestane grana na drvetu?
Registri kodera "ispiru" se
tj., njegova stanja popunjavaju se nulama
tako je naš ulazni niz zapravo [1011000].
Zadnja 3 bita [000] zovu se
"bitovi za ispiranje" (*flush bits*) ili
"punjenje repa bitovima" (*tail-biting*).
Sada iz točke 1 skačemo u točku 2 (žuta traka)
i penjemo se za tri grane
pa na izlazu imamo potpun niz, a to je:

[11110111101011].




*Slika 6.17: Dijagram stabla (2, 1, 4) koda*

Dijagram stabla prikazuje prolaz vremena, kako idemo dublje u grane drveta. On je nešto bolji od dijagrama stanja, ali još uvijek ne nameće se kao pristup za predstavljanje konvolucijskih kodova.

Ovdje umjesto "skakanja" iz jednoga stanja u drugo, "pomičemo" se po granama stabla gore-dolje, ovisno o tomu jesmo li primili 0 ili 1.

Na slici 6.17, pune crte pokazuju dolazak bita 0, a crtkane crte dolazak bita 1. Prva dva bita pokazuju izlazne bitove, a broj unutar okruglih zagrada, opisuje stanja kodera.

Prva grana lijevo na slici 6.17 označava dolazak bita 0 ili 1. Za početno stanje pretpostavlja se da je [000]. Ako se primi 0, penjemo se, a ako se primi 1, onda se spuštamo niz stablo.

Neka je kodni niz jednak [1011] kao i prije. Na prvoj grani, spuštamo se. Izlaz je 11, a stanje registara je [100]. Zatim se primila 0 pa se penjemo. Izlazni bitovi su 11, a stanje je sada [010].

Sljedeći dolazni bit je 1. Spuštamo se i dobivamo izlaz 01 pa je sada stanje izlaz [101].

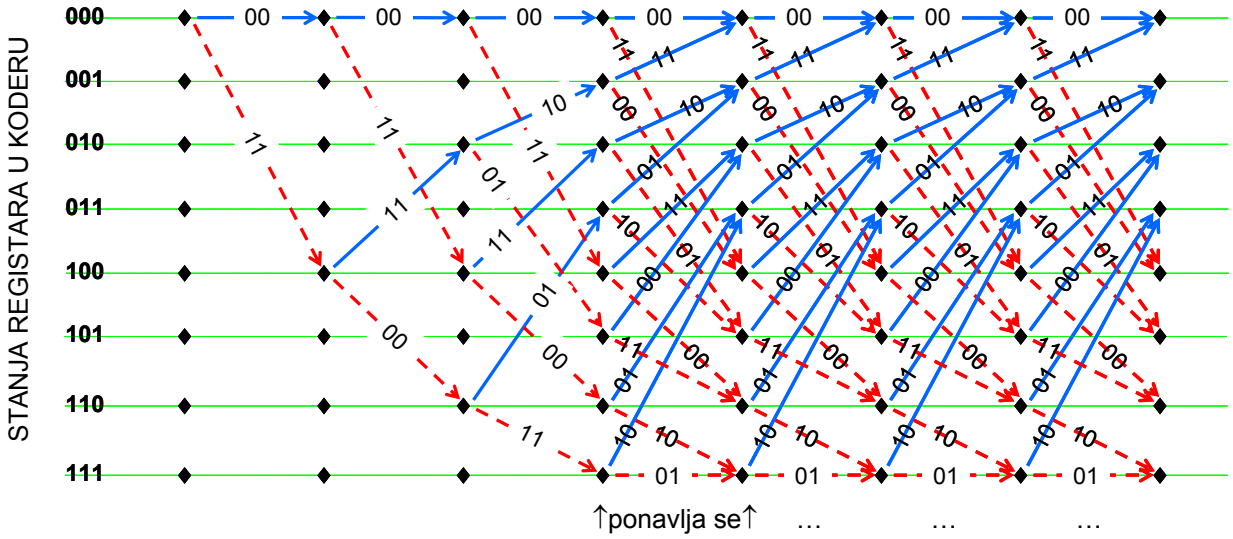
Sljedeći dolazni bit je 1 pa se spuštamo i opet su izlazni bitovi 11. Od ove točke, u odzivu na ulazni bit 0 izlaz je 01, a stanje registara je [010].

Što ako je niz duži, što ako ponestane grana na drvetu? U tome slučaju, dijagram stabla ponavlja se. U stvari moramo "isprati" koder, tj. njegova stanja popuniti nulama, tako je naš niz zapravo [1011000], a zadnja 3 bita [000] zovu se "bitovi za ispiranje" (*flush bits*) ili "punjenje repa bitovima" (*tail-biting*).

Sada skačemo u točku 2 na stablu i penjemo se za tri grane. Na izlazu imamo potpun niz, a to je [11110111101011]. Možda niste iznenađeni da je to također isti rezultat kao i onaj koji smo dobili dijagramom stanja.

### 6.4.4. 6.4.4. DIJAGRAM REŠETKE

Dijagrami rešetke (slika 6.18) su neuredni, ali općenito imaju prednost pred dijagramima stabla i stanja, jer predstavljaju linearan vremenski redoslijed događaja.



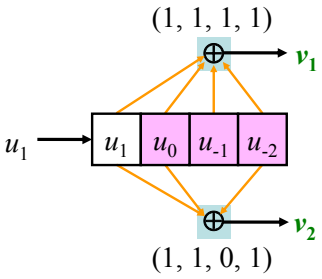

*Slika 6.18: Dijagram rešetke za (2, 1, 4) kod*[106]

Os *x* prikazuje diskretno vrijeme, a sva moguća stanja prikazana su na osi *y*. Krećemo se vodoravno kroz rešetku u vremenu. Svaki prijelaz znači dolazak novih bitova.

Dijagram rešetke ostvaruje se crtanjem svih mogućih stanja ($2^L$) na okomitoj osi. Onda spajamo svako stanje sljedećim stanjem dopuštene kodne riječi za to stanje. Postoje samo dva moguća izbora u svakome stanju. Ono je određeno dolaskom bita 0 ili bita 1. Strelice pokazuju ulazni bit, a izlazni bitovi su prikazani u zagradama. Strelice usmjerene prema gore, predstavljaju bit 0, a one usmjerene prema dolje, predstavljaju bit 1. Dijagram rešetke jedinstven je za svaki kod, isto kao što su: dijagram stanja i dijagram stabla. Možemo nacrtati rešetku za onoliki broj razdoblja koliki želimo. Svako razdoblje ponavlja moguće prijelaze.

Uvijek počinjemo stanjem 000. Počevši od ovoga stanja, rešetka se širi na *L* bitova i postaje potpuno popunjena, kao da su svi prijelazi mogući. Prijelaze ponovite iz ove točke.

Podsjetimo se, kako ustrojiti rešetku za (2, 1, 4) koder. Konvolucijskim kodovima obično su pridružena tri parametra (*n*, *k*, *m*). Oznake u zagradama znače sljedeće: *n* … broj izlaznih bitova = 2, *k* … broj ulaznih bitova = 1, a *m* … broj memorijskih registara = 4. Da bi svakoj strelici pridružili ulazna i izlazna stanja kodera (npr. 1/10 za stanje110), ona se računaju tako da se uz poznato stanje registara zbroje 2 polinoma koji tvore "dobre" kodove, a prikazana su u tablici 6.1: "Polinom-generatori što ih je pronašao Busgang za kodove brzine 1/2". Ponavljamo i sliku za (2, 1, 4) koder s pridruženim polinomima uz svako zbrajalo (slika 6.19).




*Slika 6.19: (2, 1, 4) koder s pridruženim polinomima uz svako zbrajalo*

Izlaz tvori sastav od 2 bita redoslijedom $v_1v_2$. Tako je brzina prijenosa 1/2, jer za prijenos 2 bita treba dvostruko više vremena u odnosu na prijenos 1 bita. Svojstva takvoga kodera u zaštitnome kodiranju opravdavaju "usporenje" prijenosa. Polinom-generatori su: $g_1 = x^3 + x^2 + x^1 + x^0$ i $g_2 = x^3 + x^2 + 0 + x^0$.

[106] ZADATAK: Nacrtati dijagram rešetke za zadan primljen niz kodiranih binarnih simbola!

U ovome primjeru, ulazni bit je 1, što prema dogovoru u dijagramu rešetke, usmjerava strelicu prema dolje (dolazni bit 0 usmjerava strelicu prema gore u čitavome dijagramu). Ovisno o pristiglome bitu na ulazu, registar se prebacuje iz stanja u stanje (ili ostaje u istome stanju). Ako je registar u stanju [110], dolaskom 1 na ulaz u koder, stanje registra mijenja se u stanje [111], a to je dodatan razlog usmjerenja strelice prema dolje. Ova 2 razloga povezana su međusobno uzročno-posljedično. Dođe li 1, kao sljedeća znamenka na ulaz u koder, očito da koder ostaje u ovome stanju (to potvrđuje i smjer strelice), a ako dođe 0, stanje registara se mijenja u [011] pa strelica iz ovoga stanja (u trenutku $t_3$ iz primjera) doseže to stanje.

## 6.5. 6.5. Dekodiranje

Postoji nekoliko različitih pristupa dekodiranju konvolucijskih kodova. Oni su združeni u dvije osnovne kategorije.

1. Serijsko dekodiranje
   - Fano algoritam
2. Dekodiranje najvećom vjerojatnosti
   - Viterbijevo dekodiranje

Obje ove metode predstavljaju dva različita pristupa istoj osnovnoj ideji dekodiranja.

### 6.5.1. 6.5.1. OSNOVNA IDEJA DEKODIRANJA

Pretpostavimo da su poslana 3 bita brzinom koda od 1/2. Primamo 6 bitova ali za sada zanemarimo bitove za ispiranje. Ovih šest bitova mogu, ali i ne moraju sadržavati pogreške. Znamo iz postupka kodiranja da se ti bitovi jedinstveno preslikavaju. Dakle, niz od 3 bita imat će jedinstven izlaz od 6 bitova. No, zbog pogrešaka, možemo dobiti bilo koju varijaciju od 2 elementa ($n = 2$) 6. razreda ($r = 6 = 6$ bitova), s ponavljanjem.

Permutacije triju ulaznih bitova rezultiraju u osam mogućih ulaznih nizova. Svaki od njih pomoću koda, jedinstveno se preslikava u nizove od šest izlaznih bitova. Ovo čini skup dopuštenih nizova pa je zadatak dekodera odrediti koji niz se poslao.

Pretpostavimo prijem niza [111100]. To nije ni jedan od 8 mogućih nizova. Kako ga dekodirati? Možemo učiniti dvije stvari:

1. Možemo usporediti primljen niz sa svim dopustivim nizovima i odabrati onaj s najmanjom Hammingovom udaljenošću (ili najmanjim neslaganjem među bitovima)
2. Možemo napraviti korelaciju i odabrati nizove s najboljom korelacijom.

Prvi postupak je u osnovi ono što se krije pod nazivom *dekodiranje tvrdom odlukom* (*hard decision decoding*), a drugi postupak je *dekodiranje mekom odlukom* (*soft decision decoding*). Podudarnost bitova, kao i umnožak između primljenoga niza kodne riječi, pokazuje da smo dobili dvosmislen odgovor pa još uvijek ne znamo što se poslalo.

Kako broj bitova raste, povećava se broj izračuna potrebnih za dekodiranje na grub način (*brute force*), pa dekodiranje ovim načinom više nije praktično. Moramo pronaći učinkovitiji način da se ne ispituju sve inačice, ali da možemo riješiti nejasnoće gdje imamo dva moguća odgovora. U tablici 6.4 ta dva odgovora, prikazana su podebljano i osjenčano.

*Tablica 6.4 Sporazum o bitovima koristi se kao mjera odluke između primljenoga niz i osam mogućih ispravnih vrijednosti kodiranoga niza.*

| Ulaz | Ispravan kodiran niz | Primljen niz | Broj podudarnih bitova |
|---|---|---|---|
| 000 | 000000 | 111100 | 2 |
| 001 | 000011 | 111100 | 0 |
| 010 | 001111 | 111100 | 2 |
| 011 | 001100 | 111100 | 4 |
| 100 | **111110** | 111100 | **5** |
| 101 | **111101** | 111100 | **5** |

| Ulaz | Ispravan kodiran niz | Primljen niz | Broj podudarnih bitova |
|---|---|---|---|
| 110 | 110001 | 111100 | 3 |
| 111 | 110010 | 111100 | 3 |

Ako se dobila poruka dužine $s$ bitova, onda je moguć broj kodnih riječi $2^s$. Kako možemo dekodirati niz bez provjere svake od tih $2^s$ kodnih riječi? To je osnova ideje postupka dekodiranja.

### 6.5.2. 6.5.2. SERIJSKO DEKODIRANJE

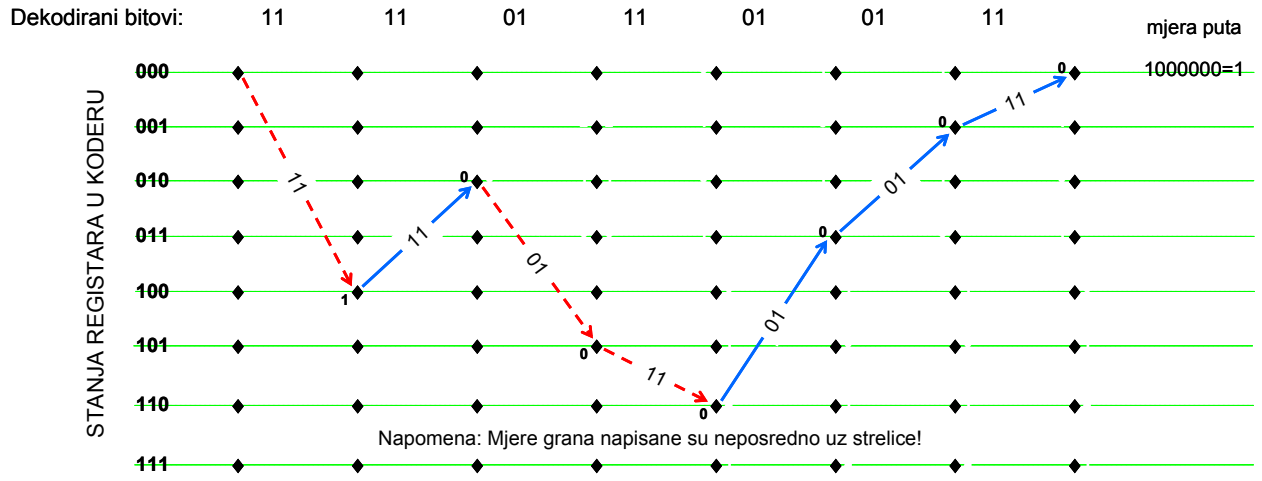
Serijsko dekodiranje, bila je jedna od prvih metoda predloženih za dekodiranje konvolucijski kodiranoga toka bitova. Nju je prvi predložio Wozencraft, a poslije je Fano predložio bolju inačicu. Serijsko dekodiranje najbolje se opisuje prispodobom. Daju nam se neke upute koje se sastoje od poznatih smjernica. No, osoba koja je dala upute nije učinila dobar posao pa se ponekad ne prepoznaje smjerokaz, ali zbog toga što se ne vidi niti jedna oznaka, osjeća te se da ste na krivome putu. Vraćate se (*backtrack*)[107] do točke gdje možete prepoznati smjerokaz, a onda odabirete zamjenski put sve dok ne naiđete na sljedeći smjerokaz i konačno dođete do odredišta. U ovome postupku, možete se vratiti nekoliko puta, ovisno o tome koliko su dobre smjernice o putovima.

Slično, u serijskome dekodiranje bavite se samo jednim putom istovremeno. Možete se odreći puta u bilo koje vrijeme i vratiti se natrag te slijediti drugi put, ali važna stvar je da u bilo kojemu trenutku slijedite samo jedan put.

Serijsko dekodiranje omogućuje kretanja kroz rešetku naprijed i natrag. Dekoder prati svoje odluke i svaki puta donosi dvosmislenu odluku. Ako podudarnost[108] raste brže od vrijednosti nekoga praga, dekoder odustaje od toga puta i vraća se natrag na zadnji put gdje je podudarnost bila ispod toga praga.

#### *6.5.2.1. 6.5.2.1. Dekodiranje dijagramom rešetke*

Konačan izgled dijagrama rešetke (*trellis diagram*) prikazuje slika 6.20.




*Slika 6.20: Kodiran niz, ulazni bitovi [1011000], izlazni bitovi* [11 11 01 11 01 01 11]

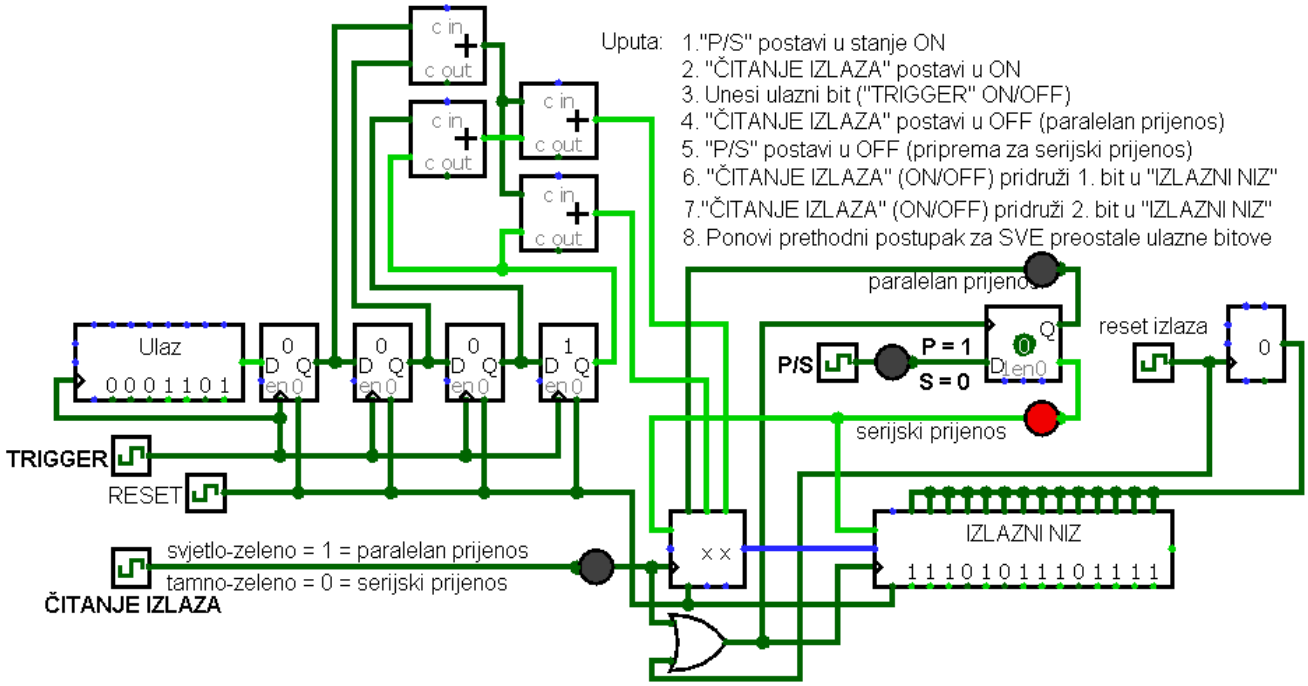
Na slici 6.20, dolazni bitovi prikazani su na vrhu. Možemo započeti samo u točki 1. Kodiranje je jednostavno. Idemo gore ako je primljen bit 0, a dolje ako je primljen bit 1. Trag bitova u našemu primjeru je niz [1011000] i pokazuju ga crte. Vidimo da dijagram rešetke daje isti izlazan niz kao i ostale četiri metode: *pregledna tablica*, *impulsni odziv*, *dijagram stanja* i *dijagram stabla.* Svi dijagrami izgledaju slično i jedinstveni su za svaki kod.

---

[107] *Backtracking* … Pretraživanje unatrag je opći algoritam za pronalaženje svih (ili nekih) rješenja za neke računalne probleme, na način da se postupno grade kandidati za rješenja, a napušta se pojedini djelomičan kandidat $c$ (*backtracks*), čim se utvrdi da $c$ ne može dovesti do pravoga rješenja.

[108] *tallies* … podudara

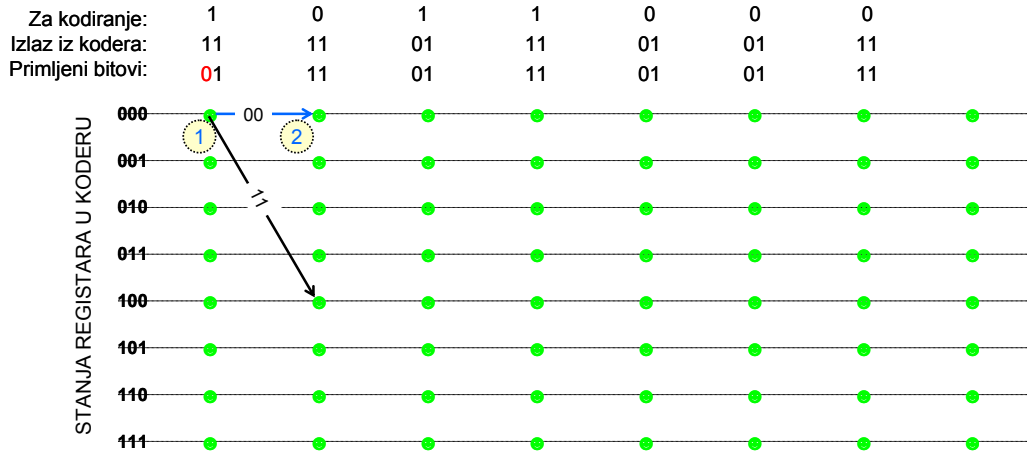
Pogledajmo primjer (slika 6.21).




*Slika 6.21: Koder u simulacijskome programu LOGISIM-win-2.7.1 za (2, 1, 4) kod*

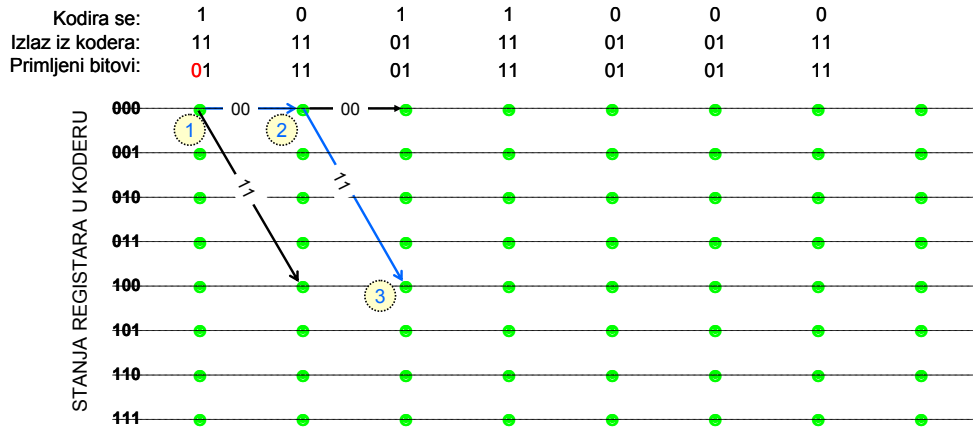
Za kod (2, 1, 4), u prethodnome poglavlju nacrtali smo dijagrame kodera. Pretpostavimo da se kodirao niz bitova [1011000]. Ne zaboravite da su posljednja tri bita potrebna za "ispiranje" registra. Njihov naziv je *bitovi na repu* (*tail bits*). Ako se nije pojavila pogreška, dobit ćemo: [11 11 01 11 01 01 11].

Napomena: Prikaz niza brojeva [1011000] na slici 6.21, unosi se obrnutim redoslijedom u koder i obrnutim redoslijedom prikazuje ga se na izlazu kao niz [11 10 10 11 10 11 11]. Ali recimo da smo umjesto toga niza dobili: [01 11 01 11 01 01 11]. Pojavila se jedna pogreška. Prvi primljen bit je 0 umjesto 1 (slika 6.22.a).



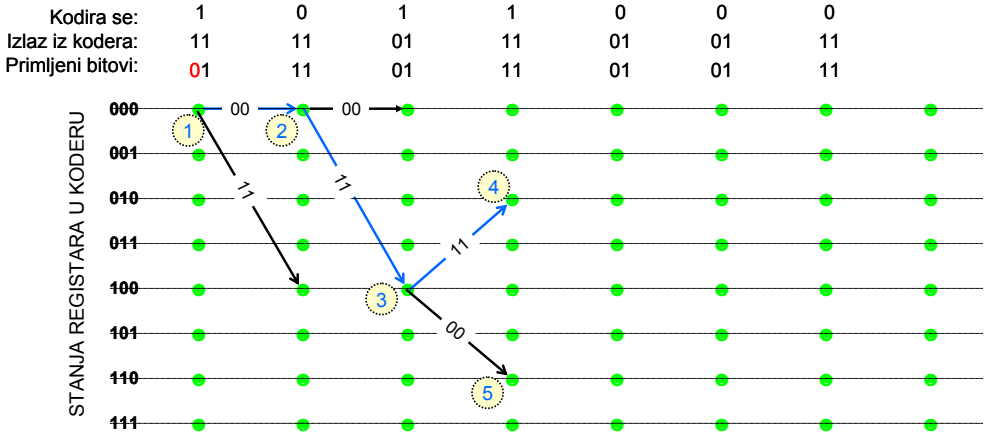

*Slika 6.22.a: Serijsko dekodiranje pretragom puta za (2, 1, 4) kod*

1. **Točka odluke 1** Dekoder promatra prva dva bita, 01. Odmah se uočava da je došlo do pogreške zbog toga što prva dva bita mogu imati vrijednosti samo 00 (ako je prvi bit na ulazu u koder bio 0) ili 11 (ako je prvi bit na ulazu u koder bio 1). No, koji je od dvaju bitova primljen kao pogrešan, prvi ili drugi? Dekoder slučajno odabire 00 kao polazni izbor. Da bi bitovi odgovarali sastavu 00, dekodira se ulazni bit kao 0. U svoj brojač pogrešaka dekoder postavlja 1. Sada se dekoder nalazi u točki odluke 2 (slika 6.22.b).



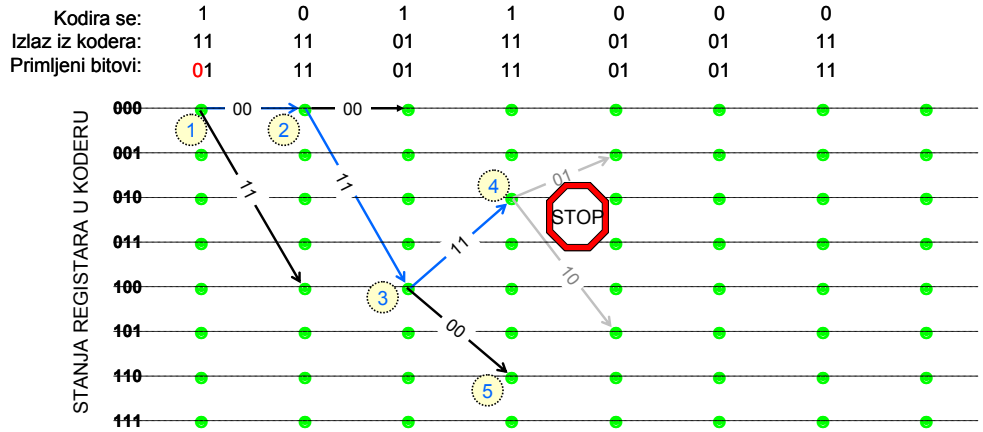

*Slika 6.22.b: Serijsko dekodiranje pretragom puta za* (2, 1, 4) *kod*

2. **Točka odluke 2** Dekoder promatra sljedeći skup od 2 primljena bita, a oni su 11. Odavde, donosi odluku da je poslana 1 što odgovara upravo jednoj od kodnih riječi. Ova odluka dovodi dekoder do točke odluke 3 (slika 6.22.c).




*Slika 6.22.c: Serijsko dekodiranje pretragom puta za* (2, 1, 4) *kod*

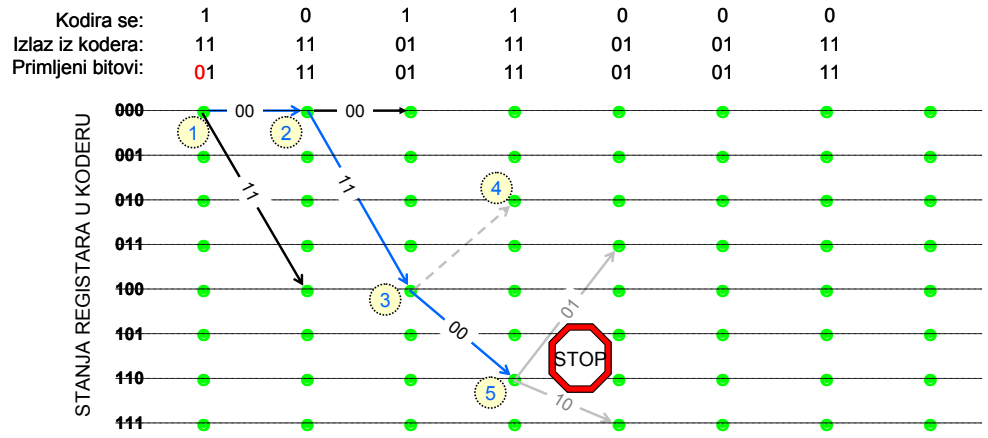
3. U **točki odluke 3.** Primljeni bitovi su 01, ali pod-izbori za kodne riječi su 00 (ako je sljedeći ulazni bit u koder prije kodiranja bio 1) i 11 (ako je sljedeći ulazni bit u koder prije kodiranja bio 0). Uočava se kao nova pogreška pa se broj pogrešaka povećan na 2. Dok god je broj pogrešaka manji od vrijednost praga 3 (a to smo postavili na temelju statistike kanala), dekoder nastavlja naprijed. On proizvoljno odabire put prema gore (11) (prema točki 4) i nastavlja do donošenja odluke u točki broj 4 ako se kodirala 0 (slika 6.22.d).




*Slika 6.22.d: Traženje puta serijskim dekodiranjem za* (2, 1, 4) *kod*

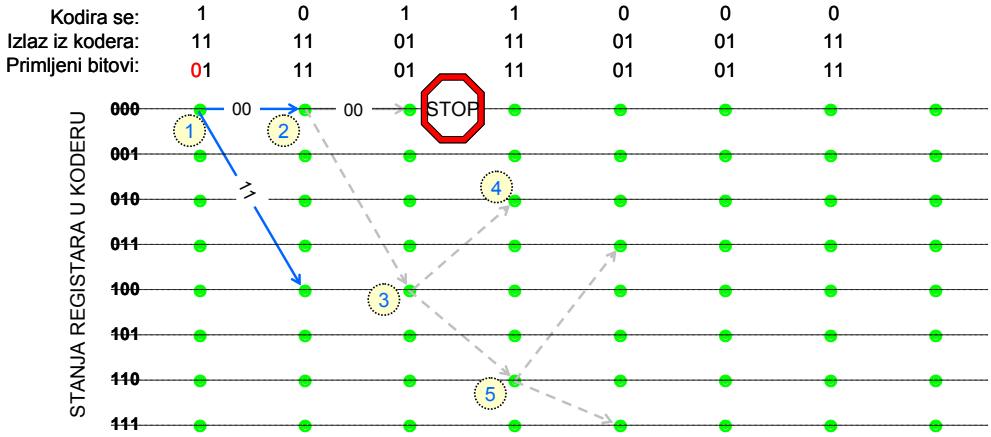
4. U **točki odluke 4**. Dekoder prepoznaje još jednu pogrešku, jer su primljeni bitovi 11, ali su mogući izbori za kod(ira)ne znamenke 10 i 01, jer samo ove znamenke mogle biti rezultat kodiranja. Ulaznom "1" mogle su dovesti koder ili u stanje 101 ili ulaznom "0" mogle su dovesti

koder u stanje 001. Broj pogreška povećava ukupan zbroj i izjednačava ga s 3 pa to upućuje dekoder da se vrati natrag za jedan korak (slika 6.22.e).



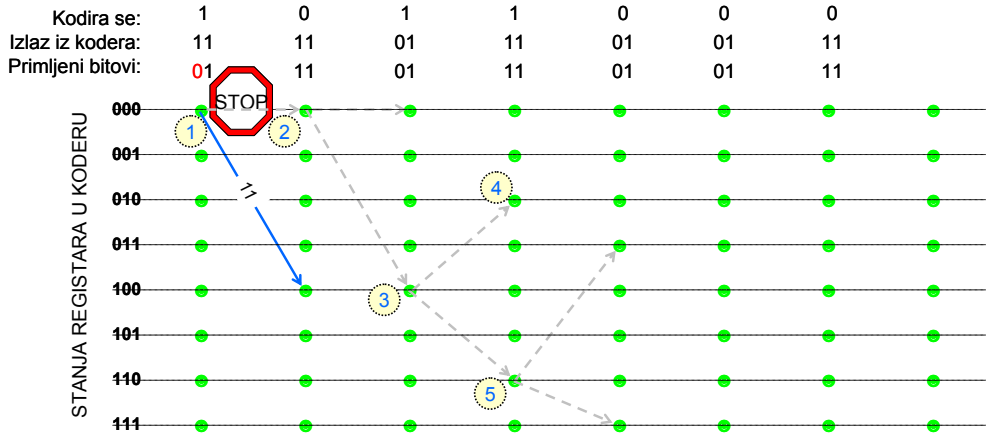

*Slika 6.22.e: Traženje puta serijskim dekodiranjem za (2, 1, 4) kod*

5. Dekoder se vraća do točke 3, gdje je zbroj pogrešaka manji od 3 pa izabire drugi put iz točke 3 prema točki 5. Dekoder opet nailazi na pogrešku. Primljeni bitovi su 11, ali mogući izlazi iz kodera su 01 i 10. Brojač pogrešaka opet raste na 3 pa se dekoder još jedanput vraća natrag (slika 6.22.f).




*Slika 6.22.f: Traženje puta serijskim dekodiranjem za (2, 1, 4) kod*

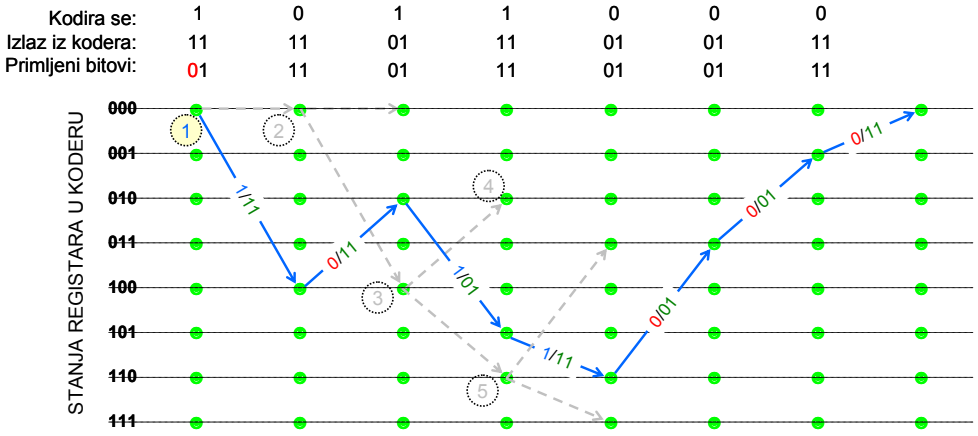
6. Oba moguća puta iz točke 3 su iscrpljena. Dekoder mora ići natrag do točke 2 (prije 3. točke). On se i vraća do točke 2, ali ovdje, ako prati točku 2, brojač pogrešaka odmah se povećava na ukupan zbroj 3. Dakle, dekoder se mora vratiti iz točke 2 natrag u točku 1 (slika 6.22.g).




*Slika 6.22.g: Traženje puta serijskim dekodiranjem za (2, 1, 4) kod*

7. U točki 1, prvi primljeni par binarnih znamenaka (01) nije ispravan pa se prva znamenka (0) zamjenjuje (1) i tako se ispravlja neispravno primljena znamenka. Od točke 1, izborom 1 kao prve znamenke, svi izbori na koje se naiđe, savršeno se podudaraju s izborom primljenih

združenih kodiranih znamenaka pa dekoder uspješno dekodira poruku kao [1011000] (slika 6.22.h).




*Slika 6.22.h: Traženje puta serijskim dekodiranjem za* (2, 1, 4) kod*,*

Za serijsko dekodiranje nizova upotrebljava se memorija pa se tako ova metoda koristi zajedno s kodovima duge duljine ograničenja, gdje je *S*/*N* također malen. Neke veze planetarnih misija agencije NASA, koristile su prednost serijskoga dekodiranja.

### 6.5.3. 6.5.3. MAKSIMALNA VJEROJATNOST I VITERBIJEVO DEKODIRANJE

Viterbijevo dekodiranje najpoznatija je provedba dekodiranja najvećom vjerojatnosti (najmanjom mjerom grane/puta). Ovdje smo sustavno suzili mogućnost izbora u svakome vremenskom razmaku. Glavna korist smanjenja izbora je:

1. Pogreške se javljaju rijetko. Vjerojatnost pogreške je mala.
2. Vjerojatnost dviju pogrešaka u nizu je mnogo manja od vjerojatnosti pojave jedne pogreške tj. one su slučajno raspodijeljene.

Viterbijev dekoder ispituje čitav primljen niz zadane duljine. Dekoder računa mjeru za svaki put i donosi odluku na temelju toga pokazatelja. Istražuju se svi putovi sve dok se dvije staze ne spoje u jednome čvoru. Zatim se odabire put s nižom mjerom grane, a onaj s višom mjerom grane odbacuje se. Staze koje se odaberu, zovu se *preživjele* (*survivors*).

Za niz od $N$ bitova, ukupan broj mogućih primljenih nizova je $2^N$. Od njih, samo $2^{kL}$ su valjani. Viterbijev algoritam primjenjuje *načela najveće vjerojatnosti* (*maximum-likelihood principles*) za ograničiti usporedbu na samo $2^{kL}$ preživjelih staza umjesto da se provjeravaju sve staze.

Najčešća korištena mjera udaljenosti binarnih simbola je Hammingova mjera (za dekodiranje tvrdom odlukom). Ona je samo običan umnožak[109] (određuje se i kao broj za koliko se bitova na istim položajima, međusobno razlikuju dva promatrana niza bitova) između *primljene* i *dopuštene* kodne kombinacije. Ostale mjere također se koriste i njih će se opisati u nastavku (vidi tablicu 6.7).

*Tablica 6.7 Svaka grana ima Hammingovu mjeru ovisno o primljenome nizu i o ispravnoj kodnoj riječi u tomu stanju*

| primljeni bitovi | ispravna kodna riječ 1 | Hammingova mjera 1 | ispravna kodna riječ 2 | Hammingova mjera 2 |
|---|---|---|---|---|
| 00 | 00 | 0 | 11 | 2 |
| 01 | 10 | 2 | 01 | 0 |
| 10 | 00 | 1 | 11 | 1 |

Ove mjere su *kumulacijske*, tako da je *put s najmanjom ukupnom mjerom konačan pobjednik*. No, sve to ne bi imalo smisla dok se ne vidi algoritam u radu (Laboratorijska vježba 15).

[109] VIP!

## 16. *Laboratorijska vježba 15: Dekodiranje algoritmom dekodiranja nizova*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

## 6.6. 6.6. Optimalno dekodiranje konvolucijskih kodova

### 6.6.1. 6.6.1. VITERBIJEV ALGORITAM

U našoj raspravi o raznim shemama dekodiranja za blok-kodove, vidjeli smo da postoji mogućnost dekodiranja mekom i tvrdom odlukom. Pri dekodiranju mekom odlukom, **r**, vektor, koji označava izlaze iz podudarnih filtara, uspoređuje se s različitim signalizacijskim točkama u konstelaciji sustava kodirane modulacije. Izabire se ona s najbližom Euklidovom udaljenosti. Pri dekodiranju tvrdom odlukom, **r** se najprije pretvara u binaran niz **y** donošenjem odluke o pojedinim komponentama **r**, a zatim se odabire kodna riječ, koja je najbliža **y** u smislu Hammingove udaljenost.

Vidi se da je u oba pristupa, temeljni zadatak pronaći *put kroz rešetku koji je na najmanjoj udaljenosti od određenoga niza*. Ovaj temeljni problem javlja se u mnogim područjima komunikacije i drugim područjima elektrotehnike. Osobito, na isti problem naišlo se u procjeni ML niza pri prijenosu preko kanala ograničene pojasne širine s među simbolskim smetnjama, shemama CPM[110] demodulacija, prepoznavanju govora, nekim shemama razvrstavanja uzoraka, itd. Svi ovi problemi u suštini su isti, a mogu se nazvati *algoritmi optimalne pretrage rešetke* (*optimal trellis searching algorithms*). Poznat Viterbijev algoritam, pruža zadovoljavajuće rješenje svih tih problema.

U dekodiranju tvrdom odlukom konvolucijskih kodova, želimo odabrati put kroz rešetku čija kodna riječ, označena kao **c**, nalazi se na minimalnoj Hammingovoj udaljenosti od kvantiziranoga primljenoga niza **y**. Pri dekodiranju tvrdom odlukom, kanal je binaran bez pamćenja. Za šum u kanalu, pretpostavlja se da je bijeli Gaussov šum. Zbog toga što željeni put započinje od stanja "sve 0" i vraća se u stanje "sve 0", možemo pretpostaviti da ovaj put obuhvaća ukupno $m$ grana, a budući da svaka grana odgovara količini od $n$ bitova izlaznoga kodera, ukupan broj bitova u **c**, a također i u **y** je $m \cdot n$. Označimo s $\mathbf{c}_i$ odnosno $\mathbf{y}_i$, niz bitova koji odgovaraju $i$-toj grani, gdje je $1 \le i \le m$, a svaki $\mathbf{c}_i$ odnosno $\mathbf{y}_i$ duljine je $n$. Hammingova udaljenost između **c** i **y** je, dakle:

$$d(\mathbf{c},\mathbf{y}) = \sum_{i=1}^{m} d(\mathbf{c}_i,\mathbf{y}_i) \tag{6.6}$$

Pri dekodiranju mekom odlukom, imamo slično stanje uz tri razlike.

1. Umjesto **y**, bavimo se izravno izlaznim vektorom **r**, iz optimalnoga digitalnoga demodulatora (što se podudara ili filtarskom ili korelacijskom vrstom).
2. Umjesto niza binarnih nula i jedinica **c**, rukujemo odgovarajućim nizom **c'**

$$c'_{ij} = \begin{cases} \sqrt{\varepsilon} & c_{ij} = 1 \\ -\sqrt{\varepsilon} & c_{ij} = 0 \end{cases} \qquad za\ 1 \le i \le m \quad i \quad 1 \le j \le n$$

3. Umjesto Hammingove udaljenosti, koristimo Euklidovu udaljenost. To je posljedica činjenice da se proučava kanal s dodanim bijelim Gaussovim šumom.

Iz gornjega, imamo

$$d_E^2(\mathbf{c}',\mathbf{r}) = \sum_{i=1}^{m} d_E^2(\mathbf{c}'_i,\mathbf{r}_i)\,. \tag{6.7}$$

Iz jednadžbi (6.6) i (6.7), vidi se generički oblik problema kojega moramo riješiti. Za zadan vektor **a** treba pronaći put kroz rešetku, početnim stanjem, $S_i = 0$ ("sve 0") i završnim stanjem, $S_m = 0$ ("sve 0"), tako da se minimizira neka udaljenost koja se mjeri između nizova **a** i **b** koja odgovara tomu putu.[111] Važna činjenica koja čini ovaj problem lako rješivim jest da udaljenost koja nas zanima između **a** i **b** u oba slučaja može se napisati kao zbroj udaljenosti koja odgovara putu pojedinih grana. To se lako uoči iz jednadžbi (6.6) i (6.7).

---

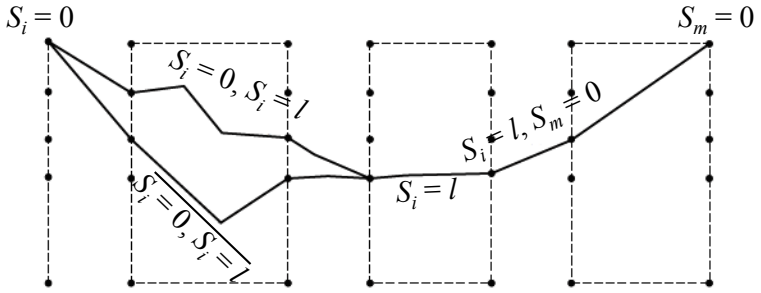
[110] *Continuous-Phase Modulation* … modulacija kontinuiranom fazom

[111] Problem se može formulirati kao problem maksimalizacije. Na primjer, umjesto da se smanjuje Euklidova udaljenost, mogla bi se povećati korelacija.

Pretpostavimo da se problem općenito formulira kao minimiziranje mjere μ u obliku

$$\mu(\mathbf{a},\mathbf{b}) = \sum_{i=1}^{m} \mu(\mathbf{a}_i, \mathbf{b}_i)$$

Put koji se sastoji od $\overline{(S_i = 0, S_i = l)}$ i $(S_i = l, S_m = 0)$ optimalan je put, a prikazuje ga slika 6.23.

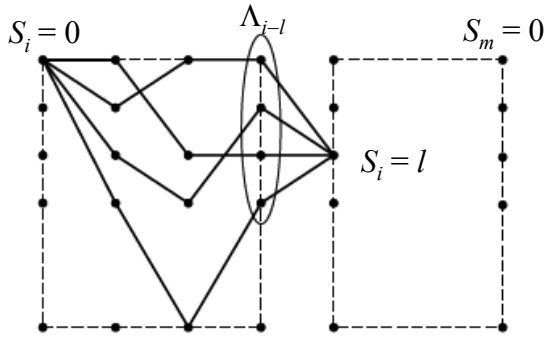



*Slika 6.23: Usporedba optimalnoga putu, $(S_i = 0, S_i = l, S_m = 0)$, s pod optimalnim putom koji se sastoji od ulančenja $\overline{(S_i = 0, S_i = l)}$ i $(S_i = l, S_m = 0)$.*

Najprije uočimo da, ako je put $(S_i = 0, S_i = l, S_m = 0)$, $1 \le i \le m$, optimalan put, gdje je $0 \le l \le M-1$ ($M = 2^{(L-1)k}$ je broj stanja), onda je doprinos mjere na putu $(S_i = 0, S_i = l)$ manji od doprinosa mjere bilo kojega drugoga puta $\overline{(S_i = 0, S_i = l)}$ povezujući $S_i = 0$ sa $S_i = l$.

Broj grana koje ulaze u svako stanje u rešetci jednak je $2^k$. Neka $\Lambda_{i-1}$ označuju skup $2^k$ stanja koja su povezana granom prema $S_i = l$. Ako je put od $S_i = 0$ do $S_i = l$ optimalan put između tih dvaju stanja, onda je ova putanja veza optimalnoga puta koja povezuje stanje $S_i = 0$ sa stanjem $S_{i-1} = \lambda$ za neke $\lambda \in \Lambda_{i-1}$ i grane koja povezuje $S_{i-1} = \lambda$ sa $S_i = l$. Optimalan put povezivanja od $S_i = 0$ do $S_{i-1} = \lambda$ zove se *preživjela staza* (*survivor path*) u stanju $S_{i-1} = \lambda$, ili jednostavno *preživjela* (*survivor*).

Stoga, kako bi se pronašla preživjela staza (*survivor path*) u stanju $S_i = l$, dovoljno je imati preživjele staze (i njihove podatke) za sve $S_{i-1} = \lambda$, $\lambda \in \Lambda_{i-1}$, pridružiti ih granama koje povezuju elemente $\Lambda_{i-1}$ prema $S_i = l$, pronaći mjeru rezultirajuće staze od $S_i = 0$ do $S_i = l$ te odabrati jednu s minimalnom mjerom. Ovo je nova preživjela staza u $S_i = l$. Ovaj postupak započeo je u $S_i = 0$ i završava u $S_m = 0$. Konačna preživjela staza u $S_m = 0$ je optimalna i najbolja staza (s najvećom vjerojatnošću) koja se maksimalno podudara s primljenim nizom. Ovaj postupak prikazuje slika 6.24.




*Slika 6.24: Traženje nove preživjele staze među starim preživjelim stazama.*

U svakome koraku, nova mjera preživjele je:

$$\mu(S_i = 0, S_i = l) = \min_{\lambda \in \Lambda_{i-1}} \{\mu(S_i = 0, S_{i-1} = \lambda) + \mu(S_{i-1} = \lambda, S_i = l)\} \qquad (6.8)$$

Nakon pronalaska mjere preživjele, nova preživjela staza u $S_i = l$ je staza $(S_i = 0, S_{i-1} = \lambda, S_i = l)$ za $\lambda$ koja minimizira mjeru preživjele u jednadžbi 6.8.

Prethodno naveden postupak može se sažeti Viterbijevim algoritmom.

1. Rastaviti primljen niz u $m$ pod-nizova od kojih je svaki duljine $n$.

2. Nacrtati rešetku dubine $m$ za kod koji se ispituje. Za posljednjih $L$–1 stanja rešetke, nacrtati samo one putove koje odgovaraju svim nultim ulaznim nizovima. To se radi, jer znamo da je ulaznome nizu dodano na kraju $k(L-1)$ nula zbog dovođenja kodera u početno stanje "sve 0".
3. Postavi se $i = 1$, a zatim se postavi mjera početno postavljenih svih nultih stanja, da bude jednaka nuli.
4. Nađe se udaljenost $i$-toga pod-niza primljenoga niza, prema svim granama povezujući $i$-tu fazu stanjâ s ($i$+1)-vom fazom stanjâ (*stage states*)[112].
5. Ove udaljenosti pribroje se mjerama sadržaja $i$-tih stanja da bi se dobile mjere kandidata za sadržaj ($i$+1)-ih faza stanjâ. Za svaku fazu ($i$+1)-voga stanja, postoji $2^k$ kandidata za mjeru, od kojih se svaki podudara s jednom granom koja završava u tome stanju.
6. Za svako stanje u ($i$+1)-oj fazi, odabrati kandidata s najmanjom mjerom grane, označiti granu koja odgovara ovoj minimalnoj vrijednosti kao preživjelu i pridružiti kandidata minimalne mjere kao mjeru ($i$+1)-ve faze stanja.
7. Ako je $i = m$, ići na korak 8, inače povećati $i$ za 1 i ići na korak 4.
8. Počevši stanjima "sve 0" u završnoj fazi, vratiti se natrag kroz rešetku zajedno s preživjelom stazom te dosegnuti početno stanje "sve 0". Ova staza, optimalna je staza pa se ulazni niz bitova podudara s ML dekodiranim informacijskim nizom.

Da bi se dobila najbolja podudarnost redoslijeda ulaznih bitova, treba ukloniti zadnjih $k(L-1)$ nula iz ovoga niza.

Kao što se vidi iz gornjega algoritma, kašnjenje dekodiranja i iznos memorije potrebne za dekodiranje dugih informacijskih nizova nisu prihvatljivi. Dekodiranje se ne može pokrenuti dok se ne primi cio niz (koji u slučaju konvolucijskih kodova, može biti jako dug), a sva preživjele staze moraju se pohraniti. U praksi je poželjno pod-optimalno rješenje koje ne uzrokuje takve probleme.

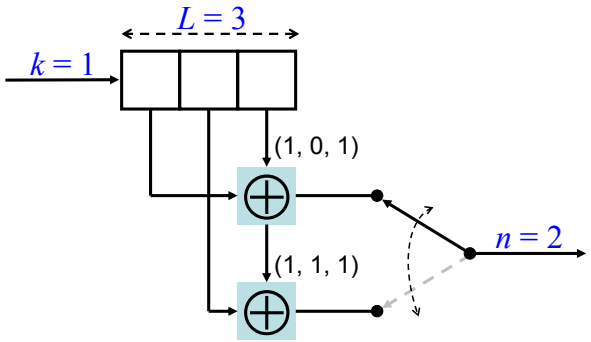
Jedan takav pristup naziva se *skraćivanje memorijskoga puta* (*path-memory truncation*). To je pristup u kojemu dekoder pri svakoj fazi u rešetci pretražuje samo δ prethodnih faza, umjesto da se pretražuje unatrag do početka rešetke. Uz ovome pristupu s (δ+1)-vom fazom, dekoder donosi odluku o ulaznim bitovima koji odgovaraju prvoj fazi rešetke (prvih $k$ bitova) pa sljedeći primljeni bitovi ne mijenjaju ovu odluku. To znači da je kašnjenje dekodiranja $k \cdot \delta$ bitova pa je samo to potrebno da bi preživjele staze odgovarale posljednjim δ fazama. Ako je $\delta \approx 5L$, računalne simulacije pokazuju da je narušavanje svojstava zbog skraćivanja memorijskoga puta zanemarivo.

### 6.6.2. 6.6.2. DEKODIRANJE TVRDOM ODLUKOM

Pretpostavimo da se kvantiziran primljen niz, dekodira tvrdom odlukom:

$$\mathbf{y} = [0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 1]$$

Konvolucijski koder jedan je od onih što ga prikazuje slika 6.25 i koristi se u ovoj skripti.




*Slika 6.25: Konvolucijski koder brzine 1/2*

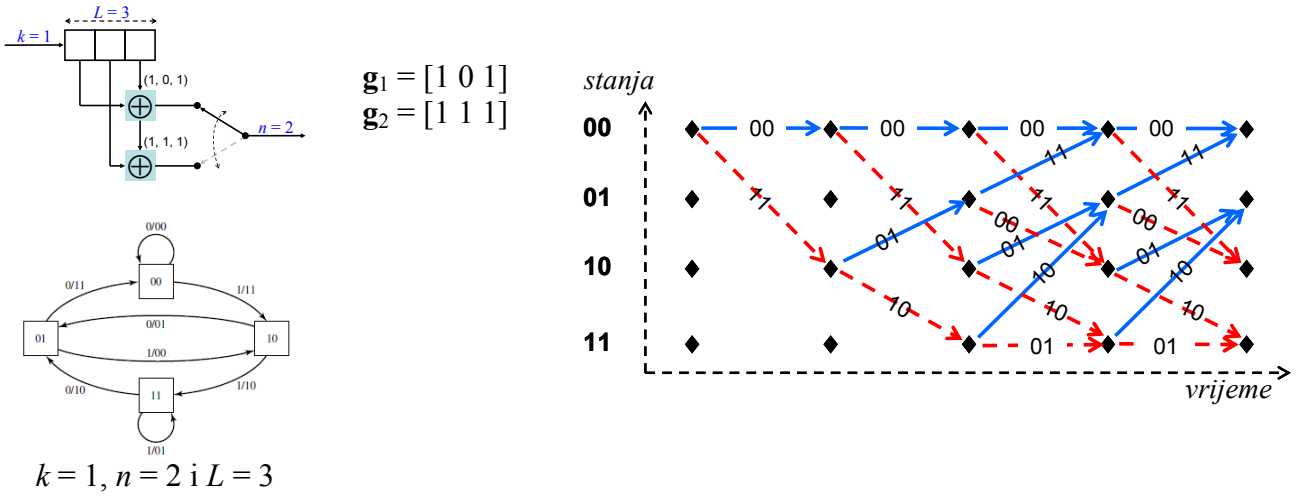
[112] *stage state* … faza stanja

U ovome koderu $k = 1$, $n = 2$ i $L = 3$. Stoga je brzina koda 1/2, a broj stanja je $2(L–1)k = 4$. Jedan od načina opisa takvoga koda (osim crtanja kodera) je navesti kako *dva izlazna bita* ($n = 2$) kodera, ovise o *sadržajima* (stanjima) posmičnih registara. To se radi na način da se odredi $n$ vektora $\mathbf{g}_1$, $\mathbf{g}_2$, ..., $\mathbf{g}_n$, koji su poznati kao *generator-nizovi* (*generator sequences*) konvolucijskoga koda. Proizvoljna $i$-ta ($1 \leq i \leq 2^{kL}$) komponenta $\mathbf{g}_j$, (gdje je $1 \leq j \leq n$) jednaka je 1, ako je to $i$-to stanje posmičnoga registra povezano zbrajalom. To odgovara $j$-tome bitu na izlazu, a u suprotnome jednaka je 0. U prije navedenome primjeru, *generator-nizovi* (*polinom-generatori*) zadani su kao:

$$\mathbf{g}_1 = [1\ 0\ 1]$$

$$\mathbf{g}_2 = [1\ 1\ 1]$$

Kao što je bio slučaj s dijagramom stanja prijelaza, ovdje opet imamo $2^k$ grana rešetki koje napuštaju svako stanje i $2^k$ grana spojenih u istome stanju. U slučaju ako je $k = 1$, uobičajeno je *punom* crtom označiti granu koja odgovara dolasku 0 u koder, a *isprekidanom* crtom označiti granu koja odgovara dolasku 1 u koder. Slika 6.26 pokazuje dijagram rešetke (*trellis diagram*) za kod koji opisuje koder na istoj slici.

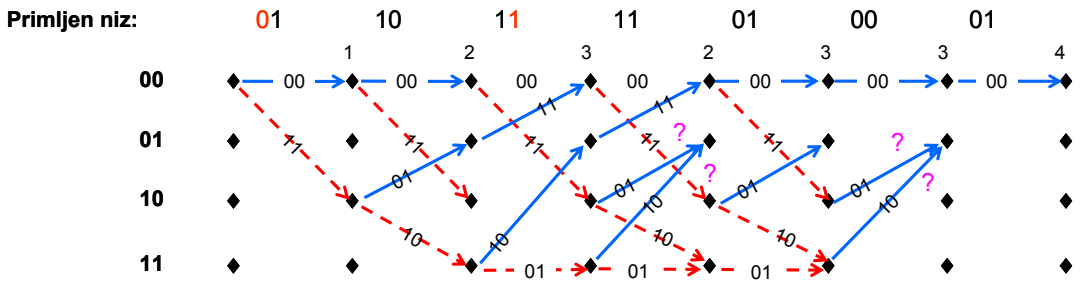



*Slika 6.26: Dijagram rešetke i dijagram stanja kodera na slici.*

*6.6.2.1. 6.6.2.1. Primjer*

Nađite ML (*maximum-likelihood*) informacijski niz i broj pogrešaka.

**Rješenje:** Radi se o (2, 1) kodu i $L = 3$. Duljina primljenoga niza **y** je 14. To znači da je $m = 7$ (4 informacijska bita + 3 nule) pa moramo nacrtati rešetku dubine sedam. Također imajte na umu da za dva posljednja stanja rešetke, nacrtat ćemo samo grane koje odgovaraju ulazima "sve 0", jer se ulazni informacijski niz postavlja s $k(L–1) = 2$ nule. To također znači da je stvarna duljina ulaznoga niza 5, što će nakon popunjavanja s dvije nule, porasti na sedam. Dijagram rešetke za ovaj slučaj prikazuje slika 6.27.




*Slika 6.27: Dijagram rešetke Viterbijevoga dekodiranja niza [01101111010001].*

Na ovoj slici, također je prikazan raščlanjen primljen niz **y**. Imajte na umu da crtanjem rešetke za posljednja dva stanja, razmatramo samo nulte ulaze u koder (uočite da u posljednje dvije faze, ne postoje isprekidane crte koje odgovara jednome ulazu). Mjera početnoga stanja "sve 0" postavlja se na nulu i računa se mjera iduće faze. U ovome koraku, postoji samo jedna grana za ulaz u svako stanje,

stoga ne postoji usporedba, a podaci se dodaju mjeri prethodnoga stanja (Hammingova udaljenost između toga dijela primljenoga niza i grana rešetke).

U idućoj fazi, ne postoji ni usporedba. U trećoj fazi, po prvi put imamo dvije grane koje ulaze u svako stanje. To znači da se usporedba mora napraviti ovdje i izabrat će se preživjeli *nasljednici* (*survivors*). Od dvije grane u koje ulazi svako stanje, jedno odgovara ukupno akumuliranim mjerama koje su preostale kao preživjele, a druga grana se briše. Ako u bilo kojemu stanju, dvije staze imaju istu mjeru, svaka od njih može (p)ostati nasljednica (preživjela). U dijagramu rešetke, takvi slučajevi označena su znakom upitnika "**?**". Postupak se nastavlja do konačnoga stanja rešetake "sve 0", a zatim počevši od toga stanja, krećemo se stazama preživjelih, prema početnome stanju "sve 0".

Ova staza, koja se obilježava kao "mukotrpna staza" (*heavy path*) kroz rešetku, optimalna je staza. Ulazan niz bitova koji se podudara tom stazom je [1100000] gdje posljednje dvije nule nisu informacijski bitovi, već su dodane za vratiti koder u stanje "sve 0". Dakle, informacijski niz je 11000. Odgovarajuća kodna riječ za odabranu stazu je [**11 10 10 11 00** 00 00], čija je Hammingova udaljenost 4 od primljenoga niza. Niti jedna druga staza kroz rešetku primljenoga niza **y**, nema Hammingovu udaljenost manju od 4. Primljen niz ima 2 pogreške.

Za dekodiranje mekom odlukom slijedi sličan postupak s kvadriranim Euklidovim udaljenostima koje zamjenjuju Hammingove udaljenosti.

## 6.7. 6.7. Viterbijev algoritam dekodiranja tvrdom odnosno mekom odlukom

### 6.7.1. 6.7.1. MJERA GRANE I MJERA PUTA

- Želimo dobiti niz bitova najvjerojatnije poruke
- Dobili smo (vjerojatno oštećen) niz bitova
- Viterbijev algoritam za zadani $K$ i $r$:
  - Radimo postupno za izračunati niz najvjerojatnije poruke.
  - Koristimo dvije mjere:
- Mjeru grane BM (*branch metric*) BM(*xmit*, *rcvd*) koja je proporcionalna negativnome logaritmu funkcije vjerojatnosti (*log likelihood*)[113], tj. negativnome logaritmu vjerojatnosti da smo primili *rcvd*, ako je poslan *xmit*.
  - Dekodiranje "tvrdom odlukom": Koristimo digitalizirane bitove, računamo Hammingovu udaljenost između *xmit* i *rcvd*. Manja udaljenost je vjerojatnija ako je BER < 1/2.
  - Dekodiranje "mekom odlukom": Izravno se koristiti funkcija primljenih napona.
- Mjera puta PM (*path metric*) PM[$s$, $i$] za svako stanje $s$ od mogućih $2^{K-1}$ odaslanih stanja u vremena prijenosa bita $i$, gdje je $0 \le i < L$ = duljina(poruke).
  - PM[$s$, $i$] = najmanji zbroj BM(*xmit*, *rcvd*) nad svim nizovima poruka $m$ koje smještaju odašiljač u stanje $s$ u vremenu $i$.
  - PM[$s$, $i$+1] računa se iz PM[$s$, $i$] i $p_0[i]$, …, $p_{r-1}[i]$

### 6.7.2. 6.7.2. VITERBIJEVO DEKODIRANJE TVRDOM ODLUKOM

- Nakon prijema svakoga bita, on se odmah digitalizira kao 0 ili 1 usporedbom naponskoga praga.
- Gubimo informacije o tome koliko je bit "dobar", jer se "1" tretira jednako za naponsku razinu $u$ = 0,9999 [V] kao i za naponsku razinu $u$ = 0,5001 [V]

[113] Funkcija se definira kao prirodan logaritam funkcije vjerojatnosti, $l(\theta; x) = \log L(\theta; x)$

- Mjera grane koja se koristi u Viterbijevome dekoderu za dekodiranje tvrdom odlukom je Hammingova udaljenost između digitaliziranih primljenih naponskih razina i očekivanih paritetnih bitova.
- Odbacivanje informacija nije (skoro) nikada dobra ideja prilikom donošenja odluka.

*6.7.2.1.* *6.7.2.1. Viterbijev algoritam koji koristi dekodiranje tvrdom odlukom*

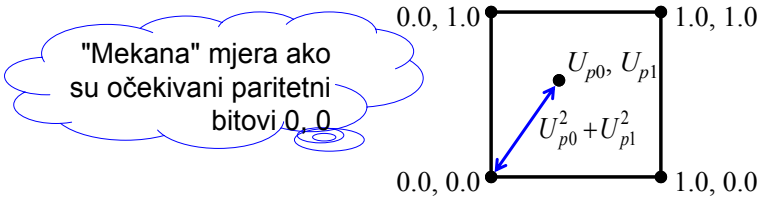
- Mjere grana, mjere pojedinačan doprinos negativnoj logaritamskoj vjerojatnost uspoređujući primljene paritetne bitove u odnosu na moguće prenesene paritetne bitove izračunate iz mogućih poruka.
- Mjera puta PM[$s$, $i$] proporcionalna je negativnoj logaritmiranoj vjerojatnosti da odašiljač bude u stanju $s$ u vremenu $i$, uz pretpostavku da je najvjerojatnija poruka dužine $i$ koja napušta odašiljač u stanju $s$.
- Najvjerojatnija poruka je ona poruka koja proizvodi najmanju mjeru puta, PM[$s$, $N$].
- U bilo kojemu trenutku postoji $2^{K-1}$ najizglednijih poruka koje pratimo. Vremenska složenost algoritma raste eksponencijski ograničenjem duljine $K$, a linearno dužinom poruke (za razliku od eksponencijskoga porasta dužine poruke za jednostavna pobrojenja).

*6.7.2.2.* *6.7.2.2. Mjera grane za dekodiranje tvrdom odlukom*

- Mjera grane BM (*branch metric*) = Hammingova udaljenost između očekivanih paritetnih bitova (na predajnoj strani kanala - koder) i dobivenih paritetnih bitova (na prijemnoj strani kanala - dekoder).
- Računa se BM za svaki prijelazni luk (usmjerenu crtu) u dijagramu rešetke.

### 6.7.3. 6.7.3. VITERBIJEVO DEKODIRANJE MEKOM ODLUKOM

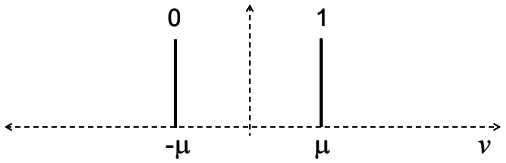
- U praksi, za svaki primljen paritetan bit primatelj dobije različite naponske razine $u$.
  - Primatelj šalje $u_0$ ili $u_1$ [V] ($-\infty \le u \le \infty$) tj. ima beskonačno mnogo pozitivnih i negativnih naponskih razina, uz pretpostavku adicijskoga Gaussovoga šuma.
- Ideja je "pročešljati"primljene naponske razine prema dekoderu prije digitalizacije.
- Definirajmo "*meku*" *mjeru grane* kao kvadriranu Euklidovu udaljenost između primljenih i očekivanih napona (slika 6.28).



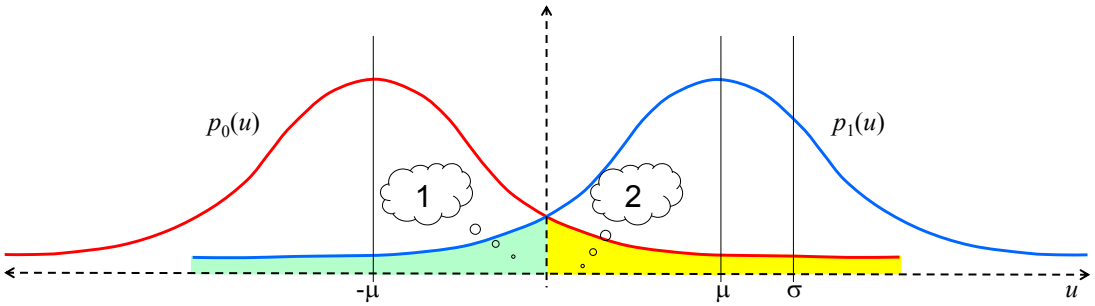

*Slika 6.28: "Meka" mjera ako su očekivani paritetni bitovi 00*

- Dekoder koji koristi meko odlučivanje, odabere put minimalnoga zbroja kvadriranih Euklidovih udaljenosti između primljenih i očekivanih napona.
  - Iako se koristi isti algoritam, različite su vrijednosti za BM i PM.

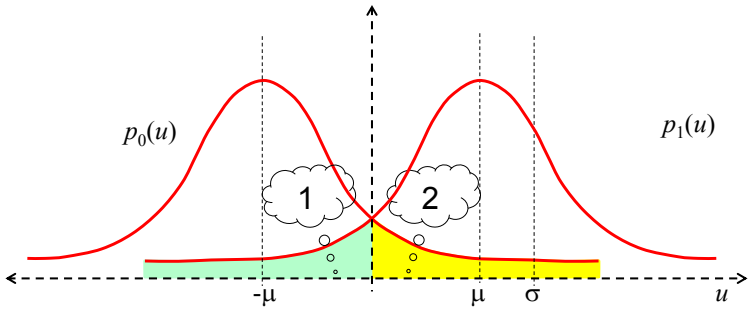
Ako šum djeluje na spektar dvaju signala koji se koriste za predstaviti bitove 0 i 1, onda je na prijemnoj strani kanala veoma teško odlučiti: radi li se o prijemu 0 ili 1? Pri prostiranju signala, energija s jednoga, prelijeva se u drugi signal. Slike 6.29.b i 6.29.c, prikazuju dva različita slučaja za različite odnose $S/N$.




a) Dva signala predstavljaju bitove 1 i 0




b) Odnos signal/šum: *S*/*N* = 2 rasprostire se zajedno sa signalom




c) Odnos signal/šum: *S*/*N* = 4 rasprostire se i prelijeva energiju iz jednoga područja u drugo

*Slika 6.29: Prelijevanje energije iz jednoga područja u drugo*

Ako je dodatan šum malen, tj. njegova varijanca je mala (jer je *snaga šuma* = *varijanca*), onda će raspršenjem, varijanca postati još manja, kao što možemo vidjeti na slici 6.29.c u odnosu na sliku 6.29.b. Iz ovih slika intuicijski možemo vidjeti, da bi se pri dekodiranju napravila manje vjerojatna pogreška, ako bi odnos *S*/*N* bio velik ili ako bi varijanca šuma bila mala.

Dekodiranje tvrdom odlukom (*hard decision decoding*) znači da se između dvaju signala obično odabire jednostavan prag odluke, tako da, ako je primljen napon pozitivan onda se signal dekodira kao 1 inače se dekodira kao 0. U vrlo jednostavnim uvjetima to znači dekodiranje maksimalnom vjerojatnošću (*maximum likelihood decoding*).

Ovom metodom odlučivanja možemo brojčano izraziti (kvantificirati) napravljenu pogrešku. Vjerojatnost dekodiranja 0, ako je poslana 1, predstavlja funkcija dvaju osjenčanih područja kako prikazuje slika 6.29.

Područje 1 predstavlja onu energiju koja pripada signalu za 1, a što se "prelila" u susjedno područje odluke pa se time bit pogrešno dekodirao kao 0 i u području 2, a to je energija koja pripada bitu 0, ali se prelila u susjedno područje. To se oduzima od primljenoga napona pa stoga može uzrokovati pogrešku pri odluci. Obzirom da je poslana 1, vjerojatnost da će se 1 dekodirati kao 0 je:

$$Pe1 = \frac{1}{2} erfc\left(\frac{\mu - U_t}{\sqrt{2}\sigma}\right)^{114}, \qquad (6.9)$$

$u_t$ … napon praga odluke, što je u našem slučaju 0,
σ … varijanca šuma ili njegova snaga.

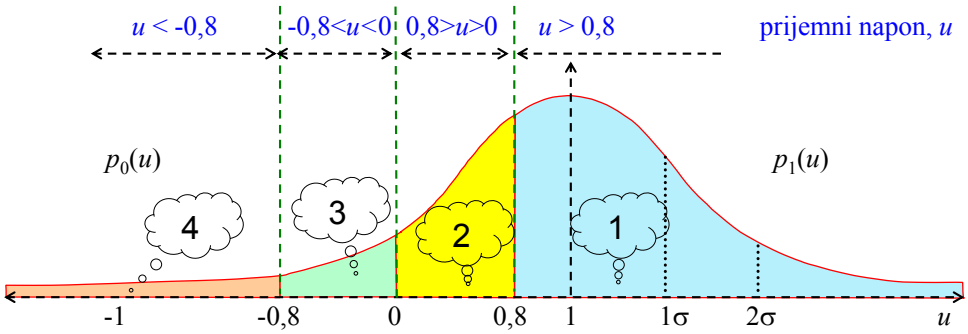
Napomena: Program C++ za izračun funkcije pogreške (*erfc*).
Jezik C++11 u zaglavlju `cmath` ima funkcije `erf()` i `erfc()`. Obje funkcije su preopterećene pa prihvaćaju argumente tipa `float`, `double` i `long double`. Za složene <`double`>, Faddeeva paket (Faddeeva package) omogućuje C++ kompleksnu primjenu <`double`>.

[114] *erfc* … error function

Možemo preurediti ovu jednadžbu kao funkciju *S*/*N*:

$$Pel = \frac{1}{2} erfc\left(\sqrt{\frac{S}{N}}\right). \tag{6.10}$$

Ovo je poznata jednadžba odnosa pogrešnih bitova. Vidimo da se pretpostavlja dekodiranje tvrdom odlukom. Ali što ako učinimo sljedeće, umjesto da imamo samo dva područja odluke, podijelimo raspoloživo područja u četiri područja kao što prikazuje slika 6.30.




*Slika 6.30: Stvaranje četiri područja za odluku*

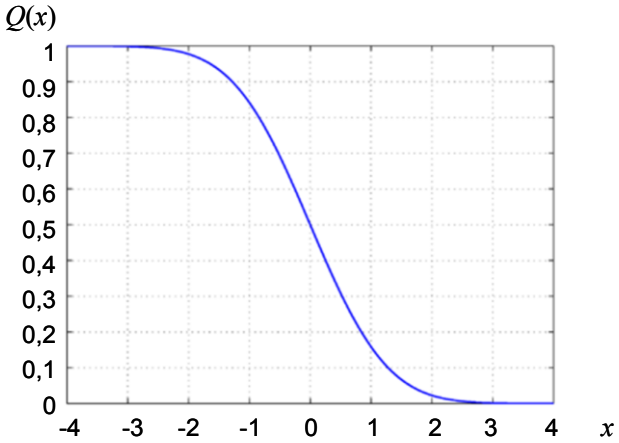
Vjerojatnost da je odluka ispravna može se izračunati iz područja ispod Gaussove krivulje. Četiri prikazana područja organizirana su na sljedeći način:

Područje 1 = ako je primljen napon *u* veći od 0,8 V
Područje 2 = ako je primljen napon *u* veći od 0, ali manji od 0,8 V
Područje 3 = ako je primljen napon *u* veći od –0,8 V, ali manji od 0 V
Područje 4 = ako je primljen napon *u* manji od –0,8 V

Sada se pitamo: Ako primljeni napon padne u područje 3, kolika je onda vjerojatnost pogreške da se poslala 1? Za dekodiranje tvrdom odlukom, odgovor je jednostavan. On se može izračunati iz jednadžbe. A kako bismo izračunali slične vjerojatnosti za prostor s višestrukim područjima? Koristimo *Q* funkciju koja je dana u tablicama mnogih knjiga.

### 6.7.3.1. 6.7.3.1. *Q*-funkcija

*Q*-funkciju prikazuje slika 6.31.




*Slika 6.31: Q funkcija*

*Q*-funkcija definira se kao

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^{\infty} \exp\left(-\frac{u^2}{2}\right) du\,. \tag{6.11}$$

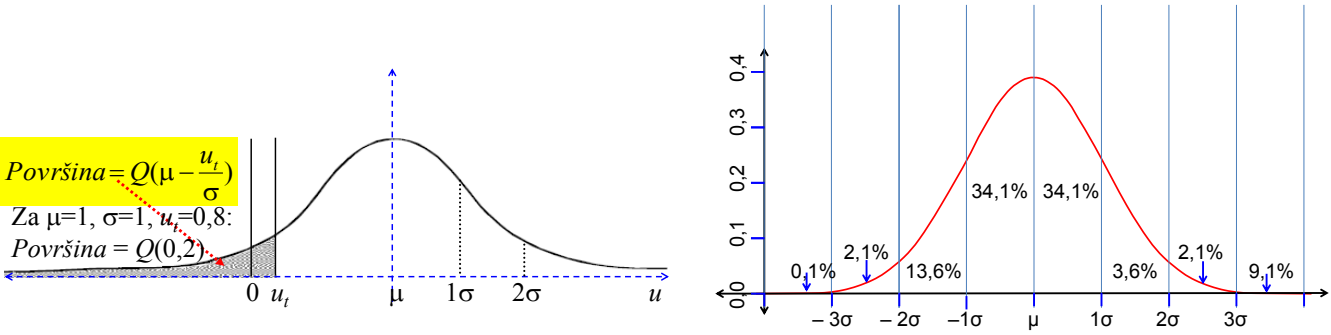
Dakle,

$$Q(x) = 1 - Q(-x) = 1 - \Phi(x), \tag{6.12}$$

gdje je $\Phi(x)$ kumulacijska funkcija gustoće normalne Gaussove razdiobe.

### 6.7.4. 6.7.4. PRAKTIČNA PRIMJENA *Q* FUNKCIJE

*Q* funkcija definira nam površinu ispod "repa" krivulje što definira udaljenost od srednje vrijednosti prema bilo kojoj drugoj vrijednosti. Dakle *Q*(2) definira signal s aritmetičkom sredinom 2 i daje nam vrijednost vjerojatnosti koja je jednaka ili veća od 4 (slika 6.32).




*Slika 6.32: Q funkcija je jednostavan način da se utvrdi vjerojatnost normalno raspodijeljene varijable*

Pretpostavlja se da je $u_t = 0{,}8$ (što je standardna vrijednost broja za 4 razine), ali to može biti bilo koji broj. Jednadžbe se mogu napisati skoro intuicijski, one su tako očite.

$P_{e1}$ (vjerojatnost da se poslala 1 ako je primljen napon u području 1) $= 1 - Q(-u_t/\sigma)$

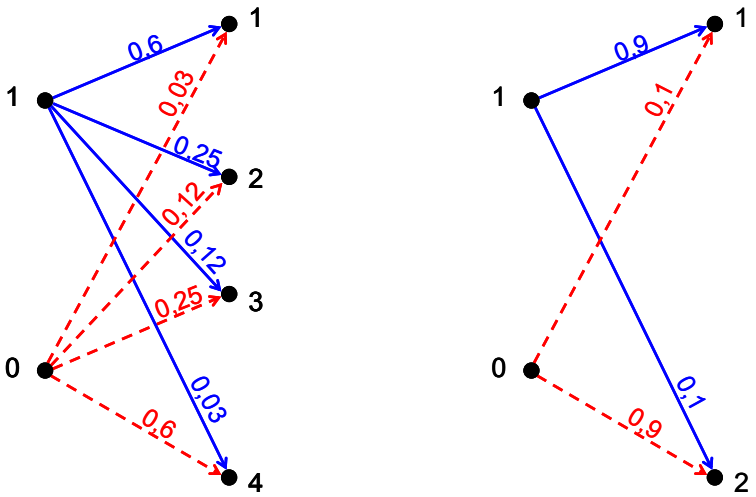
$P_{e4}$ (vjerojatnost da se poslala 1 ako je primljen napon u području 4) $= Q(2(A + u_t/\sigma)$

$P_{e2}$ (vjerojatnost da se poslala 1 ako je primljen napon u području 2) $= 1 - P_{e1} - Q(A/\sigma)$

$P_{e3}$ (vjerojatnost da se poslala 1 ako je primljen napon u području 3) $= 1 - P_{e1} - P_{e2} - P_{e4}$

Izračunali smo da je ovaj $S/N = 1$ ako je $u_t = 0{,}8$, a $A = 1{,}0$. Također smo pretpostavili da su bitovi 0 i 1 jednako vjerojatni. Ovo se također nazivaju *apriorne* vjerojatnosti za bitove 0 i 1 i gotovo se uvijek pretpostavlja da su jednake u komunikaciji osim za radarske sustave, gdje apriorne vjerojatnosti obično nisu poznate.

Dakle, područje 4 ili $P_{e4}$ jednako je $Q(1{,}8)$ što se može iščitati iz tablica. Područje 1 onda je jednako $1–Q(0{,}2)$. Ostala područja mogu se brzo izračunati na ovaj način. Možemo pokazati uvjetne vjerojatnosti izračunate na grafički način (slika 6.33).




*Slika 6.33: Uvjetne vjerojatnosti dekodiranja 0 ili 1 ovisno o veličini primljenoga napona, ako se napon kvantizira u 4 razine u odnosu na dvije razine.*

Ovaj postupak diobe prostora odluke u područja veća od dva, kao što je ovaj primjer s 4 razine, naziva se *dekodiranje mekom odlukom* (*soft decision decoding*). Ove vjerojatnosti također se nazivaju *prijelazne vjerojatnosti*.

Postoje 4 različite vrijednosti napona za svaki primljen signal kojima donosimo odluku. Među signalima, kao i u stvarnome životu, više informacija označava bolje odluke. Dekodiranje mekom

odlukom unaprjeđuje osjetljivost mjere dekodiranja. U tome slučaju poboljšavaju se svojstva za čak 3 dB s 8 razina.

Sada, za izračunati mjeru meke odluke, napravili smo tablicu od prije navedenih vjerojatnosti.

| poslano | $u_4$ | $u_3$ | $u_2$ | $u_1$ |
|---|---|---|---|---|
| 1 | 0,03 | 0,12 | 0,25 | 0,60 |
| 0 | 0,6 | 0,25 | 0,12 | 0,03 |

Izračuna se prirodan logaritam svake od ovih vrijednosti i normalizira ga se tako da je jedna od vrijednosti 0. Nakon maloga rukovanja brojevima, dobivamo

| poslano | $u_4$ | $u_3$ | $u_2$ | $u_1$ |
|---|---|---|---|---|
| 1 | 0,0 | –1 | –4 | –10 |
| 0 | –10 | –4 | –1 | 0 |

U dijelu o dekodiranju, izračunali smo Hammingovu mjeru množenjem dobivenih bitova kodnim riječima. Sada radimo istu stvar, osim što umjesto primanja 0 i 1, primamo jedan od ovih napona. Dekoder koristi mjeru za taj napon iz tablice kao što je ona iznad koju čuva u svojoj memoriji i računa sljedeće.

Pretpostavimo da smo primili naponski par ($u_3$, $u_2$). Dopuštene kodne riječi su 01 i 10.

Mjera za 01 = $p(0/u_3) + p(1/u_2) = (–4) + (–4) = –8$

Mjera za 10 = $p(1/u_3) + p(0/u_2) = (–1) + (–1) = –2$

Promatrajući samo ove dvije mjere možemo kazati da je pojava 01 u odnosu na 10 puno vjerojatnija, naglašavaju se razlike i potpomaže otkrivanje rezultata dekodiranja.

### 6.7.5. 6.7.5. LOGARITAMSKI ODNOS

Postoje i drugi načini poboljšanja svojstava dekodiranja "*poigravanjem*" mjerom. Važna mjera koju treba poznavati zove se *mjera logaritamske vjerojatnosti* (*log likelihood metric*). Ova mjera uzima u obzir vjerojatnost pogreške kanala, a definira se kao

$$\text{mjera podudarnosti} = \frac{\log_{10} 2(1-p)}{\log_{10} 2} \tag{6.13}$$

$$\text{mjera neslaganja} = \frac{\log_{10} 2(p)}{\log_{10} 2} \tag{6.14}$$

Za $p = 0{,}1$

Mjera podudaranja = –20

Mjera neslaganja = –1

Dakle, ako smo primili bitove 01, a kodna riječ je 00, ukupna mjera bit će –20 + –1 = –21, a mjera za potpuno podudaranje bit će –40.

Fano algoritam koji se koristi za serijsko dekodiranje, upotrebljava malo drugačiju mjeru, a svrha je poboljšati osjetljivost.

Viterbijevo dekodiranje vrlo je važno, jer ono također vrijedi i za dekodiranje blok-kodova. Ovaj oblik dekodiranja rešetkom također se koristi za *modulaciju kodiranom rešetkom* TCM (*trellis coded modulation*).

# 17. *Laboratorijska vježba 16: Viterbijevo dekodiranje tvrdom odlukom*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*



Koder i dekoder uvijek započinju rad iz stanja "sve 0".

Primjer u vježbi

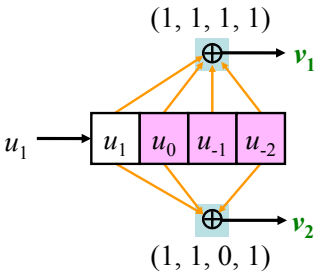
Polinom-generatori za (2, 1, 4) kod su:

$$g_1 = 1\cdot x^3 + 1\cdot x^2 + 1\cdot x^1 + 1\cdot x^0$$

i

$$g_2 = 1\cdot x^3 + 1\cdot x^2 + 0\cdot x^1 + 1\cdot x^0$$

Pokrenuti: Coding the (2, 1, 4) code.pps

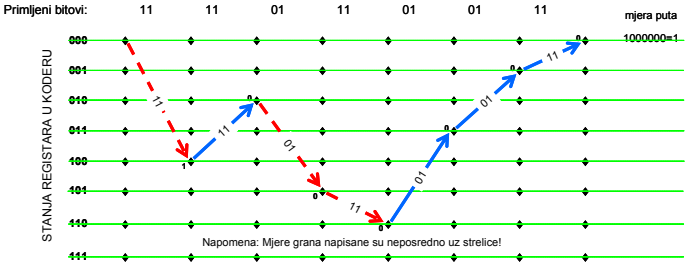



| polinom-generatori | ulaz | stanje registra | izlaz $v_1v_2$ | novo stanje | polinom-generatori | ulaz | stanje registra | izlaz $v_1v_2$ | novo stanje |
|---|---|---|---|---|---|---|---|---|---|
| | **0** | **000** | ⟶ | **000** | | **0** | **100** | ⟶ | **010** |
| $x^3\oplus x^2\oplus x^1\oplus x^0$ | 0 | 000=0 | **00** | | $x^3\oplus x^2\oplus x^1\oplus x^0$ | 0 | 100=1 | **11** | |
| $x^3\oplus x^2\oplus 0\oplus x^0$ | 0 | 0 0=0 | | | $x^3\oplus x^2\oplus 0\oplus x^0$ | 0 | 1 0=1 | | |
| | **1** | **000** | ⟶ | **100** | | **1** | **100** | ⟶ | **110** |
| $x^3\oplus x^2\oplus x^1\oplus x^0$ | 1 | 000=1 | **11** | | $x^3\oplus x^2\oplus x^1\oplus x^0$ | 1 | 100=0 | **00** | |
| $x^3\oplus x^2\oplus 0\oplus x^0$ | 1 | 0 0=1 | | | $x^3\oplus x^2\oplus 0\oplus x^0$ | 1 | 1 0=0 | | |

Primjer: Koder počinje iz stanja 000, **u** = [**1011000**], **v** = ? → Viterbijevo dekodiranje tvrdom odlukom!

| stanja ulaz | bit | stanje prije | prijelaz izlaz | stanje nakon | bit | stanje prije | prijelaz izlaz | stanje nakon | bit | stanje prije | prijelaz izlaz | stanje nakon | bit | stanje prije | prijelaz izlaz | stanje nakon |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **u** | **1** | **000** | → | **100** | **0** | **100** | → | **010** | **1** | **010** | → | **101** | **1** | **101** | → | **110** |
| $g_1$ | 1 | 000=1 | **11** | | 0 | 100=1 | **11** | | 1 | 010=0 | **01** | | 1 | 101=1 | **11** | |
| $g_2$ | 1 | 0 0=1 | | | 0 | 1 0=1 | | | 1 | 0 0=1 | | | 1 | 1 1=1 | | |

(De)kodiran izlaz je: **v** = [**11 11 01 11 01 01 11**], a dijagram rešetke (*trellis diagram*) je:



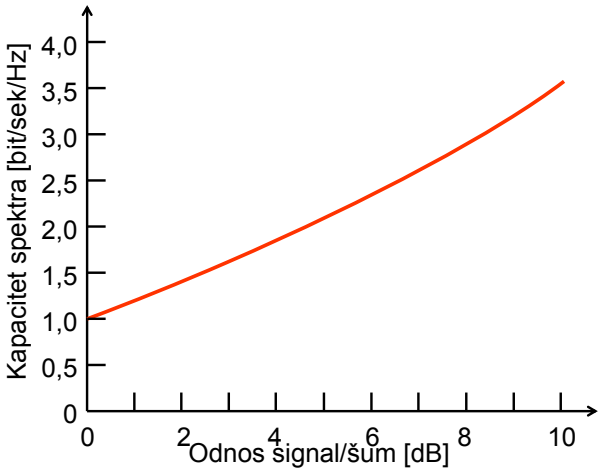
# 7. POGLAVLJE 7 - TURBO KODOVI

## 7.1. 7.1. Uvod

Turbo kodovi, predstavljaju još jedan razred kodova pristupom prema kapacitetu, a otkriveni su 1993. Nakon toga su postali prvi izbor shema kodiranja koje (su) se korist(il)e u primjenama kao npr. svemirske komunikacije satelitima.

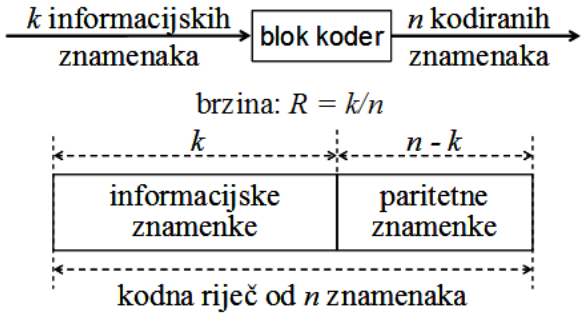
Turbo kod (ili Berrou kod[115]) linearan je blok-kod koji se ustroji iz zaključenoga i probušenoga, sustavno kodiranoga konvolucijskoga koda na određen propisan način, koristeći dvije različite kodne riječi konvolucijskoga koda. Turbo kod iste podatke kodira dva puta tako da dekoderi dviju konvolucijskih kodnih riječi potpomažu rad jedan drugoga. Kada se dekodiraju pomoću iteracijskoga dekodera mekim odlučivanjem, turbo kodovi spadaju među kodove s najboljim poznatim svojstvima za prevladavanje smetnji dodatnoga bijeloga šuma u Gaussovome kanalu.

Turbo kod predstavlja spoj opisanih kodova. To je razred ulančanih kodova gdje postoji sklop za preplitanje (*interleaver*) između dvaju paralelnih ili serijskih kodera. Sklop za preplitanje, stvara kodne riječi velikih duljina i izvrsnih svojstava, posebno za male odnose signal/šum *S/N* (*Signal-Noise Ratio*). Korištenje tih kodova, omogućuje dobitak oko 0,7 [dB] Shannonove granice na malim *S/N* (slika 7.1).

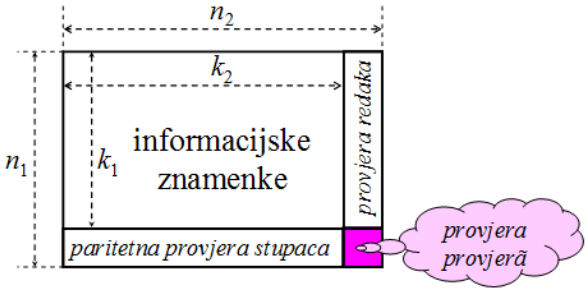



*Slika 7.1: Shannonova granica kapaciteta kanala*[116]

Turbo kod možemo zamisliti kao usavršenu strukturu *ulančanoga koda* plus *iteracijski*[117] algoritam za dekodiranje pridružen kodnome nizu. Podsjetimo na do sada obrađene učinkovite metoda kodiranja i dekodiranja. To su sustavni (*n*, *k*) blok-kodovi (slika 7.2.a) i kodovi *dvodimenzijskih polja* (slika 7.2.b). U obje vrste utisnuti su paritetni bitovi.




a) *Struktura kodne riječi blok-koda*




b) *Kodiranje dvo-dimenzijskoga polja (redak-stupac)*

*Slika 7.2: Struktura kodnih riječi za jednodimenzijske i dvodimenzijske kodove*

---

[115] Berrou i suradnici (1993)

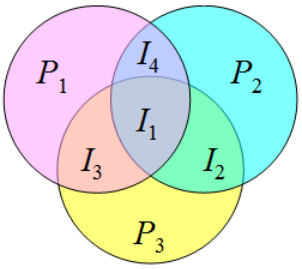
[116] Dijeli maksimalnu brzinu prijenosa informacije po jedinici širine pojasa $E_b/N_0$ u dva područja. Jedno područje dopušta prijenos bez pogrešaka, a drugo ne. (*A Mathematical Theory of Communication*, 1948., Theorem 11.)

[117] *iteracija* … ustrajno ponavljanje

Odnosi paritetnih i informacijskih bitova mogu se prikazati Vennovim dijagramima kao na slici 7.3.

| $I_1$ | $I_2$ | $I_3$ | $I_4$ | $P_1$ | $P_2$ | $P_3$ |
|---|---|---|---|---|---|---|

$n = 7, k = 4, n - k = 3$




$d = 3$

*Slika 7.3: Vennovim dijagramima prikazani odnosi između informacijskih i paritetnih bitova*

Načelo *preplitanja bitova* iz različitih kodnih riječi prikazuje slika 7.4, a koristi se npr. u CD uređajima i općenito u sustavima koji su izloženi *praskovitim* pogreškama.

| … | $I_8$ | $I_7$ | $I_6$ | $I_5$ | $I_4$ | $I_3$ | $I_2$ | $I_1$ | → |
|---|---|---|---|---|---|---|---|---|---|

ulazni podaci

memorija za preplitanje

| | | | |
|---|---|---|---|
| $I_1$ | $I_5$ | $I_9$ | $I_{13}$ |
| $I_2$ | $I_6$ | $I_{10}$ | $I_{14}$ |
| $I_3$ | $I_7$ | $I_{11}$ | … |
| $I_4$ | $I_8$ | $I_{12}$ | … |

| … | $I_{14}$ | $I_{10}$ | $I_6$ | $I_2$ | $I_{13}$ | $I_9$ | $I_5$ | $I_1$ | → |
|---|---|---|---|---|---|---|---|---|---|

izlazni prepleteni podaci za prijenos

*Slika 7.4: Preplitanje informacijskih bitova*

Koristeći prilično složene tehnike kodiranja-dekodiranja zajedno s metodama obrade signala, mogu se prikriti praskovite pogreške do 12.000 bitova podataka, koji odgovaraju npr. stazi dužine 7,5 mm na CD.

Kodiranje konvolucijskih kodova opisalo se u prethodnome poglavlju, a za analizu njihova rada, osim pregledne tablice koriste se dijagrami stabla, stanja i rešetke. Za dekodiranje, koriste se metode dekodiranja rešetkom, serijsko i Viterbijevo dekodiranje. Primjer blok-koda, čija je sposobnost ispravke pogrešaka $t = 10$ bitova, jest BCH (127, 64) kod. Kodovi, RS (16, 8) kod ili RS (64, 32) kod, imaju sposobnost ispravke $t = 4$ simbola. Kraći kodovi kao što je Golay (23, 12) kod, ima sposobnost ispravke $t = 3$ bita. Na slici 5.12.b (za DPSK modulaciju), uspoređuju se njihova svojstva brzine pogrešaka i svojstva nekih drugih kodova.

## 7.2. 7.2. Kodovi sastavljeni od jednostavnijih kodova

Izvedba kodova ovisi o minimalnoj udaljenosti blok-kodova i *slobodnoj udaljenosti*[118] konvolucijskih kodova. Da bi oblikovali blok-kodove određene brzine i velike minimalne udaljenosti, moramo povećati duljinu blok-koda $n$. Povećanjem $n$, povećava se složenost dekodiranja. U većini algoritama dekodiranja, složenost dekodiranja eksponencijski raste povećanjem duljine blok-koda.

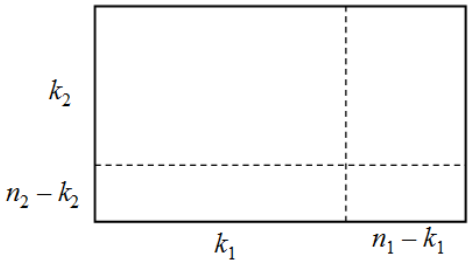
Za konvolucijske kodove, povećanje *slobodne udaljenosti* pri određenoj brzini, zahtijeva povećanje ograničene duljine koda. No, povećanje ograničene duljine koda, povećava broj stanja u kodnoj rešetci, što zauzvrat, povećava složenost dekodiranja.

Pri generiranju složenijih kodova, većina metoda povećanja duljine blok-koda temelji se na sjedinjenju jednostavnih kodova. Dekodira se metodom za dekodiranje jednostavnih komponentskih kodova. Rezultirajuće dekodiranje predstavlja pod-optimalnu shemu dekodiranja, koja u većini slučajeva radi na zadovoljavajući način. U nastavku se ukratko opisuju sljedeće tehnike: *kodovi umnoška*, *ulančani kodovi* i *turbo kodovi*.

### 7.2.1. 7.2.1. ITERACIJSKO DEKODIRANJE

Struktura kodova umnoška (*product codes*) ili kodova polja (*array codes*) vrlo je slična križaljci. Generiraju se pomoću dvaju linearnih blok-kodova raspoređenih u matričnome obliku. Dva linearna blok-koda, jedan s parametrima $n_1$, $k_1$, $d_{min1}$, a drugi s parametrima $n_2$, $k_2$, $d_{min2}$, prikazuje slika 7.5

[118] Ono što je za blok-kodove minimalna Hammingova udaljenost između kodnih nizova, to se za konvolucijske kodove zove slobodna udaljenost (*free distance*) $d_{free}$ (vidi poglavlje 9)



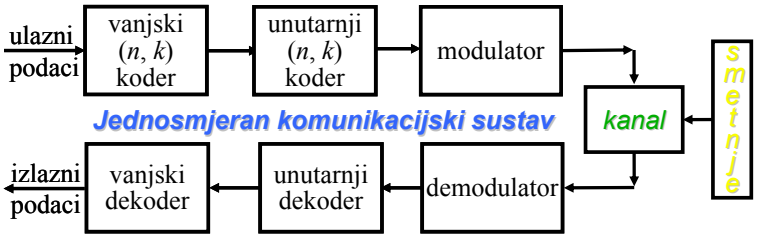

*Slika 7.5: Struktura koda umnoška.*

Rezultirajući kod je linearan ($n_1n_2$, $k_1k_2$) blok-kod. Minimalna udaljenost rezultirajućega koda, umnožak je minimalne udaljenosti sastavnih kodova, $d_{min} = d_{min1} \cdot d_{min2}$. Kod je sposoban ispraviti $\left\lfloor \frac{d_{min1} d_{min2} - 1}{2} \right\rfloor$ pogrešaka, koristeći složeno optimalno dekodiranje. Kod se dekodira koristeći svojstva sastavnih kodova. Korištenjem rednih kodova, možemo doći do najtočnije vrijednosti bitova, a zatim, koristeći stupčane kodove poboljšati te točnosti.

Ovaj postupak može se ponavljati iteracijskim načinom, poboljšavajući kakvoću "pogotka" pri svakoj iteraciji i poznat je kao *iteracijsko dekodiranje*. Vrlo je sličan načinu na koji se rješava križaljka. Za koristiti ovaj postupak dekodiranja, trebamo sheme dekodiranja za redne i stupčane kodove koje su sposobne *pogađati* svaki pojedinačan bit. Poželjne su sheme dekodiranja mekim izlazima (vjerojatnosne vrijednosti).

### 7.2.2. 7.2.2. ULANČANI KODOVI

Ulančani kod sastoji se od dvaju kodova, *unutarnjega* (*inner*) i vanjskoga (*outer*) koda, spojenih serijski kao što prikazuje slika 7.6

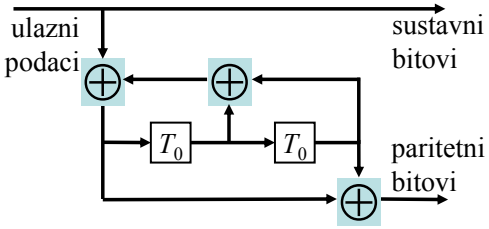



*Slika 7.6: Blok-dijagram sustava komunikacijskoga sustava s ulančanim kodiranjem.*

Unutarnji kod je konvolucijski kod, a vanjski kod je Reed-Solomonov kod. Ako je unutarnji kod, neki ($n$, $k$) kod, onda se sastav unutarnjega kodera, digitalnoga modulatora, valnoga oblika kanala, digitalnoga demodulatora i unutarnjega dekodera, može razmatrati kao kanal čiji su ulaz i izlaz binarni blokovi duljine $k$, ili elementi $q$-struke abecede gdje je $q = 2^k$. Sada se na ovome izlaznome kanalu ($q$-struki ulaz i $q$-struki izlaz), može koristiti Reed-Solomonov kod (vanjski kod), da bi se osigurala zaštita od daljnjih pogrešaka. Ako su brzine unutarnjega koda, $r_c$ i vanjskoga koda $R_c$, onda njihov umnožak predstavlja brzinu ulančanoga koda.

Minimalna udaljenost ulančanoga koda, također je umnožak minimalnih udaljenosti unutarnjega i vanjskoga koda. Za ulančane kodove, izvedba unutarnjega koda ima velik utjecaj na ukupnu učinkovitost koda. To je razlog zašto se kao unutarnji kodovi obično koriste konvolucijski kodovi koji ulaz dekodiraju mekom odlukom pomoću Viterbijeva algoritma.

Razlikuju se ne-sustavni konvolucijski NSC (*non-systematic convolutional*) kodovi, jer koderi ne stvaraju odvojeno podatkovne i zalihosne bitove i nemaju povratne veze. Pojam "konvolucijski" proizlazi iz činjenice da izlazni niz predstavlja ulazni niz konvoluiran impulsnim odzivom kodera. U koder je moguće uključiti povratnu informaciju. Za takve kodere kaže se da su rekurzijski. NSC kodovi ne dopuštaju paralelno ulančavanje pa je potreban sustavan kod. Za velike $S/N$ i za izvedbu klasičnoga NSC koda, BER je bolji od klasičnoga sustavnoga konvolucijskoga koda iste ograničene duljine. Za male $S/N$, vrijedi obrnuta tvrdnja, jer sustavan konvolucijski kod ima bolja svojstva. *Rekurzijski* sustavni konvolucijski RSC (*recursive systematic convolutional*) kodovi mogu se napraviti od NSC kodova, povezivanjem jednoga od izlaza kodera izravno na ulaz, a drugi otežani priključci stanja posmičnoga registra također se vraćaju na ulaz. Primjer RSC kodera prikazuje slika 7.7

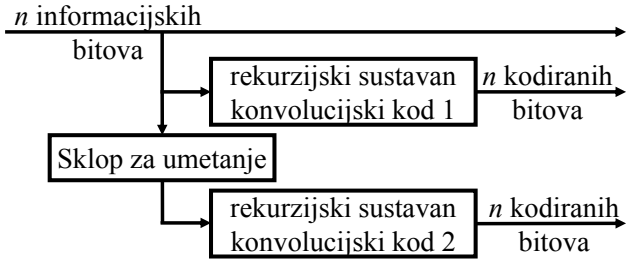



*Slika 7.7: Primjer RSC kodera uz n = 3.*

Tvorba *rekurzijskih kodova* znači da stanje kodera ovisi o prošlim izlazima i ulazima. Tako, ulazni niz konačne duljine može generirati izlazni niz beskonačne duljine, za razliku od NSC koda. To utječe na ponašanje uzoraka pogrešaka, jer pojedina pogreška u poruci izvornih bitova proizvodi beskonačan broj paritetnih pogrešaka pa je ukupan učinak bolji. NSC kodovi, linearni su isto kao i RSC kodovi.

## 7.3. 7.3. Turbo kodovi

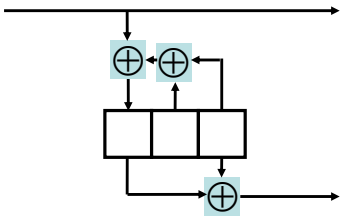
Turbo kodovi poseban su razred ulančanih kodova, jer između dvaju paralelno ili serijski spojenih kodera, imaju sklop za preplitanje (*interleaver*). Postojanje sklopa za preplitanje. rezultira vrlo velikim duljinama kodnih riječi s izvrsnim svojstvima, posebno za male odnose *S/N*, jer se omogućuje dobitak od oko 0,7 [dB] Shannonove granice. Strukturu turbo kodera prikazuje slika 7.8




*Slika 7.8: Blok-dijagram turbo kodera.*

U strukturi kodera prikazanoj na slici 7.8, *n* informacijskih bitova ulazi u prvi koder. Isti informacijski bitovi umeću se (*interleaving*) i prepliću u drugome koderu. Budući da su koderi sustavni, svaki koder generira *n* informacijskih bitova koji mu se dovedu na ulaz, a iza koji slijedi *n* paritetnih bitova. Nakon kodiranja, preko kanala se prenosi *n* informacijskih bitova i 2*n* paritetnih bitova iz oba kodera, odnosno, ukupno 3*n* bitova. Dakle, ukupna brzina je $R = n/3n = 1/3$. Ako su poželjne veće brzine, paritetni bitovi se "preskaču" (*puncture*), odnosno, prenose se samo neki paritetni bitovi, u jednakim razmacima.

Turbo koder sastoji se od dvaju sastavnih kodova razdvojena sklopom za preplitanje duljine *n*. Sastavni kodovi su obično rekurzijski sustavni konvolucijski kodovi RSCC (*recursive systematic convolutional codes*) brzine 1/2, a obično isti kod(er) koristi se kao dva sastavna kod(er)a. Rekurzijski konvolucijski kodovi drugačiji su od ne-rekurzijskih konvolucijskih kodova zbog postojanja povratnih veza pripadnih posmičnih registara. Primjer rekurzijskoga sustavnoga konvolucijskoga kodera prikazuje slika 7.9

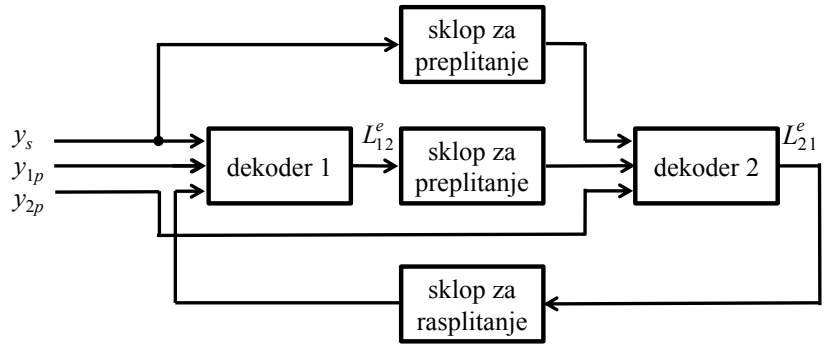


*Slika 7.9: Rekurzijski sustavni konvolucijski koder.*

Sklop za preplitanje (*interleaver*) turbo kodova, obično daje jako duge nizove (reda veličine tisuće bitova). Svojstva kodova mogu se ostvariti pametnim izborom sklopova pseudo-slučajnih nizova i sklopom za preplitanje. Niz nula postavljenih u nizu poruke ne-rekurzijskoga konvolucijskoga koda jamči, da će se koder vratiti u stanje "sve 0". Povrat u to stanje, zahtijeva opremanje (povećanje) informacijskoga niza posebnim nizom različitim od nule. Zbog postojanja sklopa za preplitanje, u većini slučajeva nemoguće je vratiti oba kodera u stanje "sve 0".

Dok turbo kodovi imaju komponente od dvaju sastavnih kodova, za njihovo *dekodiranje* prikladan je *iteracijski* algoritam. Bilo koja metoda dekodiranja, koja kao svoj izlaz daje vjerojatnosti pridružene bitovima, može se koristiti u višestrukim shemama dekodiranja. Jedna takva shema je *dekodiranje maksimalnom a posteriori vjerojatnošću* MAP (*maximum a posteriori*) i njezine inačice. Drugi popularan način manje složenosti (ali narušavanja svojstava) je Viterbijev algoritam mekim izlazom SOVA (*soft-output Viterbi algorithm*).

Korištenjem bilo kojega postupka, računaju se vjerojatnosti različitih bitova te se propuštaju u drugi dekoder. Drugi dekoder računa omjere vjerojatnosti te ih vraća prvome dekoderu. Taj postupak se ponavlja dok vrijednosti ne pokažu visoku razinu vjerojatnosti ispravnoga dekodiranja za svaki bit. U tome trenutku donosi se konačna odluka. Iteracijski postupak dekodiranja prikazuje slika 7.10

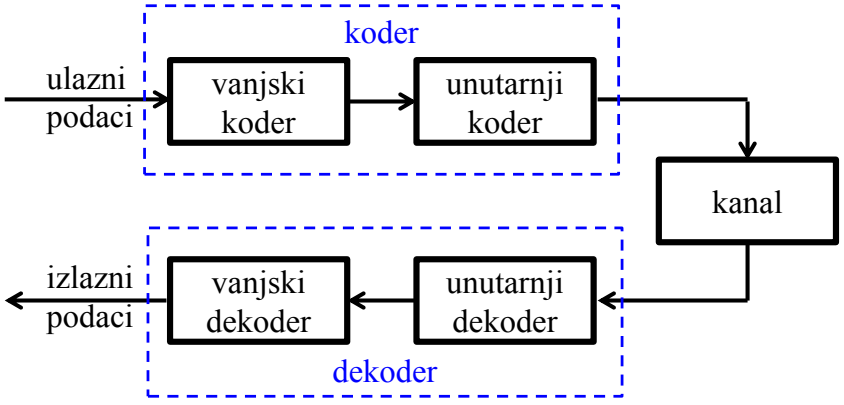



*Slika 7.10: Iteracijska shema dekodiranja za turbo kodove.*

Na ovoj slici, $\mathbf{y}_s$, $\mathbf{y}_{1p}$ i $\mathbf{y}_{2p}$ predstavljaju primljene podatke koji odgovaraju sustavnim bitovima, bitovima provjere pariteta za koder 1 odnosno za koder 2. Izrazi $L^e_{12}$ odnosno $L^e_{21}$ predstavljaju informacije koje su se razmijenile između obaju dekodera.

### 7.3.1. 7.3.1. SERIJSKI ULANČANI KODOVI

Serijski povezano kodiranje združuje dva ili više relativno jednostavnih kodova omogućujući mnogo snažniji kod, ali jednostavnijih svojstava dekodiranja nego jedan veći usporediv kod. Izlaz prvoga uređaja za kodiranje (vanjski koder) puni ulaz drugoga kodera (unutarnji koder). U dekoderu, drugi (unutarnji) kod, dekodira se prvi, a njegov izlaz, vanjski kod, dekodira se nakon toga (slika 7.11).




*Slika 7.11: Spojeno kodiranje i dekodiranje.*

Ispravak pogreška svojstvom spojenih kodova složen je za procjenu, jer njegova učinkovitost značajno ovisi o razdiobi pogrešaka. Prije otkrića turbo kodova, jedan od najmoćnijih raspoloživih kodova, bio je *ulančan kod*. On je obuhvaćao vanjski RS blok-koder iza kojega je slijedio unutarnji, konvolucijski koder, ograničene duljine 7. Ovaj kod koristi(o) se u primjenama svemirskih programa i digitalnome odašiljanju televizijskih programa (ETSI EN 301 DVB[119] standard).

Značajan nedostatak ove jednostavne sheme spajanja, je *kumulacijsko prostiranje pogreške*. U slučaju jedne pogreške na ulazu prvoga dekodera, nekoliko pogrešnih bitova može se prenijeti na ulaz

[119] DVB … *Digital Video Broadcasting*

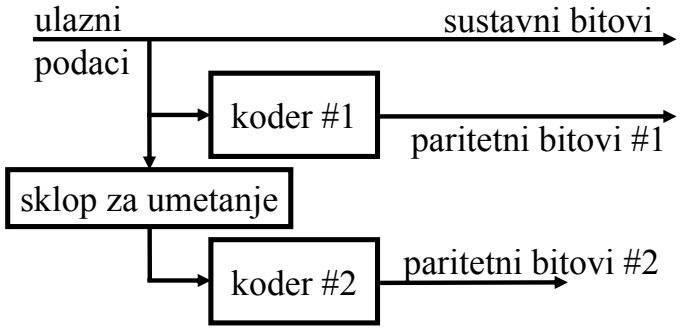
sljedećega dekodera, što rezultira još većim brojem pogrešnih bitova, a to na kraju može nadvladali sposobnost ispravke pogrešaka dekodera.

Sklop za preplitanje jednostavno permutira[120] redoslijed niza simbola. Umjesto pojave velikoga broja pogrešnih bitova unutar jedne kodne riječi, pogreške se raspodijele na veći broj kodnih riječi (uz odgovarajuće manje pogrešaka po kodnoj riječi) pa je dekodiranje uspješnije (detalji se obrađuju u laboratorijskoj vježbi 17).

### 7.3.2. 7.3.2. PARALELNO SPOJENI REKURZIJSKI SUSTAVNI KONVOLUCIJSKI KODOVI

Naziv "*turbo kod*" pomalo je pogrešan, jer se riječ "turbo" zapravo odnosi na postupak iteracijskoga dekodiranja, a ne na kodiranje. Sam naziv izvodi se iz analogije između iteracijskoga postupka dekodiranja i načela u kojemu se primjenjuju povratne informacije za poboljšati svojstava motora s unutarnjim sagorijevanjem.

U temeljnome obliku, koder turbo koda sastoji se od dvaju sustavnih kodera spojena sklopom za preplitanje, a oblik veze poznat je kao paralelno ulančenje (*parallel-concatenation*) (slika 7.12).




*Slika 7.12: Blok-dijagram turbo kodera.*

Opis rada turbo kodera:

1. Niz uzlaznih podataka primjenjuje se izravno na uređaj za kodiranje 1, a prepletena inačica istoga niza ulaznih podataka, primjenjuje se na koder 2 na slici 7.12.
2. Sustavni bitovi (tj. bitovi izvorne poruke) i dva toka paritetnih bitova (koje generiraju dva kodera) miješaju se zajedno da bi oblikovali izlaz kodera.

Iako konvolucijski kodovi predstavljaju sastavne kodere za turbo kod, u praksi su turbo kodovi zapravo blok-kodovi kojima veličinu bloka određuje veličina sklopa za preplitanje. Stoga se turbo kodovi mogu opisati kao *linearni blok-kodovi*.

Blokovska priroda turbo kodera izaziva brojne praktične probleme kao što su: točno određivanje početka i završetka kodiranoga ulaznoga niza bitova. Skoro svim shemama kodiranja zajednički problem jest, dovođenja kodera u stanje "sve 0". Nakon kodiranja, na kraju svakog bloka bitova podataka, dodaje se niz nultih bitova za prisiliti koder na povratak u stanje "sve 0". Ovaj postupak, također je poznat kao *završetak* (*termination*), a naknadno primijenjeni nizovi za *reset*, nazivaju se *završni nizovi* (*terminating sequences*). Završetak turbo kodova može se primijeniti ili samo na jedan, ili na oba sastavna kodera.

#### *7.3.2.1. 7.3.2.1. Sklopovi za preplitanje turbo kodova*

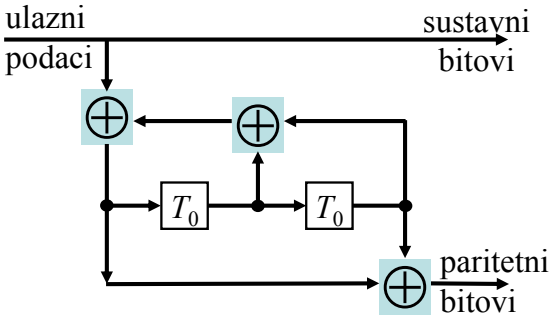
Novost paralelnoga spoja turbo kodera leži u korištenju RS kodova i uvođenju sklopa za preplitanje između dvaju kodera. Sklop za preplitanje osigurava da se dvije permutacije istih ulaznih podataka kodiraju u dva različita paritetna niza. Učinak sklopa za preplitanje povezuje pogreške napravljene u jednoj polovici turbo kodera s pogreškama za koje postoji iznimno mala vjerojatnost da će se pojaviti u drugoj polovici. To osigurava robusna svojstva u slučaju da karakteristike kanala nisu poznate pa je to glavni razlog zašto su turbo kodovi bolji od tradicijskih kodova.

[120] *permutirati* … (lat. *permutare*) razmjenjivati, zamjenjivati, promijeniti; razmještati, premjestiti, premještati

Izbor sklopa za preplitanje ključan je za svojstva sustava turbo kodiranja. Svojstva turbo kodova mogu se analizirati u smislu Hammingove udaljenosti između kodnih riječi. Ako se dogodi prekid primijenjenoga ulaznoga niza, malo je vjerojatno da jednom prepleten niz neće proizvesti veliku Hammingovu udaljenost u barem jednome od dvaju kodera. Ako slučajan podatkovni niz ima malu ukupnu Hammingovu udaljenost, malen je broj takvih kodnih riječi, jer se isti niz negdje drugdje u bloku za unos podataka drugačije prepliće.

#### 7.3.2.2. 7.3.2.2. Podrezivanje ("kljaštrenje") kodova

Turbo kod na slici 7.13 ima relativno malu brzinu od 1/2.




*Slika 7.13: Promjena brzine koda na 1/2*

Umjesto promjene strukture kodera, brži kodovi obično se generiraju postupkom poznatim kao *podrezivanje*[121] (*puncturing*), vidi poglavlje 6.3.1.4. Probušeni kodovi. U turbo kodu na slici 6.10, promjena brzine koda na 1/2 može se postići "probijanjem" dvaju paritetnih bitova prije multipleksora. Na primjer po jedan bit može se izbrisati iz svakoga paritetnoga izlaza, tako da za svaki bit podataka ostaje jedan kontrolni bit. Slično, brisanjem drugih omjera paritetnih izlaza mogu se generirati i druge brzine koda. Izbrisani bitovi moraju se u dekoderu zamijeniti umetanjem "lažnih" (*dummy*) paritetnih bitova. U slučaju dekodera mekoga odlučivanja (za iteracijsko dekodiranje), to ne smije unazaditi dekodiranje.

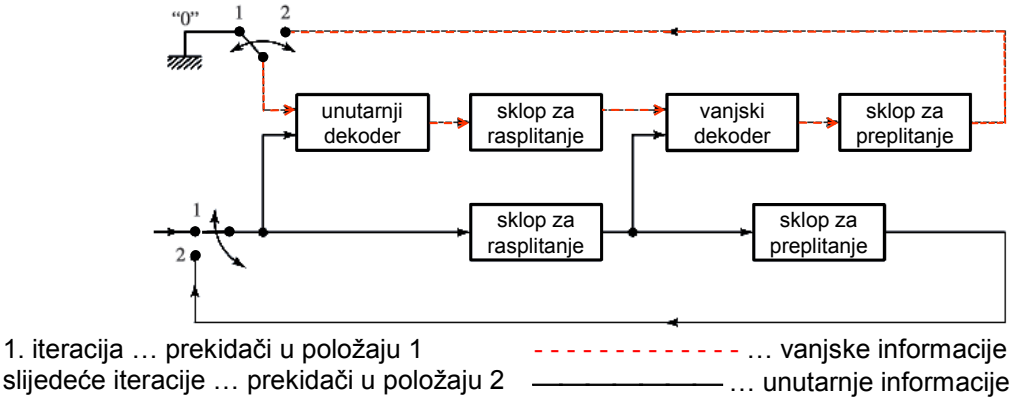
## 7.4. 7.4. Turbo dekodiranje

Ključna komponenta iteracijskoga (turbo) dekodiranja je dekoder mekim ulazom i mekim izlazom SISO (*soft-in, soft-out*). Svojstva veze demodulator-dekoder mogu se znatno poboljšati ako su dekoderu raspoložive meke odluke iz demodulatora, jer se na temelju njih donose konačne tvrde odluke. "Meka" odluka obično je zastupljena količinski *omjerom logaritamskih vjerojatnosti* LLR (*logarithm of the likelihood ratio*) za svaki bit. Ideja iteracijskoga dekodiranja je da ovu informaciju koristi niz dekodera. Imajte na umu da LLR odražava Shannonovu količinsku mjeru informacije.

Kodovi pôlja (*array codes*) naznačeni na početku ovoga poglavlja, koriste iteracijsko turbo načelo. Uobičajena tehnika dekodiranje za kodove pôlja je odvojeno dekodirati retke i stupce. Da ne bi došlo do velikih pogrešaka, najprije se dekodiraju stupci, a onda redci. Dekodiranje kodovima pôlja slično je rješavanju križaljke, jer se riječi moraju podudarati okomito i horizontalno.

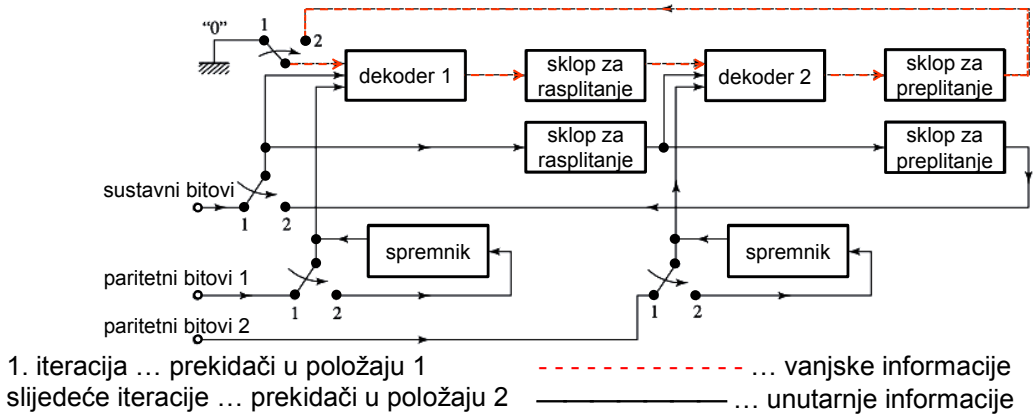
Optimalan dekoder pronašao bi najveću vjerojatnost za rješenje cijeloga polja, a to daje minimalan broj pogrešaka. Zato su se znatno povećali, složenost dekodera i kašnjenje. Dekodiranje ulančanih prepletenih kodova je otežano, jer samo jedan dekoder ima pristup do "mekih" podataka iz demodulatora. Rješenje je primijeniti "meke“ informacija u oba dekodera rasplitanjem ulaznoga signala, a zatim ovaj niz primijeniti na drugi dekoder. Poboljšanja se mogu postići tako da se "meki" podaci iz izlaza prvoga dekodera vrate na ulaz drugoga dekodera. Konačno poboljšanje je vratiti izlaz drugoga dekodera natrag na ulaz prvoga, koristeći iteracijski učinak. Slika 7.14 prikazuje iteracijski dekoder za kodove pôlja.

[121] Sinonimi su: *kljaštrenje*, *bušenje*




*Slika 7.14: Osnovni iteracijski dekoder.*

Raspoložive informacije o raznim odlukama dekodiranja dolaze iz nekoliko izvora. Neke informacije dolaze izravno od samih podatkovnih bitova. One se nazivaju *unutarnje informacije* (*intrinsic information*). Ako se dekodira *povezano* ili *kodom umnoška*, prolaz podataka iz jednoga dekodera u sljedeći naziva se vanjska informacija (*extrinsic information*). Samo vanjske informacije prenose se od dekodera do dekodera, jer su unutarnje informacije izravno dostupne. Nakon što se dekodiraju unutrašnji i vanjski kodovi, podaci se ponovno pohrane na odgovarajući način. Postupak se ponovi onoliko puta koliko je potrebno, dok se konačno ne donese tvrda odluka i ne dobije se dekodiran bit. Ovo je taj iteracijski postupak iz kojega je izveden naziv *turbo dekodiranje*. Slika 7.15 prikazuje osnovnu strukturu turbo dekodera.




*Slika 7.15: Blok-dijagram turbo dekodera.*

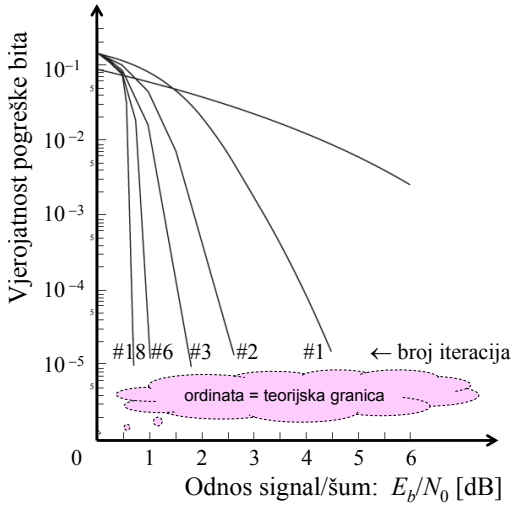
Svaki od dvaju stupnjeva dekodiranja koristi BCJR (Bahl, Cocke, Jelinek, Raviv) algoritam, osmišljen za rješenje problema otkrivanja maksimalne a posteriori vjerojatnosti MAP (*maximum a posteriori probability*). BCJR algoritam razlikuje se od Viterbijevoga algoritma dekodiranja u dva ključna motrišta.

1. To je algoritam s *jednostrukim ulazom i jednostrukim izlazom* SISO (*single input single output*) i dvije rekurzije, jedna prema naprijed, a druga natrag. Obje uključuju meke odluke. Viterbijev algoritam koristi algoritam dekodiranja *mekim ulazom* i *tvrdim izlazom*.

2. BCJR algoritam je MAP dekoder, jer smanjuje pogreške bitova procjenjujući a posteriori vjerojatnosti pojedinih bitova u kodnoj riječi. Za ponovno ustrojiti izvoran niz podataka, izlazi BCJR algoritam mekoga izlaza, veoma su ograničeni. Viterbijev algoritam procjenjuje niz maksimalne vjerojatnosti tako da maksimizira funkciju vjerojatnosti za cijeli niz, a ne za svaki bit. Za turbo dekodiranje razvijena je prikladna inačica Viterbijevoga algoritma. Ovaj algoritam naziva se Viterbijev algoritam s mekim izlazom SOVA (*soft output Viterbi algorithm*).

Uvjet praktične primjene turbo kodera je, da dekoderi moraju raditi puno brže od brzine kojom se primaju ulazni podaci tako da se može napraviti nekoliko iteracija. Dekoder s jednom iteracijom može se zamijeniti nizom međusobno povezanih dekodera ("cjevovod" dekodiranje), tako da je izlaz dostupan u konačnome stanju. U bilo kojemu slučaju prethodno postavljenih broja iteracija, kombinacije unutarnjih i vanjskih informacija, mogu se koristiti za pronaći dekodiran niz podataka. Tipično se koristi, čvrst broj iteracija (u rasponu od 4 do 10), ovisno o vrsti koda i njegovoj duljini.

Vjerojatnost pogreške bita za turbo kod brzine 1/2 uz $P_1(x) = 1 + x + x^2 + x^3 + x^4$ i $P_2(x) = 1 + x^4$ sklopom za preplitanje 256 × 256, prikazuje slika 7.16.



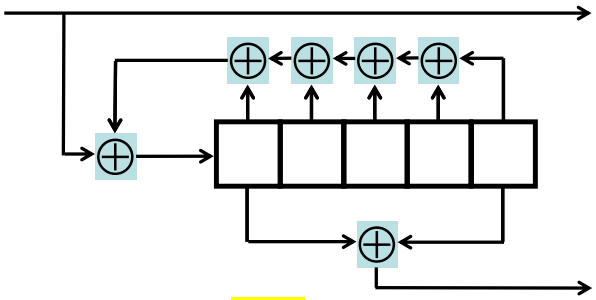

*Slika 7.16: Svojstva turbo koda kao funkcija broja iteracija (Izvor: Berrou i Glavieux, 1996).*

Može se vidjeti da nakon otprilike šest ponavljanja, nema velikoga poboljšanja. Za dekoder s $K = 3$ i 18 ponavljanja, BER = $10^{-5}$ postiže se za $E_b/N_0$ = 0,9 dB. Stavljajući ovo u kontekst slike 5.16.b. pokazuje se da uobičajen konvolucijski ($n = 47$, brzine 1/2) kod, zahtijeva $E_b/N_0$ = 4,0 dB za isti BER.

Napomenimo da se zadivljujuća dobit kodiranja postiže na štetu veličine sklopa za preplitanje. Osim toga, mogu postojati ograničenja kašnjenje/brzine zbog prirode dekodiranja. U 2011. godini, jedan od najbržih ASIC (*application specific integrated circuit*) dekoderskih uređaja postigao je propusnost od 1,28 Gb/s (uz maksimum od 6 iteracija). Turbo kod u osnovi, slučajno se pojavljuje u kanalu na temelju pseudo-slučajnoga sklopa za preplitanje. Ima fizički ostvarivu strukturu dekodiranja pa je njegov zadivljujući način rada, u središtu primjena.

### 7.5.1. 7.5.1. SVOJSTVA TURBO KODOVA (KODIRANJE I DEKODIRANJE)

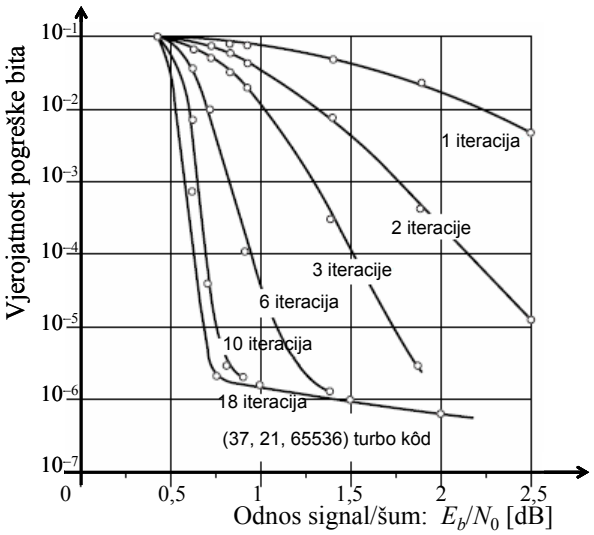
Turbo kodovi odlikuju se izvrsnim svojstvima pri malim vrijednostima *S/N*. Rad turbo kodova poboljšava se povećanjem duljine sklopa za preplitanje i brojem iteracija. Izvorni turbo-kod proučavali su Berrou i suradnici (1993) koristeći rekurzijski sustavan konvolucijski koder prikazan na slici 7.17.



*Slika 7.17: Rekurzijski 21/37 sustavan konvolucijski kod.*

Vidljivo je da u ovome kod(er)u, veze prema naprijed opisuje $\mathbf{g}_1$ = [10001], a povratne veze opisuje $\mathbf{g}_2$ = [11111]. Ovi vektori mogu se predstaviti u oktalnome obliku 21 odnosno 37, pa je uobičajeno da takav kod nazivamo 21/37 rekurzijski sustavan konvolucijski kod. Ovaj broj koristi se u turbo kodu s ograničenjem duljine $n$ = 65536, a "bušenje" se koristi za povećanje njegove brzine na 1/2.

Svojstva rezultirajućega koda korištenjem BCJR algoritma dekodiranja prikazuje slika 7.18.




*Slika 7.18: Prikaz* (37, 21, 65536) *turbo koda za različit broj iteracija.*

Nakon 18 iteracija, svojstva ovoga koda udaljena su samo 0,7 dB od Shannonove granice.

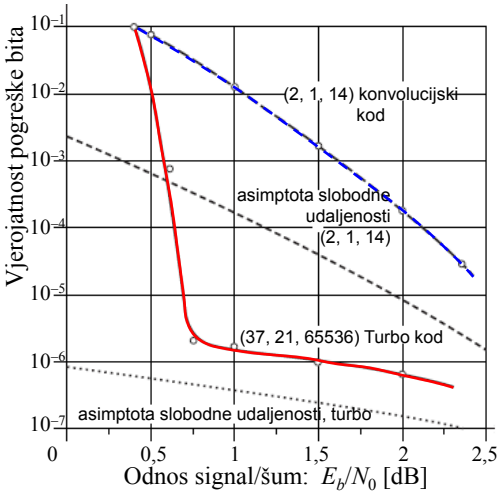
Jedan od problema s turbo kodovima je postojanje *donje razine pogreške* (*error floor*). Kao što se vidi na slici 7.18, vjerojatnost pogreške naglo se smanjuje povećanjem $E_b/N_0$ do određene točke. Nakon ove točke, vjerojatnost pogreške smanjuje se vrlo sporo. Postojanje donje razine pogreške posljedica je svojstava udaljenosti turbo kodova. Izgleda da, iako turbo kodovi imaju izvrsna svojstva, oni imaju prilično lošu minimalnu udaljenost. Razlog zbog kojega mogu dobro raditi je da, iako siromašnih svojstava udaljenosti, broj staza pri malim udaljenostima tzv. mnogostrukost te udaljenosti (*multiplicity of that distance*), $a_{d_{\min}}$, vrlo je mala.

U običnim konvolucijskim kodovima, možemo oblikovati kodove s puno boljim minimalnim udaljenostima, ali mnoštvo malih udaljenosti puno je veća. Ako sada promotrimo jednadžbu:

$$\overline{P}_b \approx \frac{1}{2k} a_{d_{\min}} f(d_{\min}) e^{-R_c d_{\min} E_b / N_0},$$

gdje su: $\overline{P}_b$, prosječan broj pogrešnih bitova za $k$ ulaznih bitova, $k$, duljina ulaznoga binarnoga bloka, $f(d)$, broj ne-nultih ulaznih bitova, a $R_c$, brzina vanjskoga koda. Vidimo da je za male $S/N$, učinak mnogostrukosti ($a_{d_{\min}}$) još utjecajniji na izvedbu koda. Za velike $S/N$ vrijednosti, minimalna udaljenost koda igra veliku ulogu. Pri velikim $S/N$ svojstva turbo kodova naglo se narušavaju.

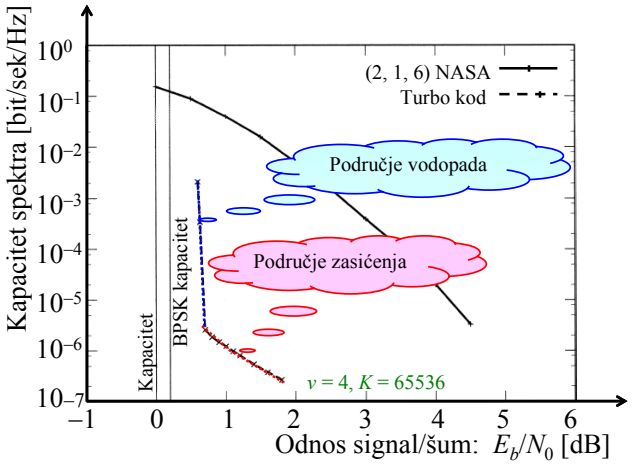
Slika 7.19 uspoređuje učinkovitost prethodno opisanoga 37/21 turbo koda brzine polovine konvolucijskoga koda i ograničene dužine 14 te Viterbijevoga algoritma dekodiranja mekom odlukom.




*Slika 7.19: Usporedba izvedbe turbo koda i konvolucijskoga koda.*

Na istome crtežu također su nacrtane granice svojstava (korištenjem zajedničkoga ograničenja) za oba koda.

Zasićenje pogrešaka (*error floor*) pojava je kojju susrećemo u modernim rijetkim ponavljajućim kodovima za ispravak pogrešaka. Ti kodovi temelje se na grafu, poput *linearnih kodova*, čije *paritetne matrice* sadrže veoma *malen broj* ne-nultih bitova (kodovi male gustoće s paritetnom provjerom) LDPC (*Low-Density Parity-Check Codes*)[122] i turbo kodova (slika 7.20).




*Slika 7.20: Područje zasićenja i područje vodopada turbo kodova*

Crtanjem omjera pogreške bitova BER (*bit error rate*) za uobičajene kodove poput Reed-Solomonovih uz algebarsko dekodiranje, ili konvolucijskih kodova uz Viterbijev algoritam dekodiranja, BER se stalno smanjuje u obliku krivulje sukladne poboljšanju *S/N* odnosa. Za LDPC i turbo kodove postoji točka nakon koje pad krivulje ulazi u zasićenje tj. područje u kojem se svojstva ustaljuju. Ovo područje zove se *područje zasićenja pogrešaka* (*error floor region*) Područje neposredno prije nagloga pada svojstava, zove se *područje slapa* (*waterfall*). U slučaju turbo kodova, zasićenje pogrešaka obično se pripisuje maloj težini kodnih riječi, a u slučaju LDPC kodova, pripisuje se obuhvatu skupova kodnih riječi ili bliskih kodnih riječi.

## 7.6. 7.6. Ulančano kodiranje

Shemu ulančanoga kodiranja prvi je predložio Forney[123] kao način za postizanje velikoga kodnoga dobitka. Sjedine se dva ili više relativno jednostavnih gradbenih blokova ili *komponentskih kodova* (ponekad nazvani *sastavni kodovi*). Rezultirajući kodovi imaju sposobnost ispravke pogreška mnogo dužih kodova. Sami kodovi obdareni su strukturom koja dopuštena relativno lako do umjereno složeno dekodiranje. Serijska veza kodera najčešće se koristi za sustave ograničene snage kao što su odašiljači na svemirskim letjelicama.

Najpopularnije od ovih shema sastoje se od Reed-Solomonova vanjskoga koda (prvi se primjenjuje, zadnji se uklanja) nakon čega slijedi konvolucijski unutrašnji kod (zadnji se primjenjuje, prvi se uklanja). Turbo kod možemo zamisliti kao usavršenu strukturu ulančanoga koda plus iteracijski algoritam za dekodiranje pridružen kodnome nizu. Zbog njegovoga jedinstvenoga iteracijskoga oblika, *turbo-kod* je na popisu kao zasebna kategorija strukturiranih nizova (tablice 0.1 i 0.2).

Opisane sheme turbo kodova, postižu vjerojatnost pogreške bita od $10^{-5}$. Koriste se kodovi brzine 1/2 preko Gaussova kanal s adicijskim bijelim šumom AWGN (*additive white Gaussian noise*) i BPSK modulacijom uz $E_b/N_0$ od 0,7 dB. Kodovi su ustrojeni korištenjem dvaju ili više komponentskih kodova uz različite inačice preplitanja istoga informacijskoga niza.

Za uobičajene kodove, konačan korak u dekoderu vodi tvrdome odlučivanju o dekodiranim bitovima (ili općenitije dekodiranim simbolima). Za ispravan rad spojenih shema kao što je shema za turbo kod, algoritam dekodiranja ne bi trebao propuštati tvrde odluke među dekodere. Za najbolje iskoristiti informacije naučene iz svakoga dekodera, algoritam dekodiranja na kraju zamjenjuje meko odlučivanje tvrdim. Koncept turbo dekodiranja je proslijediti meke odluke iz izlaza jednoga dekodera na ulaz drugoga te ponavljati ovaj postupak nekoliko puta tako da se proizvedu pouzdanije odluke.

[122] Ove kodove prvi je objavio Gallager u svome doktoratu, na MIT sveučilištu, 1962.
[123] Forney, G. D. Jr., *Concatenated Codes*, Cambridge, Massachusetts: M. I. T. Press. 1966.

# 18. *Laboratorijska vježba 17: Turbo kod i Viterbijev algoritam dekodiranja mekom odlukom*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

# 8. POGLAVLJE 8 - TRELLIS KODOVI

## 8.1. 8.1 Uvod

Što radi konvolucijski koder brzine 1/2? Prihvaća jedan bit, kodira ga i na izlazu generira dva bita. To je nešto što se radi na digitalnoj razini. Ova dva bita zatim se moduliraju (tj. pretvaraju u analogan oblik pomoću sinusnoga nositelja) te se prenose kanalom. Kodiranje, kao digitalna funkcija i modulacija, kao analogna funkcija, rade se neovisno za većinu modulacija. U modulaciji kodirane rešetke TCM (*trellis coded modulation*), ove dvije funkcije sjedinjuju se u jednu funkciju. Osnovni pojmovi neophodni za razumijevanje TCM opisani su na kraju skripte kao dodaci. To su: Euklidova i Hammingova udaljenost, koncept I i Q kanala te razmaci između nizova.

## 8.2. 8.2. Modulacija kodirane rešetke (TCM)

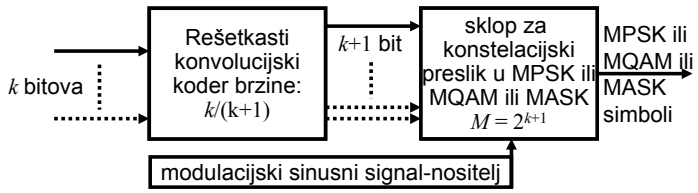
TCM koristi mnogo različitih koncepata iz obrade signala. U najjednostavnijemu smislu to je spajanje kodiranja i modulacije pa joj odatle ime *modulacija kodirane rešetke*. To znači da se za kodiranje koristi rešetka, a to je slično konvolucijskome kodiranju. Dok *kodiranje* i *modulaciju* normalno razmatramo kao dva samostalna motrišta komunikacijske veze, u TCM ona se združuju.

TCM modulacija, složen je pojam za razumijevanje, osobito zbog nelinearnoga načina rada. Ona koristi ideje iz modulacije i kodiranja, kao i dinamičko programiranje, ustroj rešetke i algebru matrica. Konvolucijski kod, koji ima optimalna svojstva ako se koristi samostalno, u TCM ne mora imati optimalna svojstva. Grayev kod koristan je za nekodiranu signalizaciju i konstelacijski preslik, ali nije uvijek tako u TCM. Proučavanje TCM nije lagana tema zahvaljujući Ungerboeck i drugima koji su je pronašli. Srećom, nema puno matematike kojom se treba ovdje baviti. No moraju se poznavati koncepti: konvolucijskih kodova, rešetke, rešetkastih struktura, koskupova i generatora koskupova.

Komunikacijska teorija kaže da je najbolje oblikovati kodove u duge nizove poruka. *Dopušteni* nizovi trebali bi međusobno biti vrlo različiti. Prijemnik onda donosi odluku između nizova koristeći svoje statistike, ali ne na temelju simbol-po-simbol. Ako se dekodira na ovaj način, vjerojatnost pogreške je inverzna funkcija *duljine niza*. U općemu obliku, vjerojatnost pogreške između nizova $p_e$, zadana je izrazom

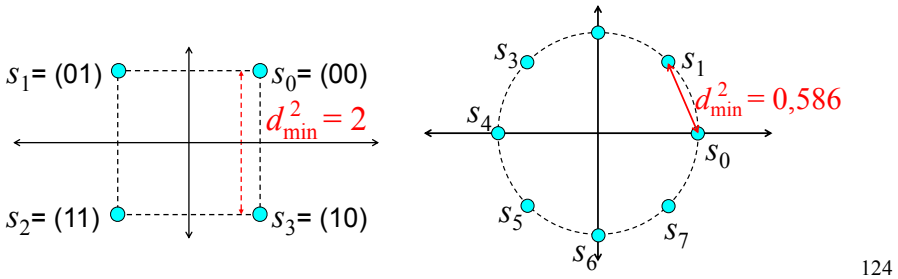
$$p_e \sim e^{-d_{\min}^2 / 2\sigma^2} \tag{8.1}$$

gdje je $d_{\min}$ Euklidova udaljenost između nizova, a $\sigma^2$ je snaga šuma. Svojstva TCM (i mnogih drugih shema) mjere se prema asimptotskome dobitku kodiranja ACG (*Asymptotic Coding Gain*). Ovo je dobitak ostvaren nekim osnovnim svojstvom visokih razina odnosa signal/šum u Gaussovome okruženju. AGC nije ostvariv u praksi, jer ne prenosimo signale na najvišim odnosima *S*/*N*, pošto postoje sklopovske i kanalske nepravilnosti koje odstupaju od pretpostavki dodatnoga bijeloga Gaussovoga šuma AWGN (*Additive White Gaussian Noise*). Dakle, prepoznaje se da su svi ovdje citirani dobici, maksimalno mogući samo u teoriji. Stvarni brojevi određuju se ispitivanjima i simulacijama u određenome okolišu. Funkcije usluge TCM sastoje od *kodirane rešetke* i *sklopa za konstelacijski preslik* kao što prikazuje slika 8.1.




*Slika 8.1: Opći prikaz modulacije kodirane rešetke*

TCM sjedinjuje funkcije konvolucijskoga kodera brzine $R = k/(k+1)$ i $M$-struki sklop za *preslik* signala čiji je zadatak preslikati $M = 2^k$ ulaznih točaka u veću konstelaciju od $M = 2^{k+1}$ konstelacijskih točaka. Za $k = 2$, imamo kod brzine 2/3 koji koristi QPSK signal ($M = 4$) i oblikuje 8-PSK signal ($M = 8$). Dakle, umjesto da se povećava propusnost, udvostručuje se broj konstelacijskih točaka. Broj točaka raste od B-PSK preko QPSK i 8-PSK do 16QAM i dalje. To je svojevrsna nadogradnja sustava, jer se uzima odabrani signal i nadograđuje ga na drugi s većim brojem konstelacijskih točaka, a prikazuje ga slika 8.2.



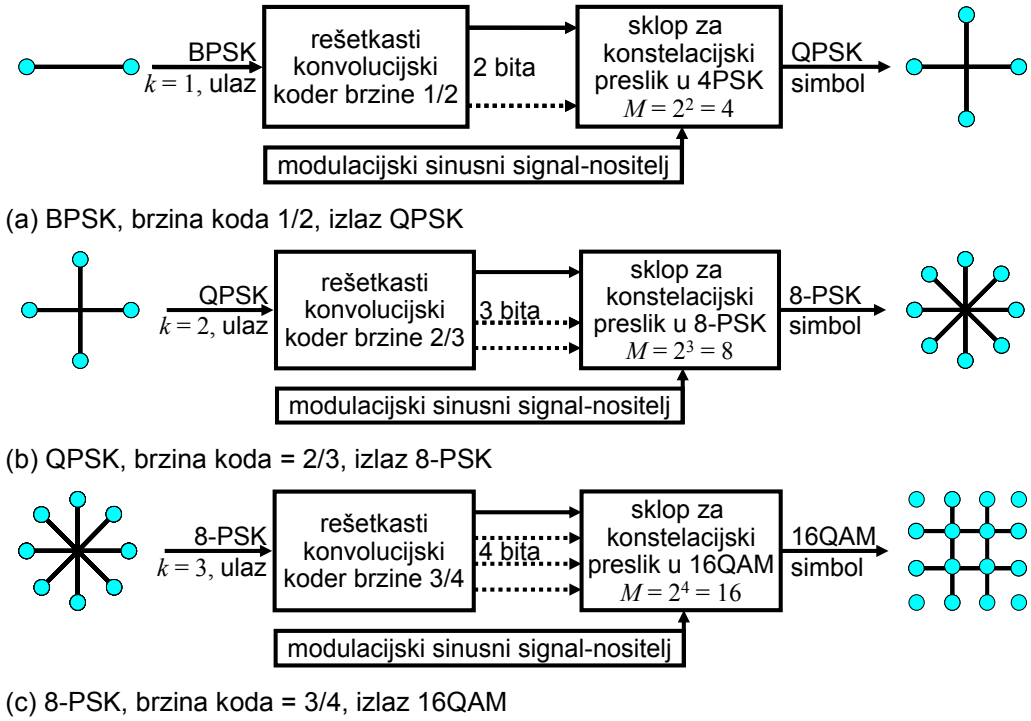

124

*Slika 8.2: Konstelacijsko udvostručenje u TCM. Prijenos QPSK signala pomoću 8-PSK konstelacije.*

Glavna svojstva TCM su:

1. TCM je modulacija koja konvolucijskim kodiranjem učinkovito koristi raspoloživu propusnost.
2. Ona štedi propusnost udvostručenjem broja konstelacijskih točaka signala. Na taj način brzina bitova povećava se, ali brzina simbola ostaje ista.
3. Konvolucijsko kodiranje ograničuje dozvoljene prijelaze simbola, kodiranjem niza (*sequence coding*).
4. Za razliku od pravih konvolucijskih kodiranja, ne kodiraju se svi dolazni bitovi.
5. Povećanjem konstelacije (broja konstelacijskih točaka), smanjuju se Euklidove udaljenosti između njih. Idući na višu konstelaciju, kodiranje niza nudi dobitak kodiranja koji nadvladava manjak snage.
6. Svojstva se mjere dobitkom kodiranja u odnosu na nekodiran signal.
7. Kao mjera dekodiranja koristi se Euklidova, a ne Hammingova udaljenost.
8. Ungerboeck je izvorno predložio TCM koristeći podjelu na skupove i manji broj stanja te kodne brzine koje se mijenjaju prema vrsti ulaznoga signala.
9. Pragmatička TCM koristi brzinu manju od savršene brzine 1/2 konvolucijskoga koda ograničenjem duljine jednake 7 ili 9. Ovo je široko dostupan kod pa njegovo korištenje te je TCM jeftinija za izvedbu.
10. Konstelacijski preslik u raščlanjivanje skupa (*set partitioning*), temelji se na prirodnome numeriranju gdje je Grayevo kodiranje poželjnije u pragmatičnoj TCM.

TCM je općenit koncept pa mijenjajući $k$, možemo stvoriti QPSK, 8-PSK ili modulacije viših razina kao što prikazuje slika 8.3.




*Slika 8.3: Opći prikaz modulacije kodirane rešetke*

[124] Vidi poučak o kosinusima: $d = \sqrt{b^2 + c^2 - 2bc \cdot \cos 45°}$

### 8.2.1. 8.2.1. PODVRSTE TCM

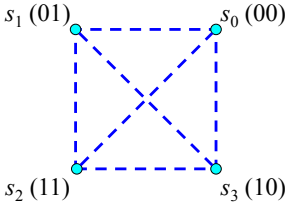
Imajte na umu da je u svakome slučaju, brzina koda drugačija. No, sve one zovu se TCM. Kodiranjem, broju bitova simbola, dodaje se samo jedan dodatan bit. Veličina simbola povećava se od $k$ na $k$+1 bitova. Ako kodiranje povećava brzinu prijenosa za jedan dodatan bit, onda trebamo udvostručiti broj konstelacijskih točaka za smjestiti ovaj bit kao što ćemo vidjeli u nastavku, gdje je $L$ izvoran broj bitova po simbolu.

Dakle, ako je BPSK izvoran signal, onda će TCM koder iznjedriti QPSK, QPSK će postati 8-PSK, a ako se izabere 8-PSK, onda će ona postati 16QAM signal. Kako se konstelacija širi, a energija signala ostaje ista, smanjuju se udaljenosti između simbola. To podrazumijeva pogoršanje izvedbe, a ne poboljšanje. Odakle onda dolazi poboljšanje?

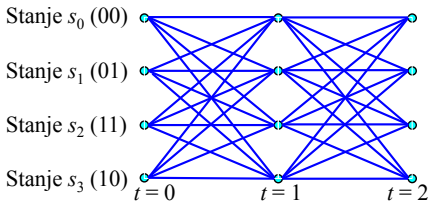
Započeli smo određenom propusnošću $B$, jer u stvarnome životu širina pojasa predstavlja veliko ograničenje. Iz ove širine pojasa, određuje se maksimalno moguća brzina simbola, koja nikada nije veća od $2B$, a obično je manja. Sada se određuje veličina abecede koja može osigurati potreban BER signala za zadanu raspoloživu snagu.

### 8.2.2. 8.2.2. NEKODIRAN QPSK SIGNAL (K = 2) PREDSTAVLJA KODIRANU TCM

Slika 8.4 prikazuje konstelaciju QPSK signala amplitude $d_{\min} = 1$ za prijenos QPSK signala.




(a) Četiri konstelacijske točke QPSK



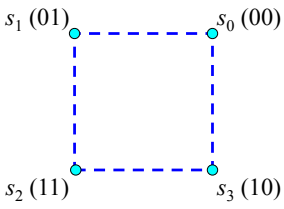

(b) Rešetka nekoga QPSK signala koji omogućuje sve prijelaze u svakome vremenskom razdoblju

*Slika 8.4: Konstelacija QPSK signala i njegova rešetka.*

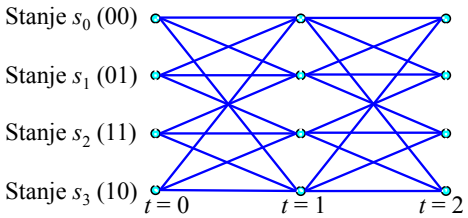
Minimalna Euklidova udaljenost između četiriju konstelacijskih točaka je $d_{\min} = \sqrt{2}$. Sada ćemo pogledati rešetku ovoga QPSK signala što je trivijalan[125] slučaj kao na slici 8.4.b.

Ova rešetka pokazuje kako se simboli prenose u nekodiranome QPSK signalu. Šesnaest staza prikazano je u dva vremenska razdoblja. Za nekodiran signal, moguća je svaka od tih 16 staza. U trenutku $t = 0$, možemo odabrati bilo koji od četiri moguća simbola, $s_0$ do $s_3$, a zatim u svakome narednome vremenskome taktu, opet možemo odabrati bilo koji od četiriju mogućih simbola. To je trivijalno, ali želimo proći kroz točku gdje ne postoje ograničenja u prijenosu nekodiranoga signala. Svi nizovi su dakle mogući, jer ovaj signal ne kodira niz.

Slika 8.5 prikazuje rešetku za raspoređenu (*staggered*[126]) QPSK, također se naziva i *pomaknuta* (*offset*) QPSK pa vidimo da postoje ograničenja u dopuštenim prijelazima. Iz svakoga stanja, možemo otići samo do dvaju susjednih simbola.




(a) Četiri konstelacijske točke S-QPSK



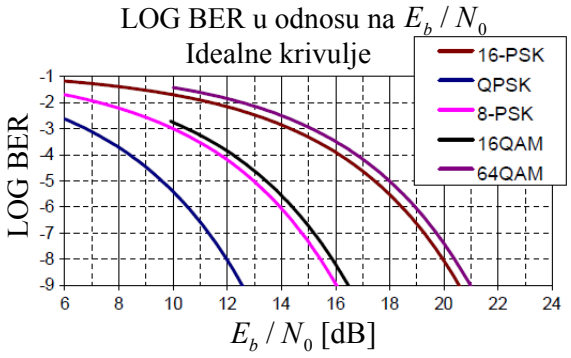

(b) Rešetka nekoga S-QPSK signala koja omogućuje samo tri prijelaza iz svake točke u svakome vremenskom razdoblju

*Slika 8.5: Konstelacija S-QPSK signala i njegova rešetka.*

[125] *trivijalan* (lat. *trivialis*) opće pristupačan, opće poznat, običan, *trivium* … raskrižje od tri puta, koji se može naći i na ulici, tj. običan, prostački, svakidašnji, suviše poznat, otrcan

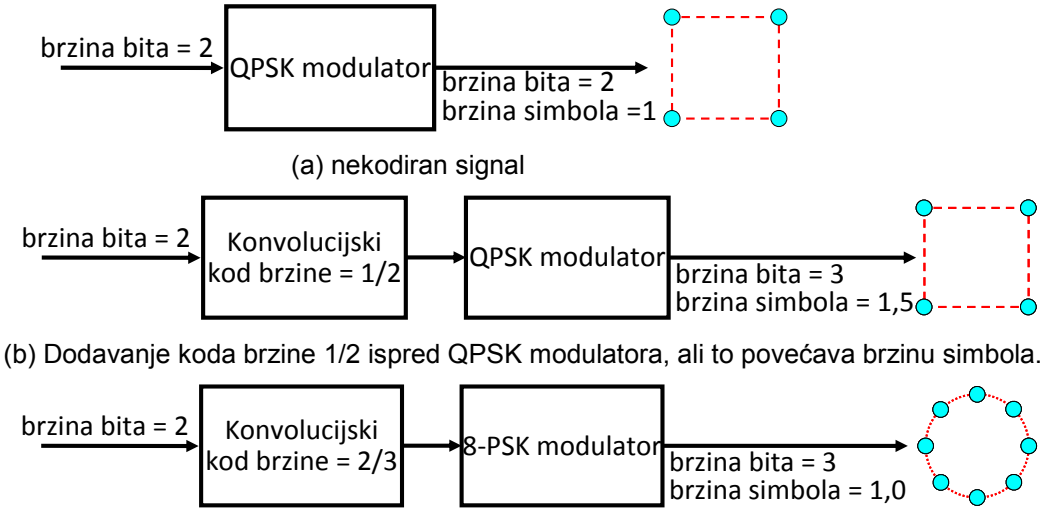
[126] *staggered* … raspoređen u šahovskome poretku (cik-cak)

Minimalna kvadrirana Euklidova udaljenost MSED (*Minimum Squared Euclidean Distance*), najmanja je kvadrirana udaljenost između svih točaka konstelacije i obično je to udaljenost između dviju susjednih točaka. Na prijemnoj strani, dekoder donosi odluku o tomu koji simbol se poslao na temelju toga u koje područja odluke (u)pada signal, poput slagalice[127]. Učinak pogreške, funkcija je MSED, $d^2_{\min}$, koji za signale na slikama 8.4 i 8.5, iznosi 1,414. Pretpostavimo da nam je želja prenijeti nekodiran QPSK signal uz BER od $10^{-5}$ (slika 8.6).




*Slika 8.6: Idealan BER u odnosu na $E_b/N_0$ u AWGN okruženju*

To će zahtijevati snagu od 9,6 [dB] za idealan $E_b/N_0$ u odnosu na BER. Ako toliko snage nije raspoloživo, npr. zato što je odašiljač malen, onda je izbor dodati kod brzine 2/3 za smanjiti BER, a on će omogućiti ovoliki BER i na manjoj vrijednosti $E_b/N_0$. Ali onda imamo još jedan problem. Ako zadržimo istu brzinu informacijskih bitova i dopustimo povećanje brzine kodnih bitova za smještaj prekobrojnih bitova, onda se zahtjev za propusnošću povećava faktorom $1/R$. Dakle, dodatnim kodiranjem, povećava se propusnost za 3/2. Ako se ne može promijeniti propusnosti, onda će se u istome omjeru morati smanjiti brzina informacije (slika 8.7).

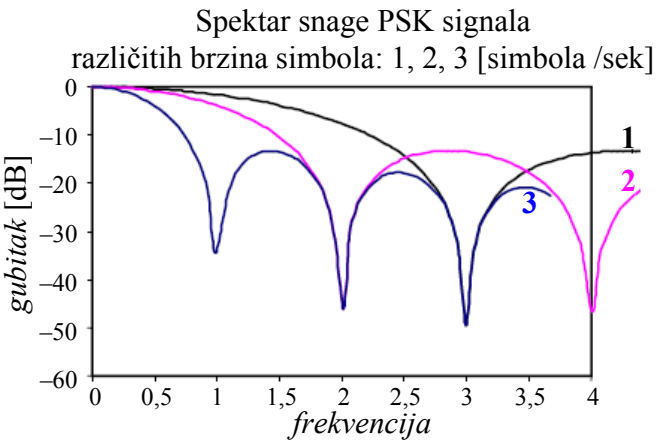



*Slika 8.7: Dodavanje nezavisnoga konvolucijskoga kodiranja QPSK signalu, povećava njegovu brzinu bitova i zahtijevanu propusnost brzine koda*

#### *8.2.2.1. 8.2.2.1. Primjer QPSK*

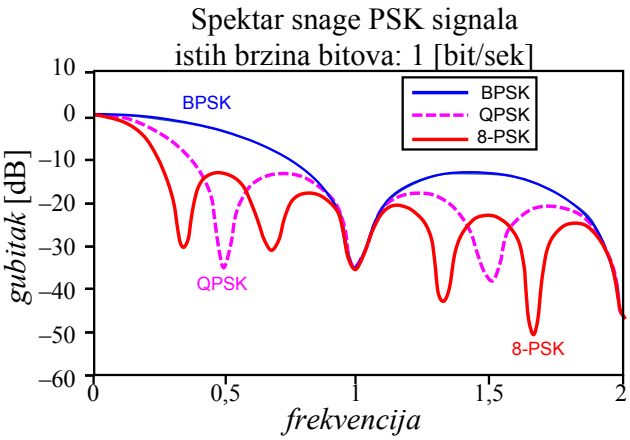
Prije kodiranja, brzina informacijskih bitova je 2 [bit/sek], a nakon dodavanja jednoga bita za kodiranje, brzina postaje 3 [bit/sek]. Brzina simbola koja je prije kodiranja bila 1 [simbol/sek] (*symbol per second*), poslije kodiranja je 1,5 [simbol/sek]. Što se događa s propusnošću ako se brzina simbola poveća od 1 na 1,5? Povećanjem brzine simbola, također se poveća propusnost (mjereno frekvencijom prve nule). Signal brzine simbola od 1 [simbol/sek] ima nisko-propusnu širinu od 1 [Hz], signal brzine simbola od 2 [simbol/sek], ima propusnost od 2 [Hz] i tako dalje. Propusnost signala povećava se od 1 [Hz] za nekodiran signal do 1,5 [Hz] za kodiran signal (slika 8.8).

[127] Dartboard, dart-board … pikado ploča, slagalica




*Slika 8.8: Protok u odnosu na potrebnu propusnost PSK signala.*

Za PSK, dok god je brzina simbola ista, sve M-PSK modulacije imaju istu propusnost, bez obzira na *M*. Ponovo pogledajmo slikuu 8.9 gdje se zahtijeva propusnost kao funkcija brzine *bitova* umjesto brzine *simbola*.




*Slika 8.9: Za istu brzinu bitova (za* M-PSK *signal), potrebna propusnost smanjuje se kako M raste.*

Sva tri signala imaju istu brzinu bita, ali 8-PSK zahtijeva najmanju propusnost. Zahtjevi za zadanu propusnost brzine bitova padaju kako *M* raste u M-PSK signalu, a to je ono što se misli pod nazivom *učinkovite modulacije bitova* (*bit-efficient modulations*).

U trećemu izboru (slika 8.7.c), pomičemo se prema gore u modulacijskome redoslijedu i na taj način možemo prenijeti signal koji je kodiran, ali bez izmjene propusnost ili spektra. Sada je problem što su 8-PSK simboli međusobno bliži. Gledajući krivulju na slici 8.6, vidi se da treba osigurati približno 3,5 [dB] veću snagu za isti BER. Budući da je to jedini način ostvarenja ove ideje (inače izborom nekodirane QPSK nema dobitka), pitamo se može li nam kod brzine 2/3 priskrbiti dobitak kodiranja od najmanje 3,5 [dB]?

### 8.2.3. 8.2.3. DOBITAK KODIRANJA KODIRANOGA SIGNALA U ODNOSU NA NEKODIRAN SIGNAL

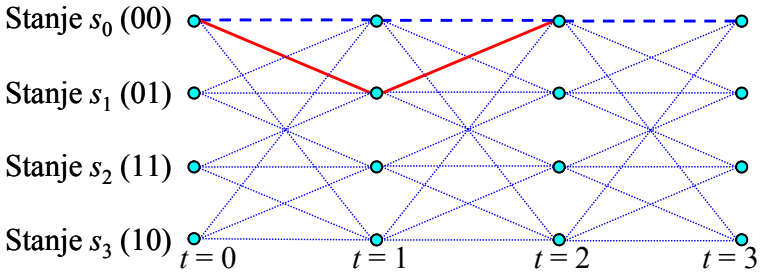
Dobitak kodiranja kodiranoga signala je

$$\gamma = \frac{d^2_{free/\,kodiran}}{d_{free/\,nekodiran}} \frac{E_{s/\,kodiran}}{E_{s/\,nekodiran}} \tag{8.1}$$

gdje je $d_{free/kodiran}$ jednaka Euklidovoj udaljenosti kodiranoga niza, a $d_{\min/nekodiran}$ je minimalna udaljenost između signala u nekodiranoj konstelaciji kao što se prethodno definiralo. Ovaj dobitak kodiranja upućuje na osnovni signal. Započeli smo QPSK i želimo BER od $10^{-5}$. Kazali smo da to zahtijeva $E_b/N_0$ od 9,6 dB. Broj do kojega smo došli za dobitak kodiranja iz jednadžbe 8.1, računa se kao smanjenje početne vrijednosti $E_b/N_0$. Činjenica da 8-PSK zapravo zahtijeva veći $E_b/N_0$ nego QPSK, osnovica za izračun je u jednadžbi 8.1. Dakle i pored toga što je $d_{\min/kodiran} < d_{\min/nekodiran}$, ako možemo priskrbiti da je $d_{free/kodiran} > d_{\min/nekodiran}$ ostvarit ćemo pozitivan dobitak. Znamo što je $d_{\min}$, ali što je $d_{free}$?

Kako izmjeriti Euklidovu udaljenost kodiranoga signala, ako je Euklidova udaljenost njegove konstelacije manja od $d_{\min}$. Sada moramo raspraviti *udaljenost između nizova* $d_{free}$, a ne udaljenost između *signala*. Mjera $d_{free}$ definira se kao Euklidova udaljenost kodiranoga signala u smislu najmanjega mogućega razmaka između svih dozvoljenih nizova. Umjesto provjere svih mogućih sjedinjenja (kombinacija), ova udaljenost mjeri se u odnosu na niz "sve 0". Isto tako mjere se Hammingova udaljenost i Hammingova težina, u odnosu na kodnu riječ "sve 0".

Pretpostavimo prijenos poruke "sve 0". Tijekom dekodiranja, dolazi do pogrešaka i dekoder slijedi krive putove kroz rešetku. Jedini način na koji dekoder može krenuti i onda se vratiti natrag na pravi put, a to je put "sve 0", jest udaljavanje pa približavanje kao što se prikazuje u nastavku na slici 8.10.




*Slika 8.10: Udaljavanje pa približavanje nizova (alternate). Uredno dekodirani nizovi prate gornju crtkanu (plavu) crtu. Najkraći, krivo dekodirani nizovi su 2 odsječka puta (puna crvena crta).*

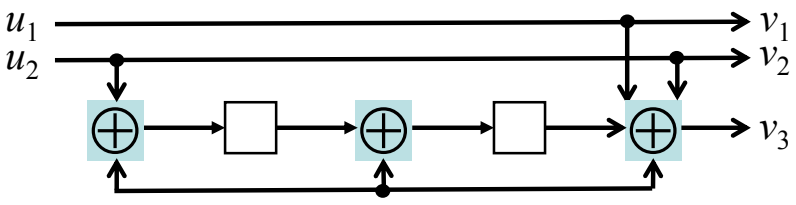
Euklidova udaljenost kodiranoga signala definira se kao najmanja udaljenost između niza "sve 0" i onoga koji se odmakne od njega, a zatim se raziđu. Što to točno znači?

Imajte na umu da je svaki niz, skup demoduliranih simbola. Ako se dvije staze razilaze, to znači da je došlo do pogreške i dekoder je donio pogrešnu odluku. Uz pretpostavku da je vjerojatnost takve pogreška mala, na slijedećim raskrižjima, za daljnje odluke treba se vratiti natrag na ispravan put. Malen dozvoljen put koji je netočan, vjerojatniji je od dugoga puta, tako da je udaljenost toga maloga puta mjera sposobnosti ispravke pogrešaka koda. To je isti koncept kao kada je za prijemnik vjerojatnije da će prihvatiti susjedne signalne točke. One predstavljaju najvjerojatniji ispravan signal, a ne odabir onih točaka koje su udaljenije pa takva temeljna ideja stoji iza *maksimalne vjerojatnosti dekodiranja* MLD (*Maximum Likelihood Decoding*).

U prijašnjemu primjeru jednostavnih rešetki vidimo da, ako se pogreška napravi u trenutku $t = 0$, a putovi se razilaze, onda se možemo vratiti na ispravan put u samo dva segmenta kao što se prikazalo. Zbroj kvadrata Euklidove udaljenosti za ovaj put, zove se *slobodna udaljenosti*, $d_{free}$ koda.

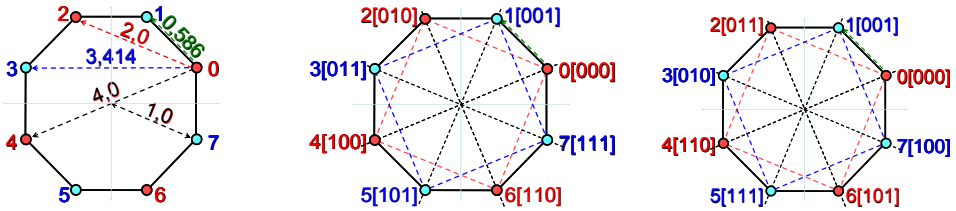
Euklidova udaljenost ovdje se određuje slijeđenjem puta razilaska i minimalne udaljenosti od puta "sve 0", simbol po simbol. Za prethodan primjer, to je zbroj SED (*Squared Euclidian Distance*) svakoga od dvaju segmenata.

Izračunajmo $d_{free}$ i dobitak kodiranja konvolucijskoga koda brzine 2/3 na slici 8.11, za vidjeti što on može učiniti.




*Slika 8.11: Konvolucijski koder brzine 2/3*

Ovo je konvolucijski koder s četiri stanja. Promatrajući shemu na slici 8.11, možemo nacrtati njegove funkcije prijenosa (slika 8.12).




(a) Preslik 8-PSK simbola (b) Prirodan preslik (c) Grayev kod

*Slika 8.12: Funkcije prijenosa za koder na slici 8.11*

To je sustavan koder, jer dva nekodirana bita prolaze, a pridružuje im se paritetan bit. Pogledat ćemo svojstva ovoga koda na temelju preslika dvaju različitih bitova u simbol. Na slici 8.12.b, simboli se preslikavaju u prirodnome binarnome redoslijedu, a na slici 8.12.c, Grayevim kodom preslikavaju se tako da se susjedne riječi razlikuju samo u jednome bitu. Imajte na umu da, iako je $d^2_{\min}$ za QPSK signal bio 2,0, isti broj je 0,586 što je skoro 75% manje za 8-PSK.

Zapišimo podatke rešetke konvolucijskoga 2/3 koda, započevši nulama u memorijama registara, a onda propustimo sva tri sastava bitova kroz registre (slika 8.13).

| stanja | bitovi | rešetka | SED | rešetka | |
|---|---|---|---|---|---|
| stanje 00 | 000<br>110<br>011<br>101 | | 0,0<br>2,0<br>3,414<br>3,414 | | |
| stanje 10 | 011<br>101<br>000<br>110 | | 3,414<br>3,414<br>0,0<br>2,0 | | |
| Stanje 01 | 100<br>010<br>111<br>001 | | 4,0<br>2,0<br>0,586<br>0,586 | | |
| stanje 11 | 111<br>001<br>100<br>010 | | 0,586<br>0,586<br>4,0<br>2,0 | | |
| | (a) Rešetka s pridruživanjem bitova | | (b) svaka putanja zamjenjuje se svojim SED | | (c) 3 odsječka koji se razilaze i onda spajaju u stanje 00 |



*Slika 8.13: Rešetka konvolucijskoga koda brzine 2/3*

Na slici 8.13.a, rešetka pokazuje četiri moguća stanja registara gdje se svaka crta odvaja u sastave bitova redom od vrha prema dnu. Na slici 8.13.b, možemo zamijeniti simbole kvadrata Euklidove udaljenosti od referentnoga simbola "sve 0". Te udaljenosti napisane su s lijeve strane, a ne na crtama da bi lakše pratili dijagram. U svakome stanju postoje četiri različita razlaza putova.

### *8.2.3.2. 8.2.3.2. Dobitak kodiranja*

Da bi se utvrdio dobitak kodiranja, moramo znati $d_{free}$ ovoga koda. Za određivanje puta minimalne udaljenosti, iz svakoga stanja slijedimo put s najmanjom kvadriranom udaljenošću (ali ne i 0). To je 2,0 za put koji započinje u stanju 00 ($s_0$). Ovo nas vodi u stanje $s_2$. Odavde, opet slijedimo put minimalne udaljenosti, što je 2,0. Ovo nas vodi u stanje $s_4$ i odavde se vraćamo u stanje $s_0$ preko puta koja ima kvadriranu udaljenost od 0,586. Nema drugoga put koji nas može odvesti natrag u stanje 00 ($s_0$), a da ima manju udaljenost.

Ukupna veličina minimalnih kvadriranih Euklidovih udaljenosti MSED za ovaj niz je zbroj sve tri navedene kvadrirane udaljenosti: 2 + 2 + 0.586 = 4.586.

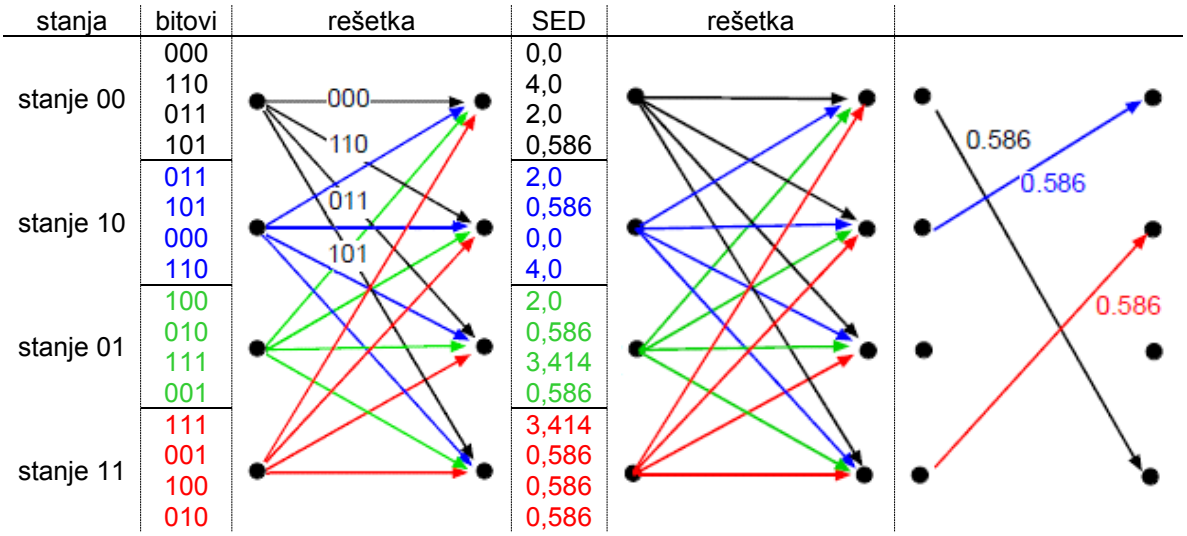
Da bi odredili dobitak kodiranja, podijelimo ovu udaljenost minimalnom udaljenošću nekodirane QPSK konstelacije, $d^2_{\min}$ = 2,0. Dobitak kodiranja iz jednadžbe 8.1, uz pretpostavku da kodiran i nekodiran signal imaju iste energije, je

$$10 \log\left(\frac{4.56}{2}\right) = 3.6 \ [\text{dB}].$$

Ovo je već bolje. Pokušajmo zamjenski preslik simbola (Grayev kod) kao što prikazuje slika 8.12.c. On daje rešetku na slici 8.13.a. Kao i prije, možemo pretvoriti svaki simbol na udaljenosti od 0-simbola, a zatim ćemo slijediti minimalan put. Ovaj put na našu žalost, ukupno možemo dobiti MSED od samo 0,586 + 0,586 + 0,586 = 1,758, ili zapravo gubitak od oko 0,5 dB. Dakle, vidimo da preslik predstavlja važno razmatranje. Grayev kod ovdje nije bio koristan. Dodavanjem bilo kojega koda brzine 2/3 ispred 8-PSK modulatora nije korisno. Sada vidimo kako nam TCM može pomoći u poboljšanju.

### *8.2.3.3. 8.2.3.3. Raščlamba skupa ili priskrba velike Euklidove udaljenosti*

Ungerboeck je želio poboljšati preslik ovoga koda u signale na poseban način nazvan *raščlamba skupa* (*set partitioning*). Osnovna ideja je da se $k$ informacijskih bitova preslika u $2^{k+1}$ konstelacijskih točaka, tako da možemo ograničiti događanje prijelaza samo uz najveći SED. Nakon što se podijele podskupovi, signali se upućuju dalje, povećavajući Euklidovu udaljenost između signala u tomu skupu. Slika 8.14 prikazuje ovaj postupak.

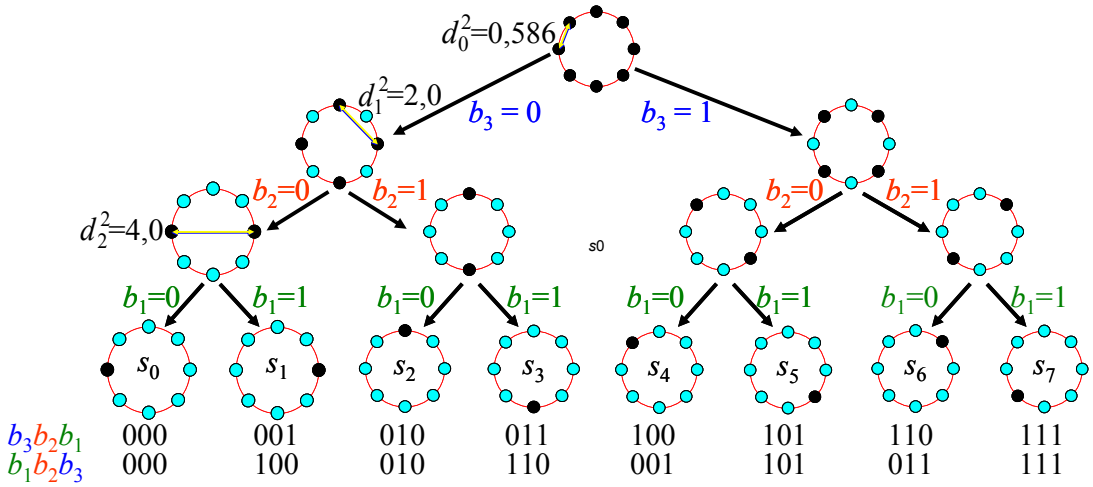



*Slika 8.14: Ako se promijeni preslik simbola u bitove, onda se mijenja i Euklidova udaljenost. (Ungerboeckova "moždana oluja")*

Konstelacija signala za 8-PSK podijeljena je (koristeći strukturu rešetke i pripadan izričaj). Imamo tri dolazna kodna bita. Oni su

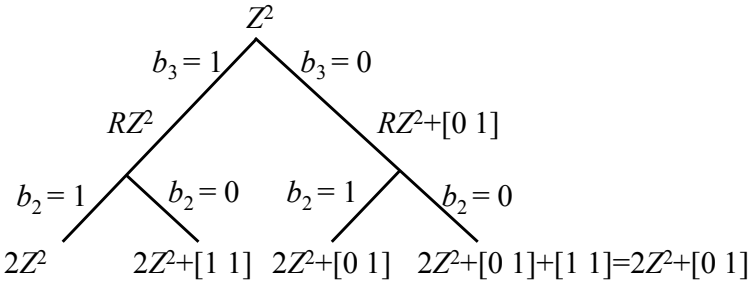
$$b_1\ b_2\ b_3$$

Počevši bitom $b_3$ pa koristeći ovaj bit, nastavimo niz stablo odluke kao što prikazuje slika 8.15.



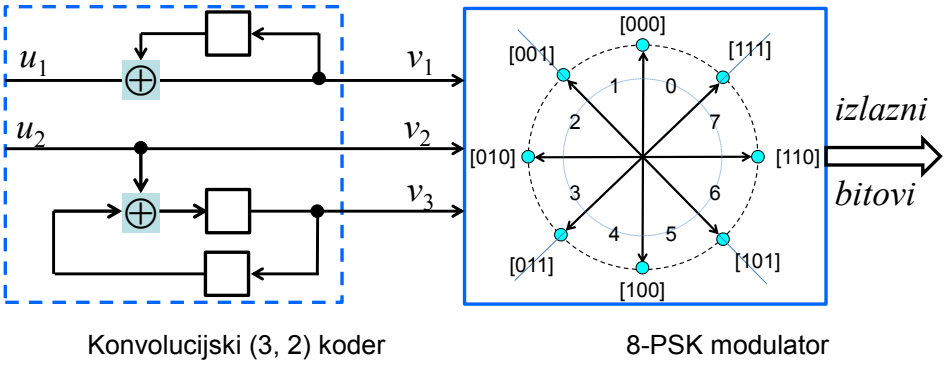

*Slika 8.15: Podjela 8-PSK konstelacije na sve veće podskupove Euklidove udaljenost.*

Ova slika pokazuje kako se osam točaka 8-PSK uzastopno raščlanjuje u disjunktne[128] koskupove tako da SED raste na svakoj razini. Postoje ukupno četiri raščlambe računajući prvi ne-podijeljen skup. Na najvišoj razini, MSED je 0,586. Na sljedećoj razini, gdje postoje samo četiri točke u svakome od dvaju koskupova, MSED se povećava na 2,0, a na posljednjoj razini, MSED je 4,0. Svaki podskup također se zove *koskup* pa izričajem rešetke, njezinim generatorima koskupova na ovaj način možemo prikazati raščlambu (slika 8.16).




*Slika 8.16: Struktura rešetke i koskup generatori za 8-PSK raščlambe skupova (3 najviše razine)*

Budući da prve dvije razine imaju manje udaljenosti, vjerojatnije su ove pogreške pa ćemo koristiti kodirane bitove za proći kroz ovaj dio. Bitovi $b_3$ i $b_2$ koji su kodirani, mogu se koristiti za odluku koju raščlambu (ili koskup) odabrati. Onda se na posljednjoj razini može koristiti nekodiran bit za izabrati prenesen signal. Najznačajniji bit $b_1$, koji se koristi na posljednjoj razini, ima veliku Euklidovu udaljenost svoga komplementa i zahtijevat će maksimalnu pogrešku od 180 stupnjeva da bi se oštetio.

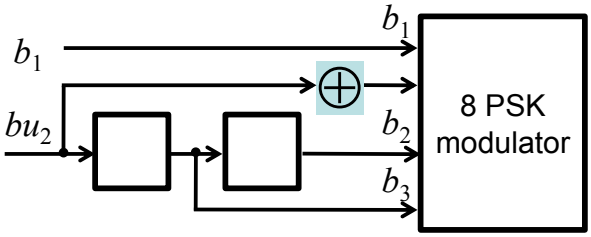



*Slika 8.17: Ungerboeckov LFSR koder s 3 stanja i 8-PSK modulator*[129]

### 8.2.3.4. 8.2.3.4. Primjer

Pomoću ovoga preslika, bitovi [010] preslikat će se u simbol $s_2$.

Ungerboeckov pristup ostavlja nekodiran najznačajniji bit i omogućuje mu da se brine o sebi preko svoje velike Euklidove udaljenosti. Ovo je ključno za veće dobitke kodiranja ovoga pristupa. Samo bitovi o kojima se odluči na najvišim razinama s manjim SED, kodiraju se, čime se smanjuje brzina kodiranja i povećava učinkovitost bita. Dakle, evo kako se može koristi TCM manje brzine koda, ostavljajući MSB nekodiran (slika 8.18).



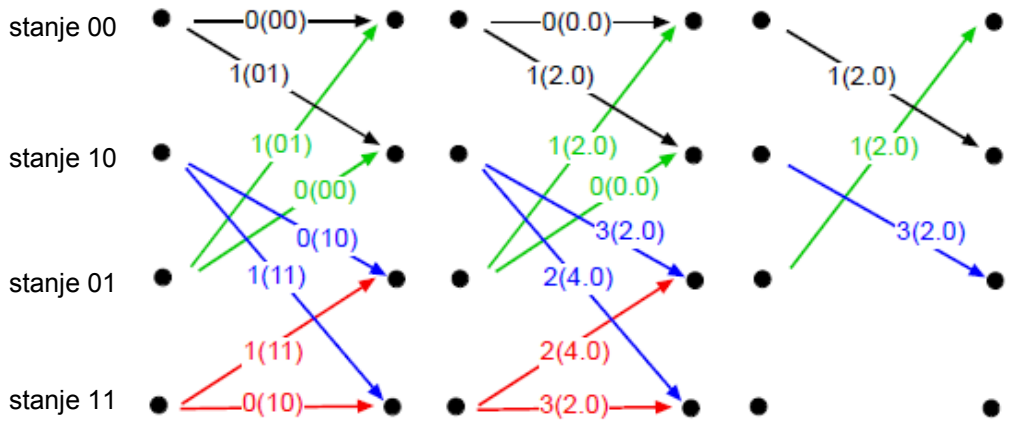

*Slika 8.18: Konvolucijski kod brzine 1/2 s 4 stanja kao osnova TCM*

[128] *disjunktan* … (lat. *disjuncitvus*) razdvojen, odvojen, isključen, rastavan, koji razdvaja, odvaja, isključuje; suprotan;
[129] Koder napraviti u Logisim™ 2.7.1

Sada umjesto korištenja brzine koda 2/3 na oba dolazna bita, možemo ostaviti jedan bit nekodiran, a zatim koristiti brzinu koda 1/2 na drugome bitu. Tri odlazna bita daju istu brzinu koda od 2/3. Međutim, drugi pristup više obećava, budući da brzina koda 1/2 ima bolji dobitak kodiranja.

Ovo je temelj TCM, selektivno kodirati samo neke bitove i iskoristiti povećanje Euklidove udaljenosti dobivene podjelom u podskupove, a onda ostaviti nekodirane bitove koji su prirodno zaštićeni svojom velikom SED.

Započinjemo koderom s četiri stanja (slika 8.19).




*Slika 8.19: Kodirana rešetka brzine 1/2 (jedan bit ulazi, 2 bita izlaze)*

Prvo nacrtamo rešetku konvolucijskoga kodera brzine 1/2. Kao što smo učinili za kod brzine 2/3. Ova rešetka pretvara se u kvadrirane udaljenosti, a zatim se prepoznaje putanja minimalne dužine. Staza minimalne dužine označena je kao [01 10 01]. Odgovarajuće udaljenosti od 00 simbola bita su 2,0, 2,0, 2,0, a njihov zbroj je 6,0 pa je MSED za ovaj kod jednak 6. Dobitak kodiranja je

$$10\log\left(\frac{6}{2}\right) = 4{,}77\ [\text{dB}]$$

To je bolje od 3,6 dB dobitka što smo ga imali za kod brzine 2/3, tako da ovo izgleda obećavajuće. Ali ne tako brzo. Ovo je samo dobitak kodiranja koda brzine 1/2. Što je s nekodiranim bitom, jesmo li (na) njega zaboravili? Ovo se obrađuje u laboratorijskoj vježbi 18.

## 8.3. 8.3. Korišteni pojmovi

*Blok-kodovi*: Kodovi koji od informacijskoga bloka bitova duljine $k$, proizvode kodnu riječ duljine $n > k$ ($n$ … duljina bloka ili duljina koda).

*Brzina koda*: Odnos broja informacijskih bitova $k$ u bloku u odnosu na ukupan broj prenesenih bitova $n$ u kodnoj riječi ($n = k$ + broj zalihosnih bitova).

*Ciklički kodovi*: Kodovi definirani preko polinom-generatora, koji se množe podatkovnim polinomom za dobiti polinom kodne riječi. Polinomi su definirani konačnim poljem, a krugovi posmičnih registara koriste se za množenja i dijeljenja polinoma. Primjeri su: Golayevi kodovi, BCH i Reed-Solomonovi kodovi.

*Dekodiranje najvećom vjerojatnošću*: Postupak dekodiranja koji maksimizira vjerojatnost primljenoga niza izborom bilo koje od mogućih kodnih riječi. Ako su kodne riječi jednako vjerojatne, ova metoda dekodiranja daje najmanju moguću vjerojatnost pogreške. U praksi, kao mjera "bliskosti" (*najveća vjerojatnost*) uzima se minimalna Hammingova udaljenost.

*Dijagram rešetke*: Graf koji se koristi za predstaviti vremensko ponašanje konačnoga automata, definiranjem stanja kao čvorova i mogućih prijelaza iz stanja kao grana.

*Galois polje*: Galois polje (ili konačno polje) skup je objekata ili elemenata na kojima su definirane dvije operacije + i ·, poštujući neke specifične propise (tj., operacije su komutacijske i distribucijske). Primjer: Ako je $p$ prost broj, skup cijelih brojeva $\{0, 1, \ldots, p-1\}$ mod $p$, oblikuje konačno polje.

*Generator-matrica*: Matrica $k \times n$ koja proizvodi kodnu riječ (duljine $n$) blok-kodova, množenjem informacijskoga bloka (duljine $k$).

*Hammingova udaljenost*: Broj mjesta na kojima se razlikuju dva niza jednake duljine. Minimalna udaljenost koda najmanja je udaljenost između bilo kojega Hammingovoga para kodnih riječi.

*Ispravak praskovitih pogrešaka*: Metoda ispravke pogrešaka koja je učinkovita za pojavu pogrešaka u skupinama, za razliku od slučajnih pogrešaka.

*Kapacitet kanala*: Maksimalna brzina prijenosa kojom se informacije mogu prenijeti preko kanala, s po volji malom frekvencijom pojave pogrešnih bitova.

*Kodna riječ*: Niz bitova dobivenih dodavanjem suvišnih bitova izvornoj poruci.

*Konvolucijski kodovi*: Kodovi koji se generiraju linearnim posmikom u krugu posmičnoga registra, a koji obavlja operaciju *konvolucije* nad informacijskim nizom.[130]

*Linearni kodovi*: To su kodovi čija linearna kombinacija bilo kojega skupa kodnih riječi, također daje kodnu riječ.

*Modulacija kodirane rešetke*: Metoda sastavljena od zajedničkoga *kodiranja* i *modulacije*, a temelji se na oblikovanju konvolucijskih kodova prilagođenih modulacijskim signalima. Oni su postavljeni tako da se poveća Euklidova udaljenost među signalizacijskim točkama moduliranoga niza.

*Otkrivanje pogreške*: Mehanizam kojime se otkriva prisutnost pogrešaka u kodnoj riječi. Uzorci pogrešaka koje se ne mogu otkriti, zovu se *neotkrivene pogreške*.

*Preplitanje*: Uređaj koji kodira simbole iz nekoliko kodnih riječi, tako da su bitovi pojedine kodne riječi međusobno dobro razdvojeni.

*Razdioba težine*: Popis Hammingovih udaljenosti svake kodne riječi od određene referentne kodne riječi.

*Serijsko dekodiranje*: Postupak za sustavno pretraživanje kroz kodna stabla s ciljem približnoga rješenja puta minimalne udaljenosti primljenoga niza. Primjeri: Fano algoritam, *algoritam stoga* (*stack algorithm*).

*Sindrom*: Vektor što ga računa dekoder za pronaći mjesto pogreške koju se može ispraviti.

*Sustavni kodovi*: Kodovi čiji je prvih (ili zadnjih) $k$ simbola svake kodne riječi jednako informacijskome vektoru.

*Turbo kodovi*: Kodovi generirani paralelnim spajanjem dvaju (ili više) jednostavnih konvolucijskih kodera, odvojenih sklopovima za preplitanje.

*Ulančani kodovi*: Kodovi koji se temelje na serijskome spoju dvaju kodova odvojenih sklopom za preplitanje.

*Viterbijev algoritam*: Učinkovita metoda za dekodiranje konvolucijskih kodove na temelju pronalaska puta kroz dijagram rešetke koji je na minimalnoj udaljenosti od primljenog niza.

*Zalihost*: Dodatni bitovi (provjera pariteta bitova) dodani izvornoj poruci kako bi se omogućio ispravak pogrešaka.

[130] Napraviti simulacijski primjer

## 19. *Laboratorijska vježba 18: Konvolucijski kod brzine 1/2 s 4 stanja kao osnova TCM*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

# 9. POGLAVLJE 9 - PRAKTIČNO MOTRIŠTE LINEARNE ALGEBRE

Do sada su se koristile ograničene matematičke operacije, a uključivale su raznovrsne elementarne operacije linearne algebre koristeći aritmetiku modulo-2. Ovo nam je omogućilo doći do ove točke, a to je prilično daleko. Cjeloviti postupci kodiranja i dekodiranja u potpunosti su razumljivi. U ovome poglavlju pregledat ćemo matematičke alate posebno pogodne za oblikovanje ispravaka i analizâ pogrešaka. Osnovna načela iznesena u prethodnim poglavljima preispitat će se ponovno primjenom ovih alata, kako bi se cjelovito pokazala njihova vrijednost.

## *9.1. 9.1. Dioba polinoma modulo-2*

Prvo, razmatramo operaciju diobe polinoma modulo-2. Ovdje svi polinomi imaju binarne koeficijente. Operacije zbrajanja uključene u dijeljenje, izvode se modulo-2 (tj., $a + a = 0$). Drugim riječima, kao i prije, zbrajanje je naša standardna XOR operacija, a *zbrajanje "jednako je" oduzimanju, (množenju i/ili dijeljenju)*. Pokažimo kako podijeliti dva polinoma:

$$\underset{\text{djeljenik}}{x^6 + x^5 + x^3 + x^2 + 1} \;:\; \underset{\text{djelitelj}}{x^3 + x + 1} = \underset{\text{količnik}}{x^3 + x^2 + x^1 + x^0} \quad \underset{\text{ostatak}}{+ (x^2)}$$

U svakome koraku operacije dijeljenja, podijelimo *djeljenik* (*dividend* - broj kojega dijelimo) *najviše potencije*, *djeliteljem* (*divizor* - broj kojime dijelimo) *najviše potencije* pa napišimo rezultat kao *količnik* (*kvocijent* - rezultat dijeljenja). Onda *pomnožimo količnik* ukupnim *djeliteljem*, a rezultat *pribrojimo djeljeniku*. Na primjer, razmotrimo gornje uzorke *djeljenika* i *djelitelja*. Izraz s najvišim eksponentom u *djeljeniku* je $x^6$.

Izraz s najvišim eksponenta u djelitelju je $x^3$. Dijeljenjem $x^6$ s $x^3$ rezultira $x^3$, što je prvi član količnika. Množenje $x^3$ s $(x^3 + x + 1)$ daje $x^6 + x^4 + x^3$ (2. redak donje tablice). Sada pribrojimo taj rezultat djeljeniku $x^6 + x^5 + x^3 + x^2 + 1$ (1. redak donje tablice) i dobije se $x^5 + x^4 + x^2 + 1$ kao novi djeljenik (3. redak donje tablice). Opisani postupak piše se tehnički kao:

prvi član količnika

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. redak | $1\cdot x^6$ | $+1\cdot x^5$ | $+0\cdot x^4$ | $+1\cdot x^3$ | $+1\cdot x^2$ | $+0\cdot x$ | $+1$ | : | $1\cdot x^3$ | $+0\cdot x^2$ | $+1\cdot x$ | $+1 =$ | $1\cdot x^3$ |
| 2. redak | $1\cdot x^6$ | $+0\cdot x^5$ | $+1\cdot x^4$ | $+1\cdot x^3$ | $+0\cdot x^2$ | $+0\cdot x$ | $+0$ | + | | | | | |
| 3. redak | $0\cdot x^6$ | $+1\cdot x^5$ | $+1\cdot x^4$ | $+0\cdot x^3$ | $+1\cdot x^2$ | $+0\cdot x$ | $+1$ | | | | | | |

Primjenjujemo sada isti postupak na dobiveni polinom (novo dobiven djeljenik) $x^5+x^4+x^2+1$ u 3. retku. To jest, dijelimo $x^5$ (koji je trenutno član s najvišim eksponentom) s $x^3$ i dobijemo $x^2$. Sljedeći član količnika je, dakle, $x^2$. Onda pomnožimo $x^2$ s $(x^3 + x + 1)$ i dobijemo $x^5 + x^3 + x^2$ pa ga zbrojimo s $x^5 + x^4 + x^2 + 1$. Cijelovita operacija dijeljenja radi se na sljedeći način:

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $x^6$ | $+x^5$ | | $+x^3$ | $+x^2$ | | $+1$ | | : | $x^3$ | $+x$ | $+1$ | = | $x^3$ $+x^2$ $+x$ $+1$ |
| $x^6$ | | $+x^4$ | $+x^3$ | | | | + | | djelitelj | | | | količnik |
| | $x^5$ | $+x^4$ | | $+x^2$ | | $+1$ | | | | | | | |
| | $x^5$ | | $+x^3$ | $+x^2$ | | | + | | | | | | |
| | | $x^4$ | $+x^3$ | | | $+1$ | | | | | | | |
| | | $x^4$ | | $+x^2$ | $+x$ | | + | | | | | | |
| | | | $x^3$ | $+x^2$ | $+x$ | $+1$ | | | | | | | |
| | | | $x^3$ | | $+x$ | $+1$ | + | | | | | | |
| | | | | $x^2$ | | | ← stupanj ostatka *manji* je od stupnja *djelitelja* | | | | | | |

*Postupak prestaje* ako je *ostatak* polinom, čiji je *stupanj nižega reda* od stupnja *djelitelja*. Ovdje je naš ostatak $x^2$. Ostatak može, naravno, biti i 0. To se događa ako je djeljenik višekratnik djelitelja. Budući da naši polinomi imaju binarne koeficijente, mogu se napisati kao binarni vektori, predstavljajući samo koeficijente polinoma.

Na primjer, polinom $x^6 + x^5 + x^3 + x^2 + 1 = 1\cdot x^6 + 1\cdot x^5 + 0\cdot x^4 + 1\cdot x^3 + 1\cdot x^2 + 0\cdot x^1 + 1\cdot x^0$, prikazuje se kao vektor [1101101] dug sedam bitova. Bit na mjestu #*i* brojeći s desne strane, vrijednost je koeficijenta $x^i$, gdje je prvo mjesto označeno kao #0. Polinom $x^3 + x + 1$ prikazuje se kao [1011]. Navedena dioba onda ima sljedeći binarni prikaz:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | | : | 1 | 0 | 1 | 1 | = | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | | | | + | | | | | | | ↑ | ↑ | ↑ | ↑ |
| | 1 | 1 | 0 | 1 | 0 | 1 | | | | | | | | $x^3$ | $x^2$ | $x^1$ | $x^0$ |
| | 1 | 0 | 1 | 1 | | | + | | | | | | | | | | |
| | | 1 | 1 | 0 | 0 | 1 | | | | | | | | | | | |
| | | 1 | 0 | 1 | 1 | | + | | | | | | | | | | |
| | | | 1 | 1 | 1 | 1 | | | | | | | | | | | |
| | | | 1 | 0 | 1 | 1 | + | | | | | | | | | | |
| | | | | 1 | 0 | 0 | | ostatak = $x^2$ | | | | | | | | | |

Dakle,

$$\underbrace{x^6+x^5+x^3+x^2+1}_{\text{djeljenik}} \;:\; \underbrace{x^3+x+1}_{\text{djelitelj}} = \underbrace{x^3+x^2+x^1+x^0}_{\text{količnik}} \qquad \text{ostatak} = x^2$$

Napomena:

$$x^3+x+1.$$

"Jedinica" u polinomu kojega opisuje struktura LFSR, ne odgovara spoju-priključku (*tap*) LFSR - ona odgovara ulazu prvoga bita (tj. $x^0$, koji je jednak 1). Potencije izraza predstavljaju spojene (*tapped*) registre bitova, računajući s lijeva (ili s desna). Prvi i zadnji bitovi uvijek su povezani povratnom vezom kao ulazni odnosno izlazni spojevi.

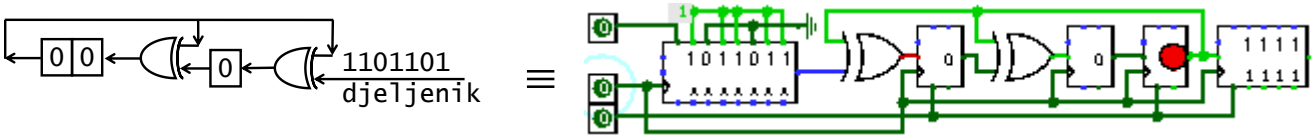
## 9.2. 9.2 Veza između diobe polinoma i LFSR sklopa

Kao što će se poslije pokazati, operacija od velike praktične koristi jest izračun $F(x)$ mod $G(x)$, tj., izračun ostatka koji se dobije nakon dijeljenja polinoma $F(x)$ s $G(x)$. Operacija diobe uvijek je u binarnome obliku kao što se obradilo u poglavlju 9.1. Vratimo se operaciji prikazanoj u prethodnome poglavlju (operacija [1101101] **:** [1011]). Operacija se može opisati kako slijedi:

- U prvome koraku, stavimo vektor [1011] ispod djeljenika, gdje lijeva 1 u [1011] dolazi ispod lijeve 1 u [1101101].
- Zbrajanjem dvaju vektora smanjuje se stupanj djeljenika za jedan.
- Sada učinimo istu operaciju na vektoru [110101], dobivenome prethodnim korakom. To jest, možemo pomaknuti djelitelj sve dok se lijeve 1 od [1011] i [110101] ne podudare, čime se dodatno smanjuje stupanj prethodnoga polinoma.
- Taj postupak ponavlja se dok posljednji ostatak nije polinom stupnja 2 ili manji (tj. binarni vektor duljine 3).

Navedeni postupak može se opisati i drugačije.

- Uzmimo polinom [1101101], odbacimo njegovu prvu jedinicu lijevo i okrenemo dva bita (komplement dvojke), koja se nalaze dva odnosno tri mjesta desno od ove 1. Dobije se vektor [110101].
- Učinimo istu operaciju nad dobivenim vektorima: [110101], [10011] itd. (tj., svakome vektoru odbacimo lijevu 1 i okrenimo bitove koji su dva odnosno tri mjesta s njegove desne strane).
- Ovaj postupak može se ugraditi u sklop prikazan na slici 9.1.

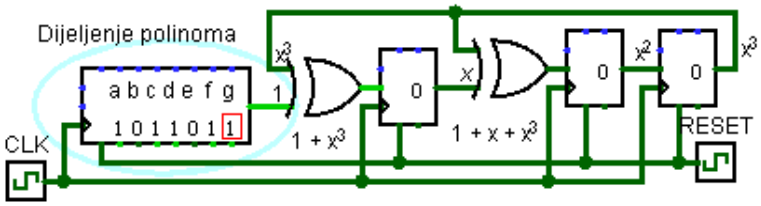



*Slika 9.1: Sklop za računanje* $(x^6+x^5+x^3+x^2+1)$ mod $(x^3+x+1)$

Koeficijenti djeljenika pune se u posmični LFSR (registar s desnom linearnom povratnom vezom na slici 9.1 - lijevo) koji se prethodno dovede u početno stanje "sve 0" (sva stanja registara su nula). Kada jedinica (1) napusti krajnje lijevo stanje, ona "ispadne" iz registra, ali se ta (1) (na "žici"), preko

povratne veze, pribraja bitovima koji su dva, odnosno tri mjesta desno. To je upravo ono što se čini u prije opisanoj ***operaciji dijeljenja***. Konačan sadržaj registra (nakon što se čitav djeljenik unese u registar), je traženi *ostatak*. Specifičan pristup dijeljenju $x^6 + x^5 + x^3 + x^2 + 1$ u gornjemu primjeru može se zamijeniti bilo kojim općim polinomom $G(x)$. Sklop na slici 9.1 lijevo, općenit je sklop koji generira $G(x)$ mod $(x^3 + x + 1)$ (tj. ostatak dijeljenja).

Pokazano promatranje čini, da je *sklop* na *slici 9.1 jednak sindrom-generatoru prikazanome na* slici 3.2 (*i ponovno nacrtan kao slika 9.2*).




$R(x) = 1 \oplus x \oplus x^3$

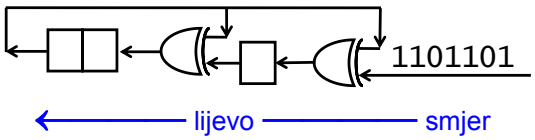
| | $x^0x^2x^3x^5x^6$ | 1 | + | $x$ | + | ~~$x^2$~~ | + | $x^3$ |
|---|---|---|---|---|---|---|---|---|
| $(x^6 + x^5 + x^3 + x^2 + 1)$ | : | $(x^3 + x + 1)$ | | | | | | |

*Slika 9.2: Krug za dijeljenje polinoma* $x^6+x^5+x^3+x^2+1 : x^3+x+1 = x^3+x^2+x^1+x^0$ *(precrtano iz slike 3.2)*

| *Sadržaji registra* | | | |
|---|---|---|---|
| $1 \equiv L_{ijevi}$ | $x \equiv S_{rednji}$ | $x^2 \equiv D_{esni}$ | |
| 1 | 0 | 0 | |
| 1 | 1 | 0 | |
| 0 | 1 | 1 | $\equiv x^3$ |
| 0 | 1 | 1 | $\equiv x^2$ |
| 0 | 1 | 1 | $\equiv x^1$ |
| 1 | 1 | 1 | $\equiv x^0$ |
| 0 | 0 | **1** | ostatak |

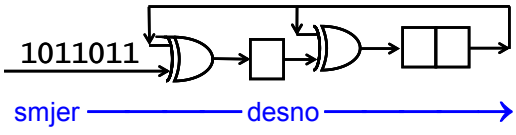
Količnik

Jedan krug zapravo je odraz drugoga. Oni su, naravno, potpuno jednaki, jer krug neće promijeniti svoj rad ako se okrenu strane (smjer posmika lijevo ili desno) kao što to prikazuju slike 9.1 i 9.2.




*Slika 9.1 (ponovljena)*

≃




*Slika 9.2 (ponovljena)*

Od sada ćemo se držati uobičajenoga načina crtanja, kao što prikazuje slika 9.2, gdje se ulaz napaja s lijeva, a registar R1 napušta se s desna. Kako bi se učvrstila pokazana jednakost između dviju slika, trebamo sljedeće definicije.

*Definicija* Polinom koji odgovara vektoru **m** (ili polinom predstavljen kao $M$), odnosi se na polinom čiji su koeficijenti bitovi vektora **m**. Označimo ovaj polinom $M(x)$.

*Primjer* Polinom koji odgovara vektoru **m** = [1001011] je $M(x) = 1 + x^3 + x^5 + x^6$. Imajte na umu da je bit sasvim desno u vektoru **m**, koeficijent s najvišim eksponentom. Držat ćemo se ovakvoga dogovora, ako se drugačije ne naznači.

*Definicija* Slijedi podrobnije objašnjenje pojedinosti.

1. **Polinom podudaran LFSR** je polinom čiji su koeficijenti odgovarajuće povratne veze LFSR.
2. **LFSR podudaran polinomu** je LFSR čije povratne veze odgovaraju koeficijentima polinoma.

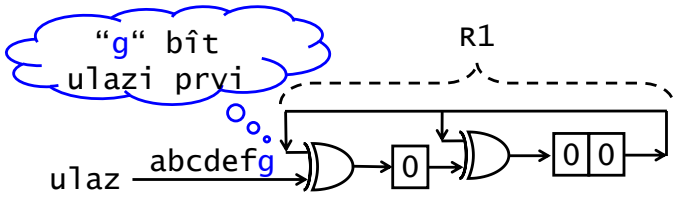
Ako R predstavlja LFSR, onda $R(x)$ označava njegov odgovarajući polinom i obrnuto.

Poveznica između polinoma i njemu odgovarajućega LFSR može se objasniti na sljedeći način: najprije je potrebno da broj stanja LFSR bude jednak *stupnju* polinoma djelitelja. Označimo stanja LFSR, počevši s lijeve strane, uzastopnim potenciranjem $x$, počevši s 0. Onda je stanje sasvim desno, označeno kao $x^{n-1}$, gdje je $n$ stupanj polinoma. *Povratna veza, što izlazi iz stanja koje je sasvim desno, vodi se natrag prema svim stanjima obilježenim potencijama od x koje se pojavljuju u polinomu.*

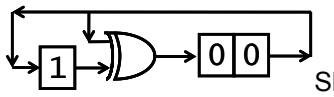
Povratna veza omogućuje određivanje odgovarajućega LFSR iz zadanoga polinoma. Sličan je postupak određivanja polinoma koji se podudara s određenim LFSR. Ne-nulti koeficijenti polinoma podudarni su onim stanjima LFSR koja su spojena na vrata EXOR zajedno s povratnom vezom. Ako je duljina LFSR jednaka $n$, koeficijent uz $x^n$ u polinomu jednak je 1.

*Primjer*

1. Uočite vezu između polinomnim $1 + x + x^3$ i LFSR na slikama 3.2 ili 3.1.

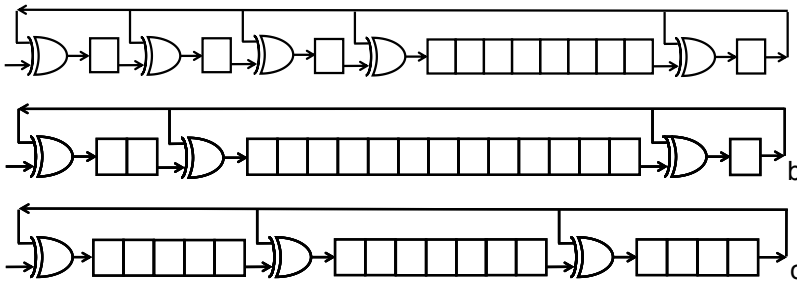


Slika 3.1



Slika 3.2

2. Promotrimo četiri LFSR prikazana na slici 5.11, a koja su precrtana u sliku 9.3.




$P(x) = 1 \oplus x \oplus x^2 \oplus x^3 \oplus x^{11} \oplus x^{12}$

$P(x) = 1 \oplus x^2 \oplus x^{15} \oplus x^{16}$

$P(x) = 1 \oplus x^{16} \oplus x^5 + x^{12}$

$P(x)$ = CRC-32: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

d)

*Slika 9.3 Ponovno nacrtana slika 5.11*

Polinomi koji odgovaraju ovim LFSR su:

(a) CRC-12: $P(x) = x^{12} + x^{11} + x^3 + x^2 + x + 1$;

(b) CRC-16: $P(x) = x^{16} + x^{15} + x^2 + 1$;

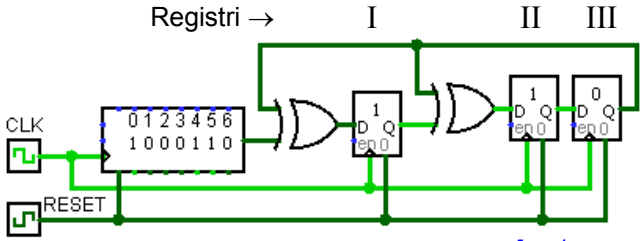
(c) CRC-CCITT: $P(x) = x^{16} + x^{12} + x^5 + 1$.

(d) CRC-32: $P(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Promatranje napravljeno na početku ovoga poglavlja, u vezi sličnosti između slike 9.1 i slike 9.2, sada se može izričito iskazati.

*Tvrdnja 9.1* Označimo LFSR s R. Neka **c** označava sadržaj R dobiven nakon posmika vektora **v** u njega. Onda je $C(x) = M(x) \bmod R(x)$. Drugim riječima, sadržaji R, dobiveni nakon posmika **v** u njega, oblikuju koeficijente ostatka $C(x)$ dobivene dijeljenjem polinoma $V(X)$, odgovarajućim polinomom $R(x)$. Ne daje se dokaz Tvrdnje 9.1 na formalan način. Detaljan prethodan prikaz, jasno potvrđuje tezu.

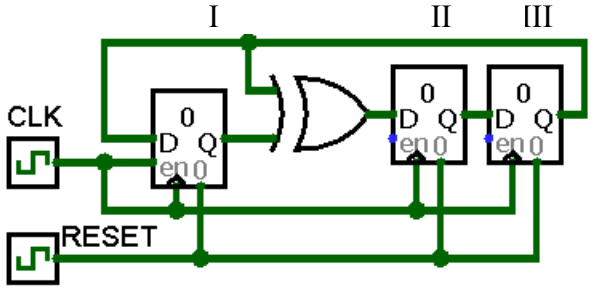
Važno je napomenuti da Tvrdnja 9.1 zapravo znači: LFSR, povezan u oblik prikazan na slici 9.2 ili na slici 9.3, predstavlja ***krug za dijeljenje***. Posmikom *ulaznoga polinoma* (vektora) u LFSR, ulazni polinom *dijeli se* polinomom $R(x) = 1 + x + x^3$ što ga oblikuju povratne veze LFSR. ***Konačan sadržaj LFSR*** je ***ostatak operacije dijeljenja*** $C(x)$ (slika 9.4).




*Slika 9.4: (vidi: Sliku 9.2) Dioba binarnih polinoma:* $(x^5+x^4+1) : (x^3+x+1) = x^2+x$

| Sadržaji registra | | | | |
|---|---|---|---|---|
| $1 \equiv L_{ijevi}$ | $x \equiv S_{rednji}$ | $x^2 \equiv D_{esni}$ | | |
| 0 | 0 | 0 | | |
| 1 | 0 | 0 | | |
| 1 | 1 | 0 | 0 | količnik |
| 0 | 1 | 1 | 1 | |
| 1 | 1 | 1 | 1 | |
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | ostatak | |

Promatrajući rad kruga na slici 9.1 precrtanoga i vodoravno zaokrenutoga oko okomite osi, lako se može provjeriti da ***uzastopni sadržaji*** *desnoga* stanja (III) LFSR oblikuju koeficijente *količnika* operacije dijeljenja. U krugu na slici 9.1 količnik se generira u stanju sasvim lijevo, a u krugovima na slikama 9.2, 9.3 ili na slici 9.4, količnik se generira u stanju sasvim desno. Međutim, u primjenama koje su značajne za kodove zaštite od pogrešaka (npr. CRC kodiranje), ***nema praktične potrebe za količnikom***. Sada ćemo protumačiti ponašanje slobodno pokrenutoga LFSR s motrišta polinoma. Takav LFSR prikazuje slika 3.1, a ponovno je nacrtan na slici 9.5.




$$R(x) = 1 \oplus x \oplus x^3$$

*Slika 9.5 Ponovno nacrtana slika 3.1*

*Tvrdnja 9.2* Neka R predstavlja LFSR u samostalnome radu. Njegov sadržaj nakon *i*-toga posmika, počevši početnim sadržajem [1000 … 0], polinomski je prikaz $x^i \bmod R(x)$.

*Primjer* Tablica 9.1 popisuje uzastopan sadržaj registra na slici 9.5 i njegove odgovarajuće polinome.

*Tablica 9.1 Uzastopan sadržaj registra na slici 9.5* $[R(x) = 1 \oplus x \oplus x^3]$[131]

| *Broj posmika #* | *Binaran sadržaj* *Stanja registara* | *Jednadžba sklopa* $R(x) = 1 \oplus x \oplus x^3$ |
|---|---|---|
| 0 | 1 0 0 | $1 = x^0 \bmod R(x)$ |
| 1 | 0 1 0 | $x^1 = x^1 \bmod R(x)$ |
| 2 | 0 0 1 | $x^2 = x^2 \bmod R(x)$ |
| 3 | 1 1 0 | $1 + x^1 = x^3 \bmod R(x)$ |
| 4 | 0 1 1 | $x^1 + x^2 = x^4 \bmod R(x)$ |
| 5 | 1 1 1 | $1 + x^1 + x^2 = x^5 \bmod R(x)$ |
| 6 | 1 0 1 | $1 + x^2 = x^6 \bmod R(x)$ |
| 7 | 1 0 0 | $1 = x^7 \bmod R(x)$ |
| itd. … | | Polinomski prikaz ostatka modulo operacije |

*Dokaz Tvrdnje 9.2* Za svrhe udžbenika i koristeći neposredno stečene spoznaje, slutnjom (intuicijom) priskrbljuju se dvije različite inačice dokaza.

***Inačica 1***

Početni sadržaj [100 … 0] odgovara polinomu $x^0$. Općenito, polinom $x \cdot F(x)$ dobije se posmikom svih koeficijenata $F(x)$ za jedno mjesto u desno. Nakon prvih $n$–1 posmika, kada se dobije 1-element u $n$-tome stanju, sadržaj registra odgovara izrazu $x^n$–1. Nakon dodatnih posmika, što predstavlja dodatno množenje s $x$, sadržaj registra odgovara izrazu $x^n$. Ono što se zapravo događa je da će registar imati 1 u svim stanjima spojenima na povratnu vezu.

[131] Struktura LFSR kao djelitelja

Tvrdnja da ti sadržaji odgovaraju polinomu $x^n$ mod $R(x)$, temelji se na sljedećemu opažanju. $R(x)$ je stupnja $n$. Uzimajući $x^n$ mod $R(x)$ rezultira polinomom koji se dobije odbacivanjem izraz $x^n$ iz $R(x)$. [Na primjer, $x^3$ mod $(1+x+x^3) = 1+x$]. Naše povratne veze onda upravo rade tako da se još jednim posmikom sadržaja $x^n-1$, dobije sadržaj $x^n$ mod $R(x)$. Ovo objašnjava zašto uzastopni posmici registra, generiraju uzastopne potencije $x$ mod $R(x)$.

Ovo ćemo još objasniti, koristeći primjer. Razmotrimo registar na slici 9.5, čije povratne veze odgovaraju polinomu $R(x) = 1 + x + x^3$. Znamo da je $x^3$ mod $(1 + x + x^3) = 1 + x$. Promatrajmo sada uzastopne sadržaje registra, navedene u tablici 9.1. Prateći sadržaj 001, 1 sasvim desno se odbaci, a zamjeni ju $1 + x = x^3$ mod $(1 + x + x^3)$. Svaki dodatan posmik u registru, množi sadržaj s $x$, dakle, stvara uzastopne potencije $x$, gdje se $x^3$ uvijek zamjenjuje $x^3$ mod $R(x)$. To jest, potencije od $x$ generiraju se modulo $R(x)$.

***Inačica 2***

Ova inačica dokaza Tvrdnje 9.2 temelji se na vezi između LFSR u samostalnome radu, oblika prikazanoga na slici 9.5 i kruga za dijeljenje, oblika prikazanoga na slici 9.2. Dokaz koji se temelji na Tvrdnji 9.1, posebno razmatra ova dva LFSR. Njihova općenitost onda je očita.

Zbog jednostavnosti, registar na slici 9.5 označava se R, dok je onaj na slici 9.2 označen R1. Najprije smo utvrdili da je sadržaj registar R nakon $i$ posmika, započevši stanjem [100], jednak konačnome sadržaju R1. Taj sadržaj dobiven nakon posmika vektora $\mathbf{v}_i$ duljine 7 u njega, sadrži jedan 1-element na mjestu #$i$, računajući s lijeva. Prvo mjesto na lijevoj strani je #0. Da bi razumjeli zašto, primijetimo da je početni sadržaj R1 "sve 0". Sadržaj će ostati "sve 0" tijekom prvih 6–$i$ posmika, nakon čega se u njega posmiče 1. Sadržaj R1 je onda [100].

Ovaj registar sada će se pokrenuti slobodno poput registra R za dodatnih $i$ posmika, sve dok se ne unese ostatak $\mathbf{v}_i$. Njegov konačan sadržaj je sadržaj R nakon $i$ posmika, započevši početnim stanjem [100].

*Primjer* Posmik vektora $\mathbf{v}_4$ = [0000100] u $R_i$. Za prva 2 posmika, R1 i dalje sadržava "sve 0". Onda se učita 1 i krug radi posmik još četiri puta. Konačan sadržaj R1 je [011], a to je sadržaj R nakon četiri posmika, započevši početnim stanjem [100].

Nastavljajući dokaz, sada primjenjujemo Tvrdnju 9.1. Polinomski prikaz $\mathbf{v}_i$ je $V_i(x) = x^i$. Onda imamo da sadržaj R nakon $i$ posmika, počevši početnim stanjem [100], predstavlja polinom $x^i$ mod $(1+x+x^3)$. Time je dovršen dokaz Tvrdnje 9.2.

Imajte na umu sljedeće, u poglavlju 3 prikazalo se da slobodno pokrenut LFSR na slici 9.5 stvara retke matrice $\mathbf{H}^T$. Tvrdnja 9.2 onda znači da $i$-ti *redak* matrice $\mathbf{H}^T$ *predstavlja polinom* $x^i$ mod $R(x)$, gdje je $R(x) = (1+x+x^3)$. Tvrdimo da za $\mathbf{b}$ kao vektor duljine 7, umnožak $\mathbf{b} \cdot \mathbf{H}^T$ daje polinom $B(x)$ mod $R(x)$. Ovo se može prikazati na sljedeći način: Za $\mathbf{b}$ = [*abcdefg*],

$$\mathbf{b} \cdot \mathbf{H}^T = a[x^0 \bmod R(x)] + b[x^1 \bmod R(x)] + c[x^2 \bmod R(x)] + d[x^3 \bmod R(x)] + e[x^4 \bmod R(x)] + f[x^5 \bmod R(x)] + g[x^6 \bmod R(x)]$$

Na temelju poznate osnovne jednadžbe, desni dio gornje jednadžbe jednak je:

$$[a + bx + cx^2 + dx^3 + ex^4 + fx^5 + gx^6] \bmod R(x) = B(x) \bmod R(x).$$

Činjenica da umnožak $\mathbf{b} \cdot \mathbf{H}^T$ daje polinom $B(x)$ mod $R(x)$ ne začuđuje, jer znamo da je slika 9.2, s jedne strane, ***krug za dijeljenje***, a s druge strane, sklop koji ***množi vektor*** i ***matricu*** (kao što se pokazalo u poglavlju 3). Razmatranja povezana matricom $\mathbf{H}^T$ su, dakle, na neki način, ponavljanje Tvrdnji 9.1 i 9.2. Ona su ovdje iznijeta za rasvjetljavanje činjenice iz drugoga kuta.

### 9.2.1. 9.2.1. KRUG ZA DIJELJENJE POLINOMA

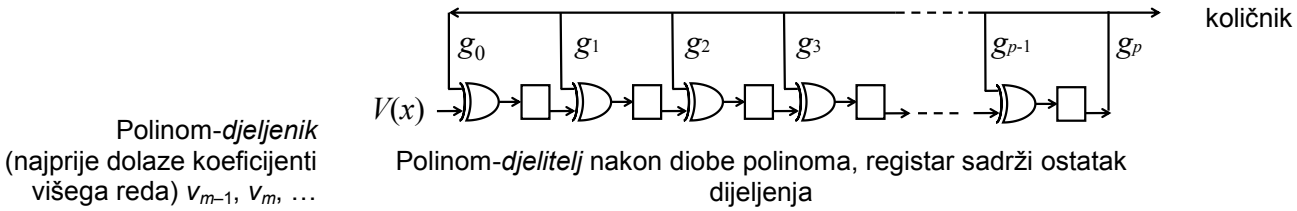
Vidjeli smo da ciklički pomak u polinomskoj kodnoj riječi i kodiranje polinomske poruke podrazumijeva diobu jednoga polinoma drugim. Takva operacija lako se ostvaruje *krugom za dijeljenje* (posmični registar s povratnom vezom). Zadana su dva polinoma $M(x)$ i $G(x)$, gdje su

$$M(x) = v_0 + v_1x + v_2x^2 + \ldots + v_mx^m$$

i

$$G(x) = g_0 + g_1x + g_2x^2 + \ldots + g_px^p$$

tako da uz $m \geq p$, krug za dijeljenje na slici 9.6, postupno dijeli polinom $M(x)$ polinomom $G(x)$.




*Slika 9.6: Opći oblik kruga za diobu polinoma polinomom.*

Tako se određuju izrazi količnika i ostatka:

$$\frac{V(x)}{G(x)} = Q(x) + \frac{P(x)}{G(x)} \tag{9.1}$$

Stanja registra najprije se postavljaju tako da se pune nulama. Prvih $p$ posmika unose najznačajnije koeficijente (višega reda) od $M(x)$. Nakon $p$-toga posmika, dobije se količnik na izlazu. Ovo je član u količniku najvišega reda. Za svaki koeficijent količnika $q_i$, polinom $q_iG(x)$ mora se oduzeti od djeljenika. Povratne veze na slici 9.6 ostvaruju ovo oduzimanje.

#### 9.2.1.1. 9.2.1.1. Primjer 9.1 Krug za dijeljenje

Za podijeliti polinomom $M(x) = x^3 + x^5 + x^6$ (**v** = [0001011]) s $G(x) = 1 + x + x^3$ (**g** = [1101]), koristimo krug za dijeljenje polinoma, oblika kao što ga prikazuje slika 9.6.

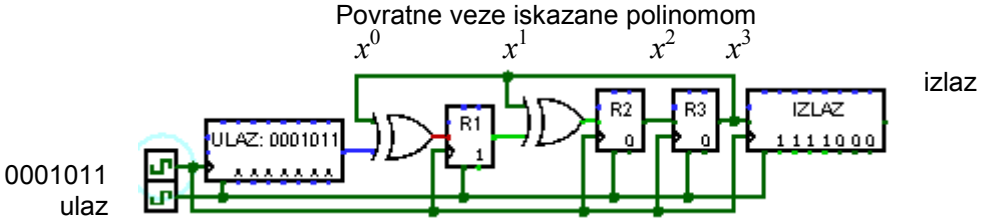
Razlika između $p$ lijevih članova preostalih u *djelitelju* (LFSR) i članova povratnih (veza) $q_i\mathbf{g}(x)$ oblikuju se pri svakome posmiku u krugu i pojavljuju se kao sadržaji registra. Pri svakome posmiku registra, član najvišega stupnja (razlika) posmiče se za jedno mjesto u desno i izlazi iz registra, a sljedeći koeficijent najviše potencije koji je preostao u vektoru $\mathbf{v}(x)$ na ulazu u posmični registar, posmiče se u registar. Nakon $m$+1 ukupnih posmika u registar, *količnik* se serijski prikazuje na izlazu, a *ostatak* ostaje u registru. Nađimo izraze za *količnik* i *ostatak*. Usporedimo provedbu kruga koracima diobe polinoma obavljenih ručno.

***Rješenje***

Krug za dijeljenje treba izvesti sljedeće operacije:

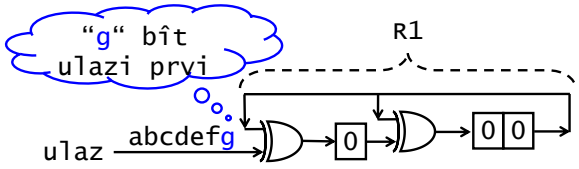
$$\frac{x^3 + x^5 + x^6}{1 + x + x^3} = Q(x) + \frac{P(x)}{1 + x + x^3}$$

Opći oblik potrebnoga LFSR, prikazuje slika 9.7.




*Slika 9.7: Krug za dijeljenje polinoma* $(x^3 + x^5 + x^6)/(1 + x + x^3)$ *iz primjera 9.1.*

Podsjetimo se redoslijeda ulaska bitova u LFSR za dijeljenje polinoma (prvi ulazi koeficijent najvišega stupnja u polinomu, a to je bit $g$ na slici 9.8):

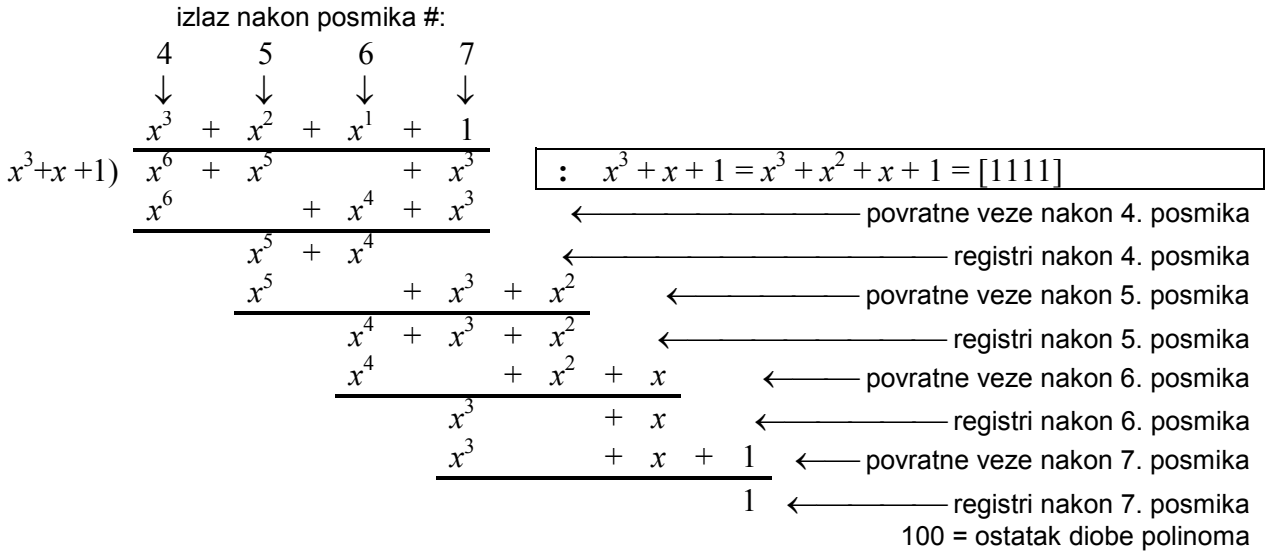



*Slika 9.8: U LFSR za dijeljenje polinoma prvi ulazi bit najviše potencije*

Pretpostavimo da su sadržaji registra u početku nula. Radne korake kruga za dijeljenje prikazuje sljedeća tablica:

| Ulazni spremnik | | | | | | | Broj posmik | Sadržaji registra | | | | | | | Povratna veza polinoma | Izlaz |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | $x^1$ | $x^2$ | $x^3$ | $x^4$ | $x^5$ | $x^6$ | $p$ | $x^0$ | R1 | $x^1$ | R2 | $x^2$ | R3 | $x^3$ | | |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | | 0 | | 0 | | 0 | | – | – |
| | 0 | 0 | 0 | 1 | 0 | 1 | 1 | | 1 | | 0 | | 0 | | 0 | – |
| | | 0 | 0 | 0 | 1 | 0 | 2 | | 1 | | 1 | | 0 | | 0 | – |
| | | | 0 | 0 | 0 | 1 | 3 | | 0 | | 1 | | 1 | | 1 | – |
| | | | | 0 | 0 | 0 | 4 | | 0 | | 1 | | 1 | | 1 | **1** |
| | | | | | 0 | 0 | 5 | | 1 | | 1 | | 1 | | 1 | **1** |
| | | | | | | 0 | 6 | | 1 | | 0 | | 1 | | 1 | **1** |
| | | | | | | – | 7 | | **1** | | **0** | | **0** | | 1 | **1** |

Nakon ($p$+1)-toga posmika ($p$ = 6), koeficijenti količnika $\{q_i\}$ prikazani su serijski na izlazu, a vidljivo je da su [1111], odnosno *količnik* polinoma (preostao u LFSR) je $\mathbf{q}(x) = 1 + x + x^2 + x^3$. Koeficijenti koji su preostali kao sadržaji registra $[p_i]$ su 100, odnosno ostatak polinoma je $p(x) = x^0 + x^1 + x^2 =$ [100]. Krug za izračun $\mathbf{v}(x)/\mathbf{g}(x)$ može se prikazati kao:



## 20. *Laboratorijska vježba 19: FCS – Niz za provjeru dugoga polinoma $x^nM(x)$ s $G(x)$ (modulo-2 dioba)*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

### 9.3.1. 9.3.1 PREISPITIVANJE NEKIH OSNOVNIH SVOJSTAVA CIKLIČKOGA KODA

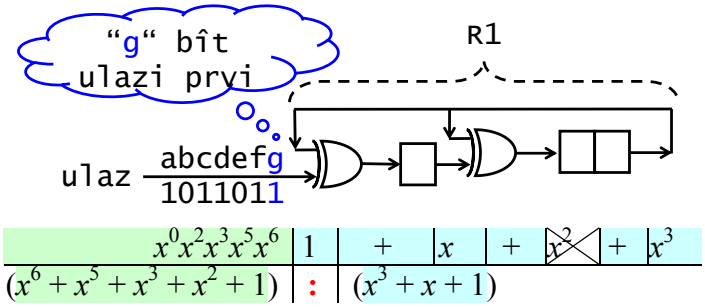
Kao što se definiralo u poglavlju 4, ciklički kod je bilo koji kod što ga generira LFSR. Također je objašnjeno da svaki takav kod zapravo nema ciklička svojstva, u smislu da je ciklički pomak kodne riječi također kodna riječ, osim ako duljina kodne riječi nije jednaka *Pu*. Od sada pa nadalje obrađujemo samo te kodove. Na temelju Tvrdnje 9.1 imamo sljedeći ***glavni*** zaključak.

*Zaključak 9.1* Neka je **m** kodna riječ cikličkoga koda koja se prenosi, **m'** je primljena inačica **m**. Neka R označavaju LFSR-generator koda i neka **s** označava sindrom pogreške dobiven pomicanjem **m'** u R.

Onda je $S(x) = \mathbf{m'}(x) \bmod R(x)$.

*Tvrdnja 9.3* Neka je *C* ciklički kod što ga stvara registar R, gdje je *n* duljina kodne riječi. Onda je vektor **v** duljine *n*, kodna riječ u *C* onda i samo onda ako je $M(x)$ višekratnik $R(x)$.

*Primjer* Razmotrimo (7, 4) kod što ga stvara LFSR na slici 9.2 (slika 9.9).




*Slika 9.9: Krug za dijeljenje polinoma* $x^6x^5+x^3+x^2+1 : x^3 + x + 1$=[1111]+[100] (*precrtano iz slike 3.2*)

Ovdje je $R(x) = (1 + x + x^3)$. Vektor **v** = [1101101] je kodna riječ našega koda dok je polinom $M(x) = 1 + x + x^2 + x^5$, koji odgovara **v**, djeljiv s $R(x)$. S druge strane, vektor **w** = [1111000] nije kodna riječi našega koda, jer polinom $W(x) = 1 + x + x^2 + x^3$ nije djeljiv s $R(x)$.

*Dokaz Tvrdnje* 9.3 Znamo iz poglavlja 3 da je **v** kodna riječ u *C* onda i samo onda ako je sindrom pogreške, koji se dobije posmikom **v** u R jednak "sve 0". Iz Zaključka 9.1 znamo da je sindrom "sve 0" "matematički jednak jednadžbi $M(x) \bmod R(x) = 0$. Nula (0) na desnoj strani jednadžbe, je *polinom* 0, tj., polinom čiji su koeficijenti "sve 0"). Ova jednadžba znači da je pri dijeljenju $M(x)$ s $R(x)$, ostatak jednak 0. To jest, $M(x)$ je višekratnik od $R(x)$. Ovo upotpunjuje dokaz Tvrdnje 9.3.

Zbog lakšega snalaženja, od sada nećemo razlikovati vektor **v** i polinom $M(x)$. Rečenica poput "$F(x)$ je kodna riječ u *C*" apsolutno je ispravna (ona zapravo znači *vektor* **f**, kojemu odgovara $F(x)$ je kodna riječ u *C*).

*Zaključak 9.2* Neka je *C* (*n*, *k*) ciklički kod što ga generira registar R duljine *n*–*k*. Kodne riječi u *C* su svi polinomi stupnja *n*–1 ili manjega, višekratnici su od $R(x)$. Stupanj $R(x)$ je duljina koda kojega generira LFSR. To je ujedno i broj paritetnih bitova. Ako se primljena poruka obrađuje kao polinom s ciljem provjere pripada li skupu ispravnih kodnih riječi, zapravo se provjerava je li djeljiva s $R(x)$.

*Definicija* Neka je *C* ciklički kod čije su kodne riječi svi polinomi stupnja *n*–1 ili manjega, a koje su djeljive polinomom $R(x)$. $R(x)$ zove se *polinom-generator koda*.

Nakon uvođenja aritmetike polinoma kao osnovnoga alata za rukovanje cikličkim kodovima, pregledajmo Tvrdnju 4.1 pa ponovno vrijedi sljedeće.

*Tvrdnja 4.1* Neka je *C* (*n*, *k*) kod čija je matrica pariteta **P** i neka **G** označava generator-matricu od *C*. Retci **P**, stvaraju se uzastopnim posmikom u LFSR duljine *q*, započevši početnim stanjem [1000 … 0].

1. Bilo koja kodna riječ u *C* sastoji se od zbroja pojedinih linearnih posmika prvoga retka od **G** u desno, gdje posmici nisu veći od *k*–1 mjesta.
2. Svaki vektor duljine *n* koji se sastoji od zbroja nekih linearnih posmika u desno prvoga retka od **G**, gdje posmici nisu veći od *k*–1 mjesta, mora pripadati skupu kodnih riječi u *C*.

Kao što se već objasnilo, $q = n–k$. Prvih $q$ bitova u prvome retku od **G** označuju povratne veze LFSR-generatora, ($q$+1)-vi bit je 1, a ostali bitovi su 0. Na temelju iskustva koje smo stekli u ovome poglavlju, jasno je da su prvih $q$+1 bitova u prvome retku **G**, koeficijenti polinom-generatora $R(x)$ koda, čiji je stupanj $n–k$, gdje linearan posmik za $i$ mjesta u desno ovoga retka, daje polinom $x^i \cdot R(x)$. Tvrdnja 4.1 znači da je $M(x)$ kodna riječ u $C$ onda i samo onda ako $M(x)$ ima oblik $A0 \cdot R(x) + A1 \cdot x \cdot R(x) + A2 \cdot x^2 \cdot R(x) + \ldots + A(K–1) \cdot x^{k–1} \cdot R(x)$.

Tvrdnja 4.1 znači da je $M(x)$ kodna riječ u $C$ onda i samo onda ako je $M(x) = R(x) \cdot (A0 + A1 \cdot x + A2 \cdot x^2 + \ldots + A(K–1) \cdot x^{k–1})$. Imajte na umu da je sada ovaj zadnji izraz opći oblik bilo kojega polinoma stupnja $n$–1 ili manjega, a koji je višekratnik od $R(x)$. Stupanj je $n$–1 ili manji dok je $R(x)$ stupnja $n–k$. Množenje polinomom stupnja $k$–1 ili manjim, daje polinom stupnja $n$–1 ili manji. Onda imamo da je *Tvrdnja 4.1 jednaka zaključku 9.2*.

### 9.3.2. 9.3.2. PREISPITIVANJE OPĆEGA POSTUPKA DEKODIRANJA CIKLIČKOGA KODA, A POSEBNO HAMMINGOVA KODA ZA ISPRAVAK JEDNOSTRUKIH POGREŠAKA

Razmotrimo sada polinom s motrišta aritmetike, pri postupku dekodiranja cikličkoga koda. Svojstvo primljene kodne riječi $M(x)$ takvoga koda je djeljivost određenim polinom-generatorom $R(x)$. Prijemom poruke $V'(x)$, postupak dekodiranja u prijemniku počinje provjerom je li $V'(x)$ još uvijek djeljiv s $R(x)$. To se radi posmikom $V'(x)$ u krug za dijeljenje. Ova operacija daje polinom sindroma pogreške $S(x) = V'(x) \bmod R(x)$.

Koeficijenti $S(x)$ su, dakle, "sve 0" onda i samo onda ako je $V'(x)$ djeljiv s $R(x)$. Na taj način, utvrdili smo da je $V'(x)$ važeća kodna riječ. Ako $S(x)$ ima najmanje jedan koeficijent različit od nule, otkriva se da je $V'(x)$ poruka s pogreškom. Pogledajmo sada neke detalje postupka za ispravak jednostrukih pogrešaka Hammingova koda. Ako se primljena poruka $V'(x)$ razlikuje od poslane poruka $M(x)$ samo i jedino u koeficijentu #$j$, onda je $M'(x) = M(x)+x^j$. Sada imamo da je: $S(x) = M'(x) \bmod R(x) = (M(x) + x^j) \bmod R(x) = M(x) \bmod R(x) + x^j \bmod R(x)$. Kako je $M(x) \bmod R(x) = 0$, imamo da je:

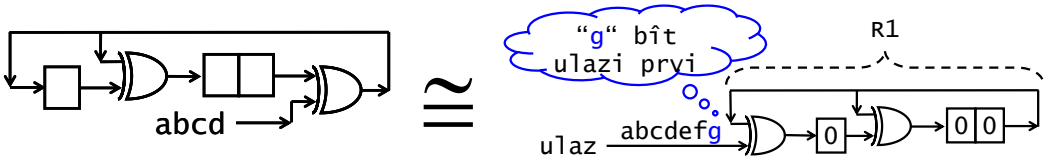
$$S(x) = x^j \bmod R(x). \qquad (9.2)$$

Da bi pronašli mjesto pogreške $j$, krug oblika što ga prikazuju slike 3.25 i 3.26b, generira uzastopne potencije $x \bmod R(x)$, počevši s $x^j$, dok se ne otkrije vrijednost $x^0$. Kako je $x^0 = x^{2^n–1}$, slijedi da je broj posmika jednak $2^n–1–j$, tj. moguće je odrediti vrijednosti $j$.

### 9.3.3. 9.3.3. PREISPITIVANJE POSTUPKA KODIRANJA

Postupak kodiranja cikličkih kodova opisao se u poglavlju 4. Ovdje dajemo izravniji pristup korištenjem aritmetike polinoma. Ako je $R(x)$ polinom-generator koda, postupak kodiranja pretvorit će informacijski polinom $I(x)$ stupnja $k$–1 ili manjega (što odgovara informacijskome vektoru **i** duljine $k$), u kodnu riječ $M(x)$ djeljivu s $R(x)$. Očit način dobivanja takvoga $M(x)$ jednostavan je množenjem $I(x)$ s $R(x)$.

Ovo ne bi vodilo sustavnome kodu (u kojemu bi se $k$–1 uzastopnih koeficijenata $M(x)$ trebalo sastojati od koeficijenta $I(x)$). Postupak kodiranja, što se prikazao na slici 3.14 i ponovo nacrtao na slici 9.10, generira $M(x)$ sustavan kod.




*Slika 9.10: Ponovno nacrtana slika 3.3 (koder)*

Sada ispitujemo ovaj postupak u smislu rukovanja polinomima. Pogledajmo najprije vezu između rada krugova na slikama 9.2 i 9.9. Tvrdimo da je sadržaj R1 na slici 9.2, dobiven posmikom vektora [000*abcd*] u njega, jednak sadržaju LFSR na slici 9.9 dobiven posmikom [000*abcd*] u njega.

Ovo se može potvrditi izravnom provjerom, pokazujući da su konačni sadržaji u oba slučaja [$a$+$c$+$d$, $a$+$b$+$c$, $b$+$c$+$d$]. Logika koja stoji iza ove jednadžbe slijedi iz promatranja, da u prvome slučaju, prva tri posmika guraju $d$ u desno stanje. Rad ovoga kruga za sljedeća četiri posmika, sada je jednak radu

drugoga kruga, počevši od prvoga posmika, budući da drugi krug *započinje* radom u koraku gdje se *d* već uklonio iz desnoga stanja LFSR. Dalje ćemo analizirati rad kodera na slici 9.10 u smislu posmika vektora [000*abcd*] u krug dijeljenja na slici 9.2. Informacijski vektor predstavlja polinom $I(x) = a+bx+cx^2+dx^3$.

Vektor **j** = [000*abcd*] predstavlja polinom $J(x) = ax^3 + bx^4 + cx^5 + dx^6$. Sadržaj R1 nakon što se $J(x)$ unese u njega je $P(x) = J(x) \bmod (1 + x + x^3)$. Polinom $J(x) + P(x)$ je, dakle, djeljiv s $x^3 + x + 1$. Ako se od vrijednosti $A$ oduzme vrijednosti $A$ mod $B$, rezultat je djeljiv s $B$. Ovdje ćemo zbrajati, jer su *XOR* operacije *zbrajanja* i operacije *oduzimanja* jednake. Kako je stupanj $P(x)$, jednak 2 ili manji, zbrajanjem $P(x)$ i $J(x)$ jednako je zamjeni triju 0 na repu **j** sadržaja R1. Vratimo li se na koder slike 9.10, već smo pokazali da pridruživanje sadržaja *abcd* LFSR (nakon što se *abcd* pomakne u njega), generira polinom koji je djeljiv s $(1 + x + x^3)$ te da su u njemu četiri uzastopna koeficijenta *abcd*, koja su potrebna kako bi kod bio sustavan.

## 9.4. 9.4. Ciklička svojstva polinoma

### 9.4.1. 9.4.1. PERIODIČNOST LFSR KOJA SE ODRAŽAVA U NJEGOVOMU PRIPADAJUĆEM POLINOMU

Neka LFSR započne posmik početnim sadržajima *x*. Njegovu periodičnost u odnosu na početni sadržaj [1000 … 0] označimo *Pu*. LFSR maksimalne periodičnosti ima svojstvo da ako se pokrene bilo kojom ne-nultom početnom vrijednošću, prolazi kroz sva moguća $2^n$–1 ne-nulta stanja prije povratka na početnu vrijednost. Imajte na umu da je kod što ga generira LFSR maksimalne periodičnosti, Hammingov kod.

Neka je $R(x)$ polinom stupnja $n$ što odgovara registru R. U smislu razmatranja polinoma, možemo kazati da registar generira polinome stupnja $n$–1 ili manjega, a to su uzastopne potencije $x$ mod $R(x)$. Polinomi između $x^0$ mod $R(x)$ i $x^{Pu-1}$ mod $R(x)$ imaju različite oblike, gdje je $x^{Pu}$ mod $R(x) = x^0$ mod $R(x)$. *Operacije u eksponentu, dakle izvode modulo Pu.*

*Tvrdnja 9.4* $(1 + x^{Pu})$ djeljiv je s $R(x)$ bez ostatka.

*Dokaz* $(1 + x^{Pu}) \bmod R(x) = [1 \bmod R(x)] + [x^{Pu} \bmod R(x)]$. Ali prema definiciji, $x^{Pu} \bmod R(x) = 1$. Stoga $(1 + x^{Pu}) \bmod R(x) = 1 + 1 = 0$. Dobivanje ostatka 0 nakon dijeljenja $(1 + x^{Pu})$ s $R(x)$ znači da je $(1 + x^{Pu})$ *djeljiv* s $R(x)$. **Q.E.D.**

### 9.4.2. 9.4.2. ZAŠTO SE KOD ZOVE CIKLIČKI? - SLUŽBENI POSTUPAK

Tvrdnja 4.1 iskazuje da bilo koji kod kojega generira LFSR, a čije su kodne riječi duljine *Pu*, ima cikličko svojstvo. To jest, ciklički pomak kodne riječi također je kodna riječ. Kao što se prije navelo, uobičajen naziv je "ciklički kod" za bilo koji kod stvoren LFSR-generatorom, iako svi ovi kodovi nužno nemaju cikličko svojstvo. Sada dokazujemo ovu tvrdnju koristeći novo-uveden matematički alat.

Korak 1: **Izračun $F(x)$ mod $(1 + x^n)$**.

Razmotrimo što se događa kada računamo vrijednost $F(x)$ mod $(1 + x^n)$, gdje je $F(x)$ polinom stupnja većega od $n$. Na primjer, izračunajmo $(x^5 + x^3 + x^2 + x + 1)$ mod $(x^3 + 1)$. Ovdje trenutno pišemo polinome u obliku u kojemu je najviši eksponent na lijevoj strani polinoma. To je učinjeno kako bi bili u skladu operacijom dijeljenja koja slijedi:

```
1 0 1 1 1 1  :  1 0 0 1  =  1 1
1 0 0 1      ⊕              ↑ ↑
                            x¹ x⁰
  0 1 0 1 1
    1 0 0 1  ⊕
      0 1 0
```

ostatak = $x^1$

Kao što se već učinilo u odjeljku 9.1, dijeljenje s $x^3$+1 znači "klizanje" djelitelja [1001] duž djeljenika [101111], s lijeva u desno. Svaki puta, ako se 1 na lijevoj strani djelitelja podudara s 1 u djeljeniku, obje znamenke se poništavaju, a 1 se dodaje djeljeniku tri mjesta u desno od poništenih 1. Imajte na umu da u konkretnome slučaju, gdje je djelitelj jednak 1001, gornja operacija podudarna je "klizanju"

djeljenika [101111] u dvije polovice, naime, [101] i [111], te ih se zbraja. To jest, [101]⊕[111] = [010].

Jednostavno se može provjeriti da se takva operacija "klizanja" odnosi na sve slučajeve gdje su ne-nulti koeficijenti djelitelja, jedinice s najvišim i najnižim eksponentom. Kada visok eksponent skenira jedan odsječak, nizak eksponent skenira odgovarajuće bitove u desnome odsječku.

Vraćajući se sada na dogovor da se najviši eksponent piše desno, izračun $F(x)$ mod $(1 + x^n)$ može se obilježiti kako slijedi. "Odsječak" binarnoga vektora predstavlja $F(x)$ u dijelovima koji su dugački $n$ bitova, počevši s lijeve strane. Ako duljina vektora nije višekratnik broja $n$, pridružuje se nekoliko 0 desno od vektora. Zatim se ti dijelovi smjeste jedan ispod drugoga i zbroje se. Rezultat je $F(x)$ mod $(1+x^n)$.

Ovaj argument objašnjava primjer. Vektor [1011010111] mod [10001] računa se kako slijedi: Djeljenik se rastavi u dijelove od po 4 bita. Djelitelj je $1 + x^4$. Dobijemo [1011], [0101], [1100]. Dvije 0 pridruže se desno od djeljenika za napraviti desni odsječak da je dugačak 4 bita pa se ta tri dijela zbroje:

| | |
|---|---|
| 1011 | |
| 0101 | |
| 1100 | ⊕ |
| 0010 | |

Ostatak je $x^2$. (uz prethodan dogovor da se najviši eksponent piše desno pa potencije rastu s lijeva u desno).

Korak 2: **Polinomski prikaz dvaju različitih cikličkih pomaka vektora.**

Razmotrimo sada vezu između polinoma $F(x)$ koji odgovara binarnome vektoru **f** duljine $n$ i polinoma $F^i(x)$ koji odgovara vektoru $\mathbf{f}^i$ koji se dobije cikličkim pomakom **f** za $i$ mjesta u desno. Stupanj bilo kojega polinoma nije naveden. On ne može biti veći od $n$–1, ali može biti manji, jer desno mogu postojati krajnje 0 ili u **f** ili u $\mathbf{f}^i$. Povezanost dvaju polinoma pronalazi se na sljedeći način. Pomnoži se $F(x)$ s $x^i$.

To će premjestiti sve koeficijente $F(x)$ u desno (ne ciklički) za $i$ mjesta. Onda je $[x^i \cdot F(x)]$ mod $(1+x^n)$ polinom $F^i(x)$ budući da će, na temelju onoga što smo prije iskazali, operacija modulo dodati $i$ koeficijenata desno od $x^iF(x)$ do $i$ 0 koje su lijevo, što je jednako cikličkome pomaku.

Korak 3: **Zaključak.**

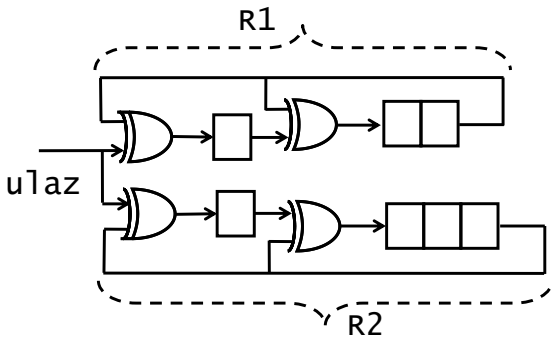
Neka su kodne riječi koda $C$ duljine $Pu$. Sada smo pokazali da ako je **b** kodna riječ u $C$, onda je $\mathbf{b}^i$, koja se dobije iz **b** pomoću $i$ posmika u desno, također kodna riječ u $C$. Neka je $R(x)$ polinom-generator od $C$. Onda je $b(x)$ mod $R(x) = 0$. Polinom podudaran $\mathbf{b}^i$ je $b^i(x) = [x^ib(x)]$ mod $(1+x^{Pu})$, budući da je $Pu$ duljine i $b$ i $b^i$.

Kako bi pokazali da je $\mathbf{b}^i$ kodna riječ, moramo pokazati da je $[x^ib(x)]$ mod $(1+x^{Pu})$ djeljiv s $R(x)$. To će se upravo napraviti. Imajte na umu da ako je $b(x)$ djeljiv s $R(x)$, onda je to i $x^ib(x)$. Iz Tvrdnje 9.4, polinom $(1+x^{Pu})$ također je djeljiv s $R(x)$. Iz toga slijedi da je $[x^ib(x)]$ mod $(1+x^{Pu})$ također djeljiv s $R(x)$. Probajte brojčani primjer: 700 je djeljiv sa 7, 21 je djeljiv sa 7. Onda slijedi da je isto tako, 700 mod 21 djeljiv sa 7. To dovršava nov dokaz Tvrdnje 4.1.

## *9.5. 9.5. Još svojstava cikličkih kodova na temelju rukovanja polinomom*

### 9.5.1. 9.5.1. KOD ČIJI SE DEKODER SINDROMA SASTOJI OD PARALELNOGA SPOJA NEKOLIKO LFSR

Vratimo se sada pitanju koje se postavilo i na koje se odgovorilo u poglavlju 4.4. Dekoder na slici 4.4 i ponovo nacrtan na slici 9.11, stvara dva neovisna sindroma izvan primljene poruke.



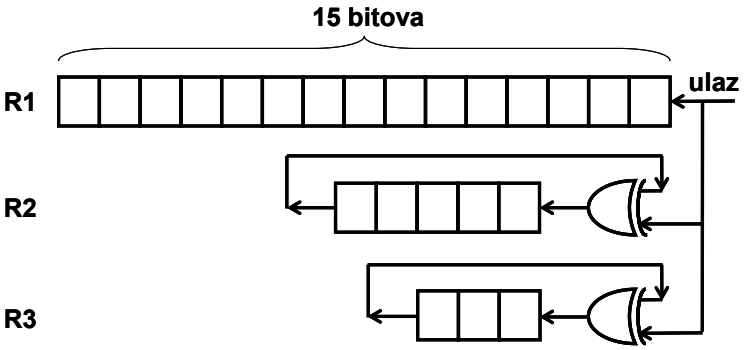

*Slika 9.11: Precrtana slika 4.4*

Problem koji smo imali, bilo je pronaći jedan LFSR koji generira isti kod. U nastavku ćemo pokazati kako se taj problem rješava aritmetikom polinoma. Povratne veze dvaju sindrom-generatora na slici 9.11 odgovaraju polinomima $(1+x+x^3)$ i $(1+x+x^4)$.

Budući da kodna riječ **v** našega koda treba dati 0 za oba sindroma, tražimo da je $M(x)$ djeljiv i s $(1 + x + x^3)$ i $(1 + x + x^4)$. Jedan način za postizanje toga cilja je zahtijevati da polinom-generator $R(x)$ koda bude umnožak dvaju polinoma. To jest, $R(x) = (1 + x + x^3) \cdot (1 + x + x^4) = (1 + x^2 + x^3 + x^5 + x^7)$. (K)ako je $M(x)$ višekratnik $R(x)$, za svaku kontrolnu riječ **v**, gornji oblik $R(x)$ jamči da je $M(x)$ djeljiv s $(1 + x + x^3)$ i $(1 + x + x^4)$.

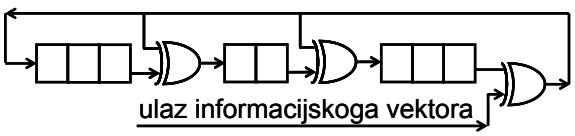
Polinom $R(x)$ *upravo je polinom-generator što odgovara LFSR prikazanome na slici 4.3*. Navodimo postupak prikazan u poglavlju 4.4 za određivanje LFSR koji djeluje kao dva paralelno povezana LFSR. Povratne veze LFSR nalaze se množenjem vektora **v** i matrice $\mathbf{M}^T$, gdje članovi polinoma sadržani u vektoru **v**, predstavljaju povratne veze jednoga registra, a retci od $\mathbf{M}^T$, koji se sastoje od linearnih posmika vektora, predstavljaju povratne veze drugoga registra. Taj *postupak jednak je umnošku dvaju polinoma, a oni odgovaraju dvjema pojedinačnim* LFSR. Naveden ***postupak*** opisuje ***operaciju konvolucije***, koja prema definiciji znači ***množenje dvaju polinoma***.

U nastavku se osvrćemo na specifičan slučaj dekodera na slici 5.5 (slika 9.12).



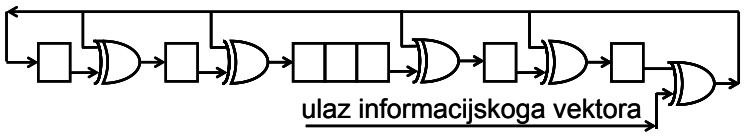

*Slika 9.12: Dva paralelno spojena dekodera sindroma*

Dekoder se sastoji od dvaju pojedinačnih dekodera sindroma spojenih paralelno, a odgovarajući polinomi su $(1 + x^5)$ i $(1 + x^3)$. Dva moguća LFSR prikazana su u poglavlju 5.9, od kojih svaki djeluje kao moguć LFSR-generator koda. Registar na slici 9.13 odgovara polinom-generatoru $(1 + x^3 + x^5 + x^8)$ koji je jednak umnošku dvaju pojedinačnih polinoma, a dobiven je primjenom prije navedenoga postupka.



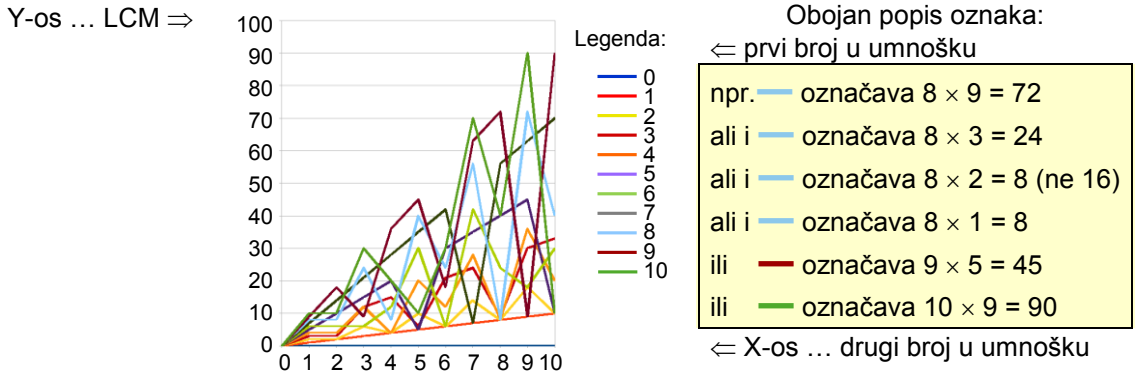

*Slika 9.13: $R(x) = (1 + x^3 + x^5 + x^8)$*

Registar na slici 9.14 odgovara polinomu $(1 + x + x^2 + x^5 + x^6 + x^7)$. Prije se nije objasnilo kako smo dobili ovaj rezultat. To sada radimo.




*Slika 9.14:* $R(x) = (1 + x + x^2 + x^5 + x^6 + x^7)$

Jedini uvjet kojega polinom-generator $R(x)$ našega koda treba zadovoljiti je, djeljivost i s $(1 + x^5)$ i s $(1 + x^3)$. Jedan od načina da se to ostvari, je zadovoljenje uvjeta $R(x) = (1 + x^5)\cdot(1 + x^3)$. Međutim, ovo je samo *dovoljan uvjet* koji omogućuje djeljivost $R(x)$ s $(1 + x^5)$ i $(1 + x^3)$. Zapravo dovoljno je da je $R(x) = \text{LCM}^{132}[(1 + x^5), (1 + x^3)]$, gdje LCM označava *najmanji zajednički višekratnik*.




*Slika 9.15: Najmanji zajednički višekratnik za brojeve od 0 do 10*

Smanjenje stupnja $R(x)$ praktički je vrlo važno, jer taj stupanj određuje broj paritetnih bitova. Smanjenjem toga broja povećava se učinkovitost sheme. Imajte na umu da $(1 + x^5)$ i $(1 + x^3)$ imaju $(x + 1)$, kao zajednički faktor. Polinom $(1 + x^5)\cdot(1 + x^3)/(x + 1) = (1 + x + x^2 + x^5 + x^6 + x^7)$, dalje je djeljiv s $(1 + x^5)$ i s $(1 + x^3)$ pa je prikladan kao polinom-generator koda. Ovo jednostavno promatranje omogućuje uštedjeti jedan paritetan bit i umjesto njega dodati jedan informacijski bit.

Nadalje, naš kod koristi kodne riječi duljine 15 zbog razloga detaljno navedenih u 5. poglavlju. To je vrijednost *Pu* za polinom $(1 + x^5)\cdot(1 + x^3)/(x + 1)$, objašnjavajući zašto je kod, što ga generira ovaj polinom, ciklički. Vrijednost *Pu* za polinom $(1 + x^5)\cdot(1 + x^3)$ je 27. Navedena pravila vrijede i općenito za bilo koji kod za ispravak $t$ praskovitih pogrešaka, vrste opisane u poglavlju 5.8 i poglavlju 5.9, čije su kodne riječi djeljive s $(1 + x^{2t-1})$ i $(1 + x^t)$.

### 9.5.2. 9.5.2. ZNAČAJKE CIKLIČKOGA KODA KAO ŠTO SE VIDI IZ PARITETA BROJA NE NULTIH KOEFICIJENATA U NJIHOVIM POLINOM-GENERATORIMA

U poglavlju 5.9 dokazala se sljedeća tvrdnja: Ako LFSR R1 s neparnim brojem stanja generira *C kod*, onda sve kodne riječi iz *C* imaju jednak broj jedinica. Ova tvrdnja dovela je do Zaključka 5.4 koji naznačuje da je uvijek moguće otkriti postojanje neparnoga broja pogrešaka u kodnoj riječi koda *C*.

LFSR s neparnim brojem stanja koja se pune povratnom vezom, odgovara polinomu koji ima *paran* broj koeficijenata različitih od nule. To je zato, jer stanja što ih puni povratna veza, odgovaraju koeficijentima polinoma, osim koeficijenta najvišega stupnja, a on je uvijek različit od nule. Ovaj zadnji koeficijent predstavlja *izlaznu* vezu krajnjega desnoga stanja. Promatranjem bilo kojega LFSR i njemu odgovarajućega polinoma obrađenoga u ovome poglavlju, to je sasvim razumljivo. Za udžbeničke svrhe, promatranjem polinoma dokazat će se sljedeća Tvrdnja 9.5, (iako rasprava koja prethodi Zaključku 5.4, a koji navodi isti rezultat, lakša je i očitija).

*Tvrdnja 9.5* Kodne riječi, što ih generira polinom $R(x)$, a imaju jednak broj koeficijenata različitih od nule, imaju i jednak broj 1-elemenata.

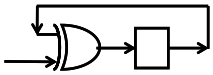
---

[132] LCM … U aritmetici i teoriji brojeva, LCM (*least common multiple*), također se naziva i (*lowest common multiple*) ili (*smallest common multiple*) dvaju brojeva *a* i *b*. Označava se LCM(*a*, *b*), a predstavlja najmanji pozitivan cio broj što je istovremeno djeljiv i s *a* i s *b* (*najmanji zajednički višekratnik*). (vidi: GCD)

*Dokaz* Jedna od osnovnih tvrdnji u algebri kaže da ako je korijen polinoma jednak $R(x)$, onda je $R(x)$ *djeljiv* polinomom $(x–A)$. U našemu slučaju, $A$ može imati samo vrijednost 0 ili 1, a budući da radimo XOR operacijom, "–" se zamjenjuje operacijom "+".

Uzmimo sada polinom $R(x)$, koji ima jednak broj *koeficijenata* različitih od nule. Na primjer, $R(x) = 1 + x^3 + x^5 + x^6$. Zamjenom vrijednosti 1 za $x$, dobijemo $R(1) = 1 + 1 + 1 + 1 = 0$. To jest, 1 je korijen od $R(x)$ pa slijedi da je $R(x)$ djeljiv s $(x + 1)$. Isto vrijedi i za bilo koji $R(x)$ koji ima paran broj koeficijenata različitih od nule. Obrnut argument također je istinit. Naime, ako je polinom djeljiv s $(x + 1)$, onda on ima paran broj koeficijenata različitih od nule. Dokaz slijedi iste crte kao prije.

Nastavljamo dokaz Tvrdnje 9.5, shvaćajući da, ako je $R(x)$ polinom-generator koda, onda je svaka kodna riječ $M(x)$ djeljiva s $R(x)$. Ako $R(x)$ ima paran broj koeficijenata različitih od nule, onda je on djeljiv s $(x + 1)$. Kodna riječ $M(x)$ je, dakle, također djeljiva s $(x + 1)$, jer ima isti broj koeficijenata različitih od nule, što zaokružuju dokaz Tvrdnje 9.5. Poželjno je dokazati i činjenicu da ako je polinom $M(x)$ djeljiv s $(x + 1)$, onda $M(x)$ ima paran broj koeficijenata različitih od nule. S točke motrišta sklopovlja, LFSR što ga prikazuje slika 9.16 odgovara polinomu $(x + 1)$.



*Slika 9.16: LFSR podudaran polinomu* $(x + 1)$

Činjenica da je $M(x)$ djeljiv s $(x + 1)$, znači da ako $M(x)$ napravi posmik u LFSR prikazan na slici 9.16, konačan sadržaj registra bit će 0. S druge strane, jasno je da ovaj registar ne radi ništa drugo već zbraja sve bitove koji ulaze u njega. Drugim riječima, činjenica da je $M(x)$ djeljiv s $(x + 1)$, znači da je zbroj njegovih koeficijenata jednak 0. To jest, $M(x)$ ima *paran broj koeficijenata* različitih od nule.

Sada obrađujemo slučaj gdje polinom-generator koda ima *neparan* broj koeficijenata različitih od nule. Ovaj slučaj manje je važan od onoga prije, ali je vrlo zanimljiv.

*Tvrdnja 9.6* Ciklički kod čiji polinom-generator $R(x)$ ima *neparan* broj koeficijenata različitih od nule, ima svojstvo da je *komplement* kodne riječ, također *kodna riječ*.

***Pojašnjenje primjedbe*** Budući da smo vrlo često nazivali "cikličkim kodom" bilo koji kod što ga generira polinom $R(x)$, trebalo bi navesti da izraz "ciklički kod" ovdje znači da je kod ciklički u punome smislu te riječi. To jest, ciklički pomak kodne riječi, također je kodna riječ. U pogledu Tvrdnje 9.4, to znači da je $(1 + x^n)$ djeljiv s $R(x)$, (gdje je $n$ duljina kodne riječi), a to je svojstvo što će se koristiti u dokazu.

*Dokaz* Neka $\mathbf{i}$ označava vektor "sve 1" čija je duljina $n$ jednaka onoj dužini kodne riječi našega koda. Neka $\bar{\mathbf{v}}$ označava komplement kodne riječi $\mathbf{v}$. Onda imamo da je $\bar{\mathbf{v}} = \mathbf{v} + \mathbf{i}$. Zato što je naš kod linearan, možemo dokazati da je $\bar{\mathbf{v}}$ kodna riječ, pokazujući da je $\mathbf{i}$ kodna riječ. Ovo se dokazuje na sljedeći način. Osnovna jedinstvenost koja se dokazala jednostavnom provjerom, je: $(1 + x^n) = (1 + x)\cdot I(x)$, gdje je $I(x)$ polinom koji odgovara vektoru $\mathbf{i}$. To jest, to je polinom stupnja $n–1$ koji ima ne nulte koeficijente za sve $x^i$, $i = 0, 1, \ldots, n–1$. Na primjer, za $n = 4$, $I(x) = (1 + x + x^2 + x^3)$. Budući da je $(1 + x^n)$ djeljiv s $R(x)$, gdje je $n$ duljina kodne riječi (vidi prethodno razjašnjenje), slijedi da je $(1 + x)\cdot I(x)$ djeljiv s $R(x)$. Kako $(1 + x)$ nije faktor od $R(x)$, jer ima neparan broj koeficijenata, slijedi da je $I(x)$ djeljiv s $R(x)$, tj., $I(x)$ je kodna riječ. Time se dovršio dokaz Tvrdnje 9.6.

Pogledajmo kako se svojstvo da je vektor $\mathbf{i}$ "sve 1", kodna riječ koda, ogleda u smislu paritetne matrice koda $\mathbf{H}^T$. Ako je $\mathbf{i}$ kodna riječ, onda je $\mathbf{i} \cdot \mathbf{H}^T = \mathbf{0}$. Množenjem $\mathbf{H}^T$ s $\mathbf{i}$, ovdje zapravo znači zbroj svih redaka od $\mathbf{H}^T$. Budući da ova operacija daje vektor $\mathbf{0}$, ona znači da *svaki stupac* matrice $\mathbf{H}^T$ *ima paran broj* 1. Možemo zaključiti da ako *polinom-generator cikličkoga koda* ima *neparan broj koeficijenata različitih od nule*, onda *svaki stupac matrice pariteta koda* ima *paran broj* 1-elemenata.

### 9.5.3. 9.5.3. OTKRIVANJE PRASKOVITIH POGREŠAKA

U poglavlju 5.10 i u poglavlju 5.11, razmatrali smo i dokazali sljedeće: Neka je $C$ Hammingov kod kojega stvara registar R duljine $n$. Ovaj LFSR onda ima maksimalnu periodičnost u smislu da ako mu se unese bilo koja ne-nulta početna vrijednost, on se posmiče prolazeći kroz sva moguća $2^n–1$ stanja različita od nule, prije povratka na početnu vrijednost. Neka je $\mathbf{v}$ kodna riječ u $C$ i neka $\mathbf{V}_j^B$ označava

vektor dobiven uvođenjem u **v** praskovite pogreške duljine $n$ čiji je uzorak $B$, gdje prasak započinje od mjesta $j$ računajući s lijeva (prvo mjesto je 0#).

Ako je praskovit uzorak $B$ nekako poznat, onda se primjenjuje sljedeći postupak za određivanje $j$ izvan $\mathbf{v}_j^B$. Nakon unosa $\mathbf{v}_j^B$ u R, posmik se koristi sve dok god R sadrži uzorak praska. Ako se dodatno učini $k$ posmika, onda je $j = 2^n–1–k$. Također smo naveli da ovo opažanje vrijedi za ispravak jednostruke pogreške Hammingova koda, a "praskovit" uzorak ima oblik [1000 … 0]. Sve ovo prethodno navedeno, sada tumačimo terminologijom[133] polinoma.

1. *Izražavanje* $\mathbf{v}_j^B$ *u obliku polinoma:* $B(x)$ je polinom koji predstavlja prasak $B$. Množenjem $B(x)$ s $x^i$ pomiče uzorak praska na $j$-to mjesto duž **v**, a zbroj $M(x) + x^j \cdot B(x)$ umeće pogreške u **v**. Onda imamo $\mathbf{v}_j^B(x) = M(x) + x^j \cdot B(x)$.
2. *Uvođenje* $\mathbf{v}_j^B$ u R*:* Sadržaj R, nakon učitavanja $\mathbf{v}_j^B$ u njega, je $\mathbf{v}_j^B(x) \bmod R(x)$.
3. *Otkrivanje j, na temelju k:* Posmik u R, nakon što on već sadrži $\mathbf{v}_j^B(x) \bmod R(x)$, znači uzastopna množenja $x$. Sadržaj registra nakon $k$ posmika, počevši od sadržaja $\mathbf{v}_j^B(x) \bmod R(x)$, je onda $[x^k \cdot \mathbf{v}_j^B(x)] \bmod R(x)$. Tvrdnja o određivanju $j$, na temelju poznavanja $k$, znači sljedeće:

*Tvrdnja* 9.7 Ako postoji $k$ takav da je $[x^k \cdot \mathbf{v}_j^B(x)] \bmod R(x) = B(x)$, onda je $j = 2^n–1–k$.

Ova Tvrdnja već se dokazala u 5. poglavlju. Sada će se opet vrlo jednostavno dokazati, pomoću aritmetike polinoma.

*Dokaz* Uočimo da je: $[\mathbf{v}_j^B(x)] \bmod R(x) = [M(x) + \mathbf{v}_j^B(x)] \bmod R(x)$

Iz toga slijedi da je:

$$[x^k \cdot \mathbf{v}_j^B(x)] \bmod R(x) = [x^k \cdot M(x) + x^{j+k} \cdot B(x)] \bmod R(x) =$$

$$= [x^k \cdot M(x)] \bmod R(x) + [x^{j+k} \cdot B(x)] \bmod R(x).$$

Očito je $[x^k \cdot M(x)] \bmod R(x) = 0$, jer je **v** kodna riječ. Onda imamo da je:

$$[x^k \cdot \mathbf{v}_j^B(x)] \bmod R(x) = [x^{j+k} \cdot B(x)] \bmod R(x).$$

Da bi se dokazala Tvrdnja 9.7, moramo pokazati da ako postoji $k$ takav da je $[x^{j+k} \cdot B(x)] \bmod R(x) = B(x)$, onda je $j = 2^n–1–k$. Ova zadnja tvrdnja očito je istinita, zbog maksimalne periodičnosti R. Ako R nema maksimalnu periodičnost, možemo pronaći vrijednost $m < 2^n–1$, takvu da je $j+k = m$ i još uvijek vrijedi $[x^{j+k} \cdot B(x)] \bmod R(x) = B(x)$. Time je dovršen dokaz Tvrdnje 9.7.

Treba objasniti da gornji dokaz uopće ne koristiti činjenicu kako R ima najveću periodičnost. Tvrdnja 9.7 i njezin dokaz, mogu se ponoviti riječ po riječ za bilo koji općenit slučaj u kojem se vrijednost $2^n–1$ zamjenjuje s $Pu$. Uočite, međutim, da Tvrdnja 9.7 počinje riječima "Ako postoji $k$ …". Nema jamstva da za bilo koji $B(x)$ stupnja $n–1$ ili manjega, postoji $k$ takav da je $[x^k \cdot B(x)] \bmod R(x) = B(x)]$, osim za $Pu = 2^n–1$. To je zbog toga što, samo R maksimalne periodičnosti prolazi kroz sva moguća stanja $B(x)$.

[133] *terminologija* … (lat. *terminus*) granica, međa, (lat. *logos*) riječ, govor; umjetan govor, skup umjetno stvorenih (stručnih) izraza (termina) nekoga znanstvenoga područja

## 21. *Laboratorijska vježba 20: Konvolucija (množenje dvaju polinoma) i 2 pojedinačna LFSR*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

## 9.6. 9.6. Jasno izraženo postupanje RS kodom

***Uvodna napomena*** RS kod za ispravak jednoga znaka definirao se u poglavlju 5 kao kod čije kodne riječi su blokovi od $2^n$–1 znakova, gdje je svaki znak duljine $n$. Broj informacijskih znakova je $2^n$–3. Dva paritetna znaka omogućuju ispravak pogreške jednoga znaka koja nastane u prijenosu. Motrišta aritmetike polinoma za ovaj kod, dana su u ovome poglavlju. Važnost ovdje uvedenih matematičkih alata nije dostatna za posebne primjene RS kodova, kao temelj za korištenje širokoga raspona kodova.

### 9.6.1. 9.6.1. KONAČNO POLJE GF(Q)

**Polje** je skup elemenata koje određuju dvije operacije nad tim elementima obilježene kao (+) i (·). Skup je *zatvoren* u smislu da, ako se izvodi bilo koja od ovih operacija između dvaju elementa u skupu, rezultat je također sadržan u skupu elemenata. Među elementima skupa trebala bi postojati dva elementa, označena 0 i 1, koja imaju svojstvo da za svaki element $x$, u skupu postoje dva elementa označena $(-x)$ i $x^{-1}$, gdje je $x+(-x) = 0$, a $x \cdot x^{-1} = 1$. Ove operacije nisu nužno one koje poznajemo iz pučke škole.

**Konačno polje** je polje s konačnim brojem elemenata. Elementi *konačnih polja GF(q)* sastoje se od brojeva 0, 1, …, $q$–1 za prost $q$. Operacije (+) i (·) obično su operacije zbrajanja i množenja, a izvode se kao modulo $q$.

*Primjer* Elementi $GF(5)$ su brojevi (0, 1, 2, 3, 4), gdje je 1+2 = 3, 3+4 = 2, 2·4 = 3. Element (–1) je element 4 dok je 1+4 = 0. Element $2^{-1}$ jednak je 3 dok je 2·3 = 1.

Svaki $GF(q)$ ima **primitivan element** takav da je svaki element polja osim elementa 0, izražen nekom potencijom $\alpha$ mod $q$. Također, $\alpha^{q-1} = 1$. To jest, operacija u eksponentu $\alpha$, izvodi modulo $q$–1. Ovo posljednje svojstvo vrijedi za svaki element polja. Bez prevelikoga ulaska u teoriju brojeva, možemo istaknuti da postojanje $\alpha$ slijedi iz svojstva prostoga broja $q$. Napomenimo s druge strane, da postojanjem recipročne vrijednosti za množenje $x^{-1}$ bilo kojega elemenata $x$ iz $GF(q)$, a koji je dio definicije polja, osigurano je postojanjem $\alpha$. To je zato što postoji $k$ takav da je $\alpha^k = x$. Onda imamo da je $x^{-1} = \alpha^{-k} = \alpha^{q-k-1}$. Naravno, sve operacije učinjene su po modulu $q$.

Zanimljivo je polje $GF(2)$ (2 je prost broj). Ovo polje ima samo dva elementa 0 i 1. Po definiciji, zbroj se radi operacijom modulo-2. To jest, 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 0. Napomena: *Ova vrsta zbrajanja upravo je XOR operacija.*

### 9.6.2. 9.6.2. POLJE GF(Q)

*Definicija* $GF(2^n)$ konačno je polje čiji su elementi skup binarnih koeficijenata od $2^n$ polinoma stupnja $n$–1 ili manjega. Dva elementa $GF(2^n)$ polja zbrajaju se na sličan način kao što se zbrajaju kodne riječi cikličkoga koda (one su također polinomi). Kao i broj $q$, u slučaju $GF(q)$, moramo za polje $GF(2^n)$ definirati izraz, **modulo**, kojime se u polju obavljaju operacije množenja. Ovaj izraz u našemu slučaju je polinom stupnja $n$.

*Definicija* Polinom $G(x)$ stupnja $n$, na kojemu se izvode operacije polja $GF(2^n)$, zove se **polinom-generator polja**.

Svaki polinom stupnja $n$, ne može biti polinom-generator u $GF(2^n)$. Treba imati svojstvo jednako prostome broju za $q$. Kao što smo prije raspravili, ovo svojstvo osigurava postojanje primitivnih elemenata $\alpha$ čije potencije generiraju sve elemente polja, osim elementa "sve 0".

Što se tiče našega polinom-generatora $G(x)$, želimo imati polinom $\alpha$ stupnja $n$–1 ili manjega, tako, da će uzastopne potencije $\alpha$, ako se primijene $\alpha$ mod $G(\alpha)$, generirati sve polinome stupnja $n$–1 ili manjega, osim za polinom "sve 0". Zadnja tvrdnja odnosi se na potrebno svojstvo $G(x)$ podsjećajući nas na LFSR maksimalne periodičnosti i odgovarajući polinom $R(x)$.

Ponovno promotrite Tvrdnju 9.2 i Tablicu 9.1 nakon nje. Vidimo da sve potencije od $x$ između 0 i 6 daju različite rezultate, ako se upotrijebe modulo operacije između $\alpha$ i polinoma $R(x)$, gdje je $R(x) = 1+x+x^3$. Ovaj $R(x)$ može biti polinom-generator od $GF(2^3)$. Da bi razjasnili gornji argument, generirajmo polinom stupnja 2 ili manjega, potencijama izraza $\alpha$ mod $G(\alpha)$, gdje je $G(\alpha) = 1+\alpha+\alpha^3$, kako je navedeno u tablici 9.2.

*Tablica 9.2 Potencije:* $\alpha \bmod (1+\alpha+\alpha^3)$ *i* $\alpha \bmod (1+\alpha^2+\alpha^4)$

| Potencije od α | α mod $(1+\alpha+\alpha^3)$ | binarno | Potencije od α | α mod $(1+\alpha^2+\alpha^4)$ | binarno |
|---|---|---|---|---|---|
| | (a) | (b) | | (c) | (d) |
| $\alpha^0 =$ | 1 | = [100] | $\alpha^0 =$ | 1 | = [1000] |
| $\alpha^1 =$ | α | = [010] | $\alpha^1 =$ | α | = [0100] |
| $\alpha^2 =$ | $\alpha^2$ | = [001] | $\alpha^2 =$ | $\alpha^2$ | = [0010] |
| $\alpha^3 =$ | $1+\alpha$ | = [110] | $\alpha^3 =$ | $\alpha^3$ | = [0001] |
| $\alpha^4 =$ | $\alpha+\alpha^2$ | = [011] | $\alpha^4 =$ | $1+\alpha^2$ | = [1010] |
| $\alpha^5 =$ | $1+\alpha+\alpha^2$ | = [111] | $\alpha^5 =$ | $\alpha+\alpha^3$ | = [0101] |
| $\alpha^6 =$ | $1+\alpha^2$ | = [101] | $\alpha^6 =$ | 1 | = [1000] |
| $\alpha^7 =$ | 1 | = [100] | $\alpha^7 =$ | α | = [0100] |
| itd. | … | … | itd. | … | … |

U stupcu (a) u tablici 9.2 popisane su uzastopne potencije od α i izračuni α mod $G(\alpha)$, gdje je $G(\alpha) = 1+\alpha+\alpha^3$ i gdje se $\alpha^3$ uvijek zamjenjuje s $1+\alpha$. Jasno se vidi da potencije od α generiraju sve elemente polja $GF(2^3)$, osim za polinom "sve 0". To znači da $G(x) = 1 + x + x^3$ može poslužiti kao polinom-generator $GF(2^3)$. Stupac (c) u tablici 9.2 popisuje potencije od α i izračune α mod $G(\alpha)$, gdje je $G(\alpha) = 1 + \alpha^2 + \alpha^4$ i gdje se $\alpha^4$ stalno zamjenjuje s $1 + \alpha^2$. Primjećuje se da je $\alpha^6 = \alpha^0$ pa polinom $G(\alpha) = 1+\alpha^2+\alpha^4$, ne može generirati sve elemente polja $GF(2^4)$. Stupac (b), skraćenim načinom izražavanja polinoma u stupcu (a), prikazuje α gdje se polinom predstavlja binarnim vektorom koji se sastoji od koeficijenata polinoma. Stupac (d), također skraćenim načinom izražavanja polinoma u stupcu (c), prikazuje α, gdje se polinom predstavlja binarnim vektorom koji se sastoji od koeficijenata polinoma. Imajte na umu da su stupci (a) i (b) zapravo ponavljanje tablice 9.1, koristeći drugi način iskaza.

Element polja α je *polinom* pa se svi elementi u polju izražavaju kao polinomi u α. U mnogim slučajevima, umjesto izražavanja određenoga elementa polja *y*, kao polinoma u α, radije imenujemo *y* nakon potenciranja α koji stvara *y* (u slučaju ako je ta potencija poznata). Kao primjer uzmimo $GF(2^3)$ kojega stvara $G(x) = 1 + x + x^3$. Element polja $(1 + \alpha + \alpha^2)$ može se imenovati kao $\alpha^5$.

Također treba uočiti da potencije α mod $G(\alpha)$, znači da je α korijen $G(x)$, u smislu da je $G(\alpha) = 0$. Ova tvrdnja razumljiva je sama po sebi, jer operacija α mod $G(\alpha)$ znači da je $G(\alpha) = 0$. Međutim, u svrhu objašnjenja, detaljnije ćemo dokazati ovu tvrdnju. Izrazimo $G(x)$ kao $F(x) + x^n$. Funkcija $F(x)$ je stupnja $n$–1 ili manjega. Njezini koeficijenti sastoje se od $n$–1 najmanje značajnih koeficijenata od $G(x)$.

Imajte na umu da se izraz $x^n$ pojavljuje u $G(x)$, jer znamo da je njegov stupanj $n$. Na primjer, $1 + x + x^3 = F(x) + x^3$ gdje je $F(x) = 1 + x$, samo onda je $G(\alpha) = F(\alpha) + \alpha^n$. Činjenica da se α prikazuje potencijama α mod $G(\alpha)$ znači da se zamjenjuje s $F(\alpha)$. Drugim riječima, $\alpha^n = F(\alpha)$. Npr., pri stvaranju stupca (a) tablice 9.2, $\alpha^3 = 1 + \alpha$. Ako je $a^n = F(\alpha)$, onda je $G(\alpha) = a^n + F(\alpha) = 0$, što dokazuje gore navedenu tvrdnju.

*Definicija* Polinom $G(x)$ stupnja $n$, korijena α, naziva se primitivan polinom ako je $\alpha^i$ (kada se predstavlja kao polinom u α), stupnja $n$–1 ili manjega, različiti za $0 < i < 2^n$–2.

Prema definiciji slijedi da je primitivan polinom stupnja $n$, polinom-generator $GF(2^n)$. Polinom stupnja $n$ ima $n$ korijena (nisu svi nužno drugačiji). Uputno je osim α, istaknuti i druge korijene polinom-generatora $GF(2^n)$. To je učinjeno u Tvrdnji 9.8.

*Tvrdnja 9.8* Ako je α korijen primitivnoga polinoma $G(x)$, onda su ostali korijeni od $G(x)$ $\alpha^{2^i}$, $i = 1, 2, \ldots, n$–1.

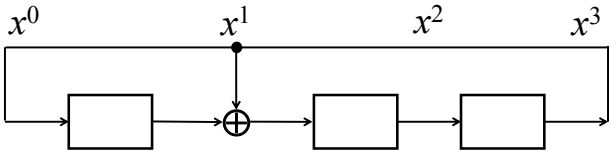
*Dokaz* Kvadrat polinomskoga izraza, jednak je zbroju kvadrata njegovih članova plus udvostručen iznos svih mogućih umnožaka dvaju članova. Na primjer, $(a+bx+cx^2)^2 = a^2 + b^2x^2 + c^2x^4 + 2(abx + acx^2 + bcx^3)$. Ako su koeficijenti $a$, $b$, $c$ binarni onda da je $a^2 = a$, $b^2 = b$, $c^2 = c$. Operacijom zbrajanja, XOR, ako se isti izraz pojavljuje paran broj (2) puta, onda se uvijek dobije 0 pa imamo da je $(a+bx+cx^2)^2 = a+bx^2+cx^4$.

Iz toga slijedi da je u slučajevima obrađenima u ovome poglavlju, $[G(\alpha)]^2 = G(\alpha^2)$. Ako je $\alpha$ korijen od $G(x)$, onda je $G(\alpha) = 0$, a $[G(\alpha)]^2 = 0$. Onda je kako slijedi $G(\alpha^2) = 0$, tj., $\alpha^2$ je korijen od $G(x)$. Ako je $\alpha^2$ korijen od $G(x)$, onda je i njegov kvadrat $\alpha^4$ korijen od $G(x)$ pa slijedi da su $\alpha^{2^i}$, $i = 1, 2, \ldots, n-1$, svi korijeni od $G(x)$. Indeks(iranje) $i$ zaustavlja se kod $n$–1, dok se na eksponentu od $\alpha$ izvode operacije modulo-$(2^n-1)$, što znači da je $\alpha^{2^n} = \alpha$. Tablica 9.3 prikazuje primitivne polinome s minimalnim brojem XOR vrata.

*Tablica 9.3: Primitivni polinomi s minimalnim brojem XOR vrata*

| Stupanj ($n$) | Polinom | | Stupanj ($n$) | Polinom | |
|---|---|---|---|---|---|
| 1 | $x+1$ | √ | 13, 24 | $x^n + x^4 + x^3 + x + 1$ | |
| 2, 3, 4, 6, 7, 15, 22 | $x^n + x + 1$ | √ | 14 | $x^n + x^{12} + x^{11} + x + 1$ | |
| 5, 11, 21, 29 | $x^n + x^2 + 1$ | √ | 16 | $x^n + x^5 + x^3 + x^2 + 1$ | √ |
| 8, 19 | $x^n + x^6 + x^5 + x + 1$ | | 18 | $x^n + x^{12} + 1$ | |
| 9 | $x^n + x^4 + 1$ | | 23 | $x^n + x^5 + 1$ | |
| 10, 17, 20, 25, 28 | $x^n + x^3 + 1$ | | 26, 27 | $x^n + x^8 + x^{12} + x + 1$ | |
| 12 | $x^n + x^{12} + x^4 + x^3 + 1$ | | 30 | $x^n + x^{16} + x^{15} + x + 1$ | |

Za praktično i brzo ispitati je li binaran polinom primitivan, potrebno je sljedeće (slika 9.17):[134]

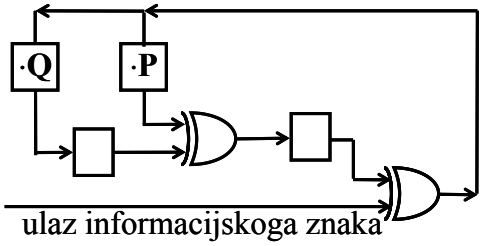



*Slika 9.17: Brzo ispitivanje je li binarni polinom primitivan*

U krug posmičnoga registara s povratnim vezama koje su povezane potencijama polinoma, uvede se bilo koje ne-nulto stanje te se napravi desni posmik. Ako krug generira svaki, i sve ne-nulte elemente polja unutar jednoga perioda ($2^n$), onda je polinom kojega definira ovo $GF(2^n)$ polje, primitivan polinom.[135]

### 9.6.3. 9.6.3. UOBIČAJENO POSTUPANJE RS KODOM KOJI ISPRAVLJA JEDAN ZNAK

Na temelju iskustva koje smo do sada stekli, možemo razumjeti povezanost između polinoma i kodnih riječi cikličkih kodova. Naš RS kod je ciklički. Njega generira LFSR (prikazan na slici 5.15) i postoji njemu pridružen polinom-generator (slika 9.17).




*Slika 9.18 RS koder za cjelovit ispravak jednoga znaka* (*precrtana slika 5.15*)

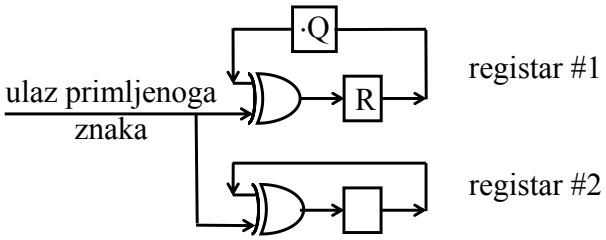
*Definicija* **Kodna riječ nekoga RS koda** je polinom čiji su koeficijenti elementi iz $GF(2^n)$.

*Primjer* Uzmimo poruku **m** = [100, 001, 011, 101, 110, 111, 010], što se obradila u poglavlju 5.14.1. Znakovi ovdje, smatraju se elementima iz $GF(2^3)$. Oni se pojavljuju u stupcu (b) tablice 9.2, gdje stupac (a) popisuje njima pridružene potencije od $\alpha$. Onda imamo: $M(x) = (\alpha^0\ \alpha^2 x\ \alpha^4 x^2\ \alpha^6 x^3\ \alpha^3 x^4\ \alpha^5 x^5\ \alpha^2 x^6)$. U slučaju ako je jedan od znakova nula, ne bi bilo pridruženih potencija $\alpha$, znak bi ostao koeficijent vrijednosti 0. RS kod zapravo je nastavak ideje cikličkih kodova što smo ju već koristili.

[134] Za vježbu, sklopom na slici 9.17, ispitati gornju tablicu i sve polinome u njoj.
[135] C++ zadatak i vježba!

Dok su kodne riječi običnih cikličkih kodova imale koeficijente iz $GF(2)$, RS kod ima koeficijente iz $GF(2^n)$, za $n > 1$. Pridruživanje između polinoma i LFSR, ovdje se izvodi na isti način kao što je učinjeno za $GF(2)$, gdje se koeficijenti polinoma određuju prema povratnoj vezi LFSR. Međutim, u našemu proširenome slučaju, povratne veze LFSR također nose ne-binarnu težinu, a ta težina trebala bi se pojaviti u koeficijentima odgovarajućega polinoma.

*Primjer* Uzmimo registre sindrom-generatora na slici 5.18, koji se precrtao u sliku 9.19.




*Slika 9.19: Ponovno nacrtana slika 5.18*

Prisjećajući se svojstva matrice **Q**, napominjemo da, ako je znak koji se vraća u registar #1, element polja $\alpha^i$, onda nakon množenja s **Q**, njegova vrijednost postaje $\alpha^i+1$. To jest, množenjem s **Q** jednako je množenjem s $\alpha$ pa povratna veza ima "težinu" $\alpha$. Onda je polinom pridružen registru #1 jednak $(\alpha + x)$, a polinom pridružen registru #2 je $(1 + x)$.

Kodnu riječ našega RS koda karakterizira činjenica da je djeljiva i s $(\alpha + x)$ i s $(1 + x)$. Ovi polinomi su relativno prosti, a polinom-generator našega koda je, dakle, $F(x) = (\alpha + x)\cdot(1 + x) = \alpha + (1 + \alpha)x + x^2$. To je upravo pripadni polinom registra što se prikazao na slici 5.15, gdje je operacija matrice **P** jednaka umnošku elementa polja $\alpha^i$ (tj. povratnoj vezi iz krajnjega desnoga stanja) i $(1 + \alpha)$. Sve kodne riječi našega koda onda su djeljive s $F(x) = \alpha+(1+\alpha)x + x^2$.

Sada razmotrimo učinak *pogreške jednoga znaka* umetnutoga u kod*nu riječ*. U slučaju Hammingova koda, učinak jedne pogreške na mjestu *#i* zbrajanjem polinoma $1\cdot x^i$ i kodne riječi, gdje umjetno postavljena jedinica (1), naznačuje "uzorak" pogreške. Ovaj uzorak je iz $GF(2)$. Za RS kod, uzorak pogreške znaka je element $\alpha^i$ iz $GF(2^n)$ (za znakove duljine $n$). Umetanje ovoga uzorka pogreške na mjesto *#j* u kodnoj riječi (znakovi se računaju s lijeva, gdje se mjesto sasvim lijevo označava #0), znači zbrajanje polinoma $\alpha^i\cdot x^i$ i polinoma što predstavlja kodnu riječ.

*Primjer* Uzmimo slučaj poruke **m'** iz poglavlja 5.14. Ona se dobila umetanjem uzorka pogreške znaka [001], na mjesto #4 u kodnoj riječi. Ako se naš RS kod temelji na polju $GF(2^3)$, uvedenome u stupcima (a) i (b) u tablici 9.2, onda je $M'(x) = M(x)\alpha^2x^4$.

Kako bi razjasnili matematičko tumačenje *posmika primljene poruke u sindrom-generator* na slici 5.18, koristimo poseban slučaj **m'** prikazan gore. Posmik **m'** u registar #1 znači dijeljenje s $(\alpha+x)$. Ako je **m** kodna riječ, onda je $\mathbf{m'}(x) \bmod (\alpha+x) = \alpha^2x^4 \bmod (\alpha+x)$. Pogledajmo detaljno dijeljenje $\alpha^2x^4$ s $(x+\alpha)$. Započnimo dijeljenjem $\alpha^2x^4$ s $x$. *Prvi izraz količnika* bit će onda $\alpha^2x^3$. Zatim množimo $\alpha^2x^3$ s $(x+\alpha)$ pa dobijemo $\alpha^2x^4+\alpha^3x^3$. To se zbroji s $\alpha^2x^4$ i postupak se nastavlja. Cijela operacija prikazana je u nastavku (zapisi su objašnjeni na kraju).

| | | | |
|---|---|---|---|
| $*\alpha^2x^4$ | | : $(x+\alpha)$ = | $\alpha^2x^3 + \alpha^3x^2 + \alpha^4x + \alpha^5$ |
| $\alpha^2x^4$ | $+\alpha^3x^3$ | + | |
| $*\alpha^3x^3$ | | | |
| $\alpha^3x^3$ | $+\alpha^4x^2$ | + | |
| $*\alpha^4x^2$ | | | |
| $\alpha^4x^2$ | $+\alpha^5x$ | + | |
| $*\alpha^5x$ | | | |
| $\alpha^5x$ | $+\alpha^6$ | + | |
| | $*\alpha^6$ | ← ostatak | |

Promatrajući prve rezultate označene *, dolazimo do zaključka: Ako *dijelimo* izraz $\alpha^ix^i$ s $(x+\alpha)$, slijedimo uzastopne korake: *smanjujemo potenciju* od $x$ za 1 i *povećavamo potenciju* od $\alpha$ za 1. Zbroj potencije $x$ odnosno $\alpha$, nakon svakoga koraka je dakle, konstantan $i + j = 6$. Postupak *završava* kada se

dobije *ostatak* $\alpha^{i+j}(\alpha^6)$. Ovdje zapravo imamo dva brojača koja rade paralelno. Jedan od njih je *uzlazni brojač*, a drugi je *silazni brojač*. Za poseban slučaj naše poruke **m'** imamo **m'**($x$) mod ($\alpha+x$) = $\alpha^6$.

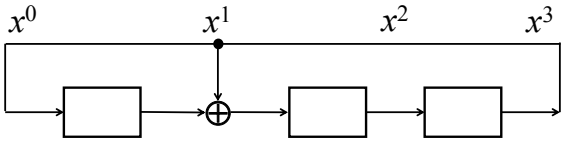
Posmik **m'** u registar #2 znači diobu s ($1+x$). Ako je **m** kodna riječ, onda je $M'(x)$ mod ($1+x$) = $\alpha^2 x^4$ mod ($1+x$). Jednostavno je dokazati da dijeljenje $\alpha^i x^j$ s ($1+x$) znači *uzastopno sniženje potencije* od $x$, dok ne dobijemo ostatak $\alpha^i$, a to je *uzorak pogreške znaka.* U našemu slučaju, ostatak je $\alpha^2$. Nakon dobivanja $\alpha^{i+j}$ i $\alpha^i$ od dva sindrom-generatora, obnovimo vrijednost $j$ (tj., položaj pogrešnoga znaka), kao što se opisalo u poglavlju 5.14.1.

## 22. *Laboratorijska vježba 21: Brzo ispitivanje je li polinom primitivan*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

*Tablica 9.3: Primitivni polinomi s minimalnim brojem XOR vrata*

| Stupanj ($n$) | Polinom |
|---|---|
| 2, 3, 4, 6, 7, 15, 22 | $x^n + x + 1$ |
| 5, 11, 21, 29 | $x^n + x^2 + 1$ |
| 8, 19 | $x^n + x^6 + x^5 + x + 1$ |
| 9 | $x^n + x^4 + 1$ |
| 10, 17, 20, 25, 28 | $x^n + x^3 + 1$ |
| 12 | $x^n + x^{12} + x^4 + x^3 + 1$ |
| 13, 24 | $x^n + x^4 + x^3 + x + 1$ |
| 14 | $x^n + x^{12} + x^{11} + x + 1$ |
| 16 | $x^n + x^5 + x^3 + x^2 + 1$ |
| 18 | $x^n + x^{12} + 1$ |
| 23 | $x^n + x^5 + 1$ |
| 26, 27 | $x^n + x^8 + x^{12} + x + 1$ |
| 30 | $x^n + x^{16} + x^{15} + x + 1$ |




$R(x) = 1 \oplus x \oplus x^3$ (ako se polinom dijeli)

*Slika 9.16: Brzo ispitivanje je li binarni polinom primitivan*

Za praktično i brzo ispitati je li binarni polinom *primitivan*, potrebno je sljedeće (slika 9.16). Sklopom na slici 9.16, ispitati sve polinome u tablici 9.3. Brojevi označavaju potenciju od $x$. (1 u polinomu znači $x^1$)

| *m* | Stanja spojena na zbrajalo po modulu-2 | *m* | Stanja spojena na zbrajalo po modulu-2 | *m* | Stanja spojena na zbrajalo po modulu-2 |
|---|---|---|---|---|---|
| 2 | (1, 2) (0x3) | 13 | (1, 9, 11, 13), (1, 8, 11, 12, 13), (1, 3, 4, 13) (0x1FFF) | 24 | (~~1, 18, 23, 24~~), (1, 2, 7, 24), (1, 3, 4, 24) (0xFFFFFF) |
| 3 | (1, 3) (0x7) | 14 | (1, 6, 10, 14), (1, 2, 12, 13, 14), (1, 3, 5, 14), (1, 11, 12, 14) (0x3FFF) | 25 | (~~1, 23~~), (1, 3, 25) (0x1FFFFFF) |
| 4 | (1, 4) (0xF) | 15 | (1, 15), (1, 14, 15) (0x7FFF) | 26 | (1, 21, 25, 26), (1, 2, 6, 26) (0x3FFFFFF) |
| 5 | (1, 2, 5) (0x1F) | 16 | (~~1, 5, 14, 16~~), (1, 11, 13, 14, 16), (1, 3, 12, 16) (1, 2, 3, 5, 16) (0xFFFF) | 27 | (~~1, 23, 26, 27~~), (1, 2, 5, 27) (0x7FFFFFF) |
| 6 | (1, 6) (0x3F) | 17 | (~~1, 15~~), (1, 14, 17), (1, 3, 17) (0x1FFFF) | 28 | (~~1, 26~~), (1, 3, 28) (0xFFFFFFF) |
| 7 | (1, 7), (1, 3, 7) (0x7F) | 18 | (~~1, 12~~), (1, 11, 18), (1, 7, 18), (1, 2, 5, 18) (0x3FFFF) | 29 | (~~1, 28~~), (1, 2, 29) (0x1FFFFFFF) |
| 8 | (1, 5, 6, 7), (1, 2, 7, 8), (1, 2, 3, 4, 8) (0xFF) | 19 | (~~1, 15, 18, 19~~), (1, 14, 17, 18, 19), (1, 2, 5, 19), (1, 5, 6, 19) (0x7FFFF) | 30 | (1, 8, 29, 30), (1, 4, 6, 30) (0x3FFFFFFF) |
| 9 | (1, 4, 9) (0x1FF) | 20 | (~~1, 18~~), (1, 3, 20) (0xFFFFF) | 31 | (~~1, 29~~), (1, 3, 31) (0x7FFFFFFF) |
| 10 | (1, 3, 10) (0x3FF) | 21 | (~~1, 20~~), (1, 2, 21) (0x1FFFFF) | 32 | (1, 11, 31, 32), (1, 2, 3, 5, 7, 32) (0xFFFFFFFF) |
| 11 | (1, 2, 11) (0x7FF) | 22 | (1, 22) (0x3FFFFF) | 33 | (~~1, 21~~), (1, 4, 6, 33) |
| 12 | (1, 4, 6, 12) | 23 | (~~1, 19~~), (1, 18, 23), (1, 5, 23) (0x7FFFFF) | 34 | (1, 8, 33, 34), (1, 2, 5, 6, 7, 34) (0x1FFFFFFF) |

Napomena: Polinomi sa zelenom podlogom su provjereni, a s ljubičastom nisu ispravni! Neprovjereni su 30, 31, 32, 33, 34!

# 23. *Laboratorijska vježba 22: Kodiranje konvolucijskih kodova impulsnim odzivom*

Napomena: Cjelovit opis nalazi se na *"Sustavu za podršku nastavi"*

# 10. POGLAVLJE 10 – DODACI

## *10.1. 10.1. Umnošci vektora i matrice*

### 10.1.1. 10.1.1. UMNOŽAK MATRICE A I MATRICE B

Definira se za slučaj da je *broj stupaca prve matrice* jednak *broju redaka druge matrice*. Rezultat je matrica **C**, [**A**($m$, $p$) × **B**($p$, $n$) = **C**($m$, $n$)].

Neka je, dakle:

$$\mathbf{A} = [a_{ik}] \qquad (m, p)\text{-matrica},$$
$$\mathbf{B} = [b_{ik}] \qquad (p, n)\text{-matrica}.$$

Onda je njihov umnožak

$$\mathbf{C} = [c_{ik}] \qquad (m, n)\text{-matrica}$$

kojoj su elementi $c_{ik}$ određeni s:

$$c_{ik} = \sum_{s=1}^{p} a_{is} b_{sk} \tag{10.1}$$

($i = 1, 2, \ldots, m$), ($k = 1, 2, \ldots, n$).

$$\mathbf{A}(m,p) \bullet \mathbf{B}(p,n) = \mathbf{C}(m,n)$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1s} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2s} & \cdots & a_{2p} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{is} & \cdots & a_{ip} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{ms} & \cdots & a_{mp} \end{bmatrix} \bullet \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1k} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2k} & \cdots & b_{2n} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ b_{s1} & b_{s2} & \cdots & b_{sk} & \cdots & b_{sn} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{p1} & b_{p2} & \cdots & b_{pk} & \cdots & b_{pn} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1k} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2k} & \cdots & c_{2n} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ c_{i1} & c_{i2} & \cdots & c_{ik} & \cdots & c_{in} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mk} & \cdots & c_{mn} \end{bmatrix}$$

gdje je:

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + \ldots + a_{1s}b_{s1} + \ldots + a_{1p}b_{p1} = \sum_{s=1}^{p} a_{1s}b_{s1}$$

$$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$

### 10.1.2. 10.1.2. UMNOŽAK VEKTORA $A^T$ I VEKTORA B

Definira se za slučaj kada je *broj stupaca prvoga vektora* jednak *broju redaka drugoga vektora*. Rezultat je vektor **c**, [$\mathbf{a}^T(1, n)$ × **b**($n$, 1) = **c**(1, $n$)]

$$\mathbf{a}^T \cdot \mathbf{b} = \overset{(1,n)}{\begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}^T} \cdot \overset{(n,1)}{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}} = \overset{(1,1)}{\left[a_1 \cdot b_1 + a_2 \cdot b_2 + \cdots + a_n \cdot b_n\right]}. \tag{10.2}$$

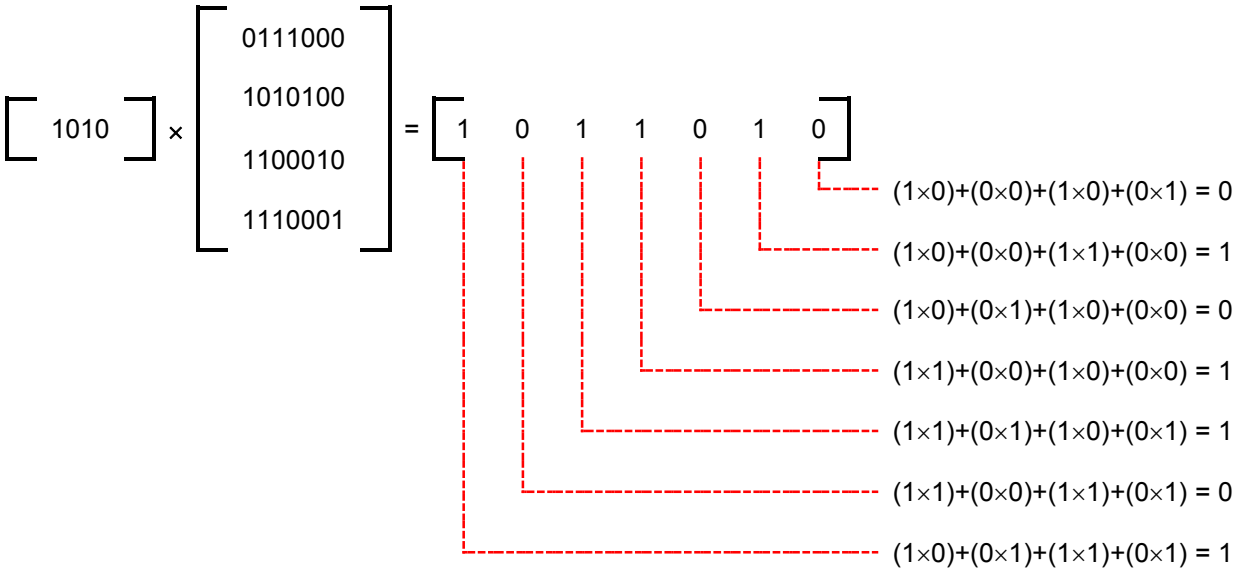
### 10.1.3. 10.1.3. UMNOŽAK VEKTORA A I MATRICE B

Definira se za slučaj kada je *broj stupaca vektora* jednak *broju redaka matrice*. Rezultat je vektor **c**, [**a**(1, $n$) × **B**($n$, $m$) = **c**(1, $m$) = **c** $^T$($m$, 1)] (npr. [**a**(1, 7) × **B**(7, 3) = **c**(1, 3) = **c** $^T$(3, 1)]).

$$\mathbf{a}\cdot\mathbf{B}=\begin{bmatrix}a_1 & \overset{(1,7)}{a_2} & \cdots & a_7\end{bmatrix}\cdot\overset{(7,3)}{\begin{bmatrix}b_{11} & b_{12} & b_{13}\\ b_{21} & b_{22} & b_{23}\\ b_{31} & b_{32} & b_{33}\\ b_{41} & b_{42} & b_{43}\\ b_{51} & b_{52} & b_{53}\\ b_{61} & b_{62} & b_{63}\\ b_{71} & b_{72} & b_{73}\end{bmatrix}}= \tag{10.3}$$

$$=\begin{bmatrix}a_1\cdot b_{11}+a_2\cdot b_{21}+\cdots+a_7\cdot b_{71} & \overset{(1,3)}{a_1\cdot b_{12}+a_2\cdot b_{22}+\cdots+a_7\cdot b_{72}} & a_1\cdot b_{13}+a_2\cdot b_{23}+\cdots+a_7\cdot b_{73}\end{bmatrix}=$$

$$=\overset{(3,1)}{\begin{bmatrix}a_1\cdot b_{11}+a_2\cdot b_{21}+\cdots+a_7\cdot b_{71}\\ a_1\cdot b_{12}+a_2\cdot b_{22}+\cdots+a_7\cdot b_{72}\\ a_1\cdot b_{13}+a_2\cdot b_{23}+\cdots+a_7\cdot b_{73}\end{bmatrix}^T}.$$




### 10.1.4. 10.1.4. UMNOŽAK MATRICE A I VEKTORA B

Definira se za slučaj kada je *broj stupaca matrice* jednak *broju redaka vektora*. Rezultat je vektor **c**, [**A**(*n*, *m*) × **b**(*m*, 1) = **c**(*n*, 1)] (npr. [**A**(3, 7) × **b**(7, 1) = **c**(3, 1)]).

$$\mathbf{A}\cdot\mathbf{b}=\overset{(3,7)}{\begin{bmatrix}a_{11} & a_{12} & \cdots & a_{17}\\ a_{21} & a_{22} & \cdots & a_{27}\\ a_{31} & a_{32} & \cdots & a_{37}\end{bmatrix}}\cdot\overset{(7,1)}{\begin{bmatrix}b_1\\ b_2\\ \vdots\\ b_7\end{bmatrix}}=\overset{(3,1)}{\begin{bmatrix}a_{11}\cdot b_1+a_{12}\cdot b_2+\cdots+a_{17}\cdot b_7\\ a_{21}\cdot b_1+a_{22}\cdot b_2+\cdots+a_{27}\cdot b_7\\ a_{31}\cdot b_1+a_{32}\cdot b_2+\cdots+a_{37}\cdot b_7\end{bmatrix}} \tag{10.4}$$

ili za naš slučaj množimo matricu $\mathbf{H}^T$ i stupčani vektor **r**:

$$\mathbf{s}=\mathbf{H}^T\cdot\mathbf{r}\Rightarrow[3\ retka\times 7\ stupaca]\cdot[7\ redaka\times 1\ stupac]=[3\ retka\times 1\ stupac] \tag{10.5}$$

Rezultat je stupčani vektor **s** (sindrom) koji ima strukturu [3 *retka* × 1 *stupac*]. Obično ga pišemo u transponiranome obliku kao redni vektor **s** = ($\sum_1$ $\sum_2$ $\sum_3$) gdje su zbrojevi ($\sum$) binarne znamenke 0 ili 1.

## 10.2. 10.2. Kompleksije

### 10.2.1. 10.2.1. FAKTORJELE

#### 10.2.1.1. 10.2.1.1. Definicija

Funkcija faktorjele (*factorial function*) najčešće se definira kao:

$$n! = \prod_{k=1}^{n} k \ \forall n \in \aleph, 0! = 1 \tag{10.6}$$

Broj $n!$ ("$n$ faktorjela") pozitivan je cio broj koji nastaje množenjem nekoga broja svim prethodnim pozitivnim cijelim brojevima.

$$n! = 1 \times 2 \times 3 \times \ldots \times (n-2) \times (n-1) \times n$$

Prema definiciji, 0! (nula faktorjela) jednako je 1

$$0! = 1;\ 1! = 1$$

$$n! = (n-1)! \times n \ (n > 0)$$

$$n! = 1 \times 2 \times 3 \times \ldots \times n$$

ili rekurzijski

$$n! = \begin{cases} 1 & \text{ako je } n = 0 \\ (n-1)! \times n & \text{ako je } n > 0 \end{cases} \quad \text{prema dogovoru: } 0! = 1 \tag{10.7}$$

Zbog dogovora, 0! = 1, a dogovor osigurava:

1. da rekurzijska relacija $(n+1)! = n! \times (n+1)$, vrijedi i za $n = 0$;
2. dozvoljava jednostavno pisanje izraza za beskonačne polinome, npr. $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$;
3. u kombinatorici, ova relacija vrijedi za sve nulte veličine;
4. broj kombinacija ili permutacija praznoga skupa jednostavno je broj 1.

#### 10.2.1.2. 10.2.1.2. Primjer 10.1

Napišite funkciju za izračun faktorjele broja, te u glavnome programu pozovite tu funkciju i izračunajte

$$\binom{n}{k} = \frac{n!}{k!*(n-k)!} \text{ za } k = 2 \text{ i } n = 7.$$

*14. Faktorjele*

```
#include <iostream>
using namespace std;
double faktorjela(int a){
if(a>1)return(a*faktorjela(a-1));
else return(1);
}
int main(){
int n(7),k(2);
double
c=faktorjela(n)/(faktorjela(k)*faktorjela(n-
k));
cout<<n<<" nad "<<k<<" je: "<<c<<"\n";
return 0;
}
```

```
C:\windows\system32\cmd.exe
7 nad 2 je: 21
Press any key to continue . . .
```

*10.2.1.3.* *10.2.1.3. Dvostruka faktorjela n!!*

Oznaka $n!!$ označava u matematici dvostruku faktorjelu, a odnosi se na faktorjelu parnih odnosno neparnih brojeva.

$$n!! = \begin{cases} 1 & \text{za } n = 0 \text{ ili } n = 1 \\ (n-2)!! & \text{za } n \geq 2 \end{cases} \qquad 0!! = 1 \tag{10.8}$$

*10.2.1.4.* *10.2.1.4. Primjer parne faktorjele*

$$10!! = 2 \times 4 \times 6 \times 8 \times 10 = 3840$$

*15. Parne faktorjele*

```
#include <iostream>
using namespace std;
double parna(int a){
if(a>1)return(a*parna(a-2));
else return(1);
}

int main(){
int paran;
paran:
cout<<"Upisi paran broj: ";
cin>>paran; //broj mora biti paran
if(paran%2!=0)goto paran;
cout<<paran<<"!!(parne faktorjele) =
"<<parna(paran)<<"\n";
return 0;
}
```

```
C:\windows\system32\cmd.exe
Upisi paran broj: 3
Upisi paran broj: 9
Upisi paran broj: 10
10!!(parne faktorjele) = 3840
Press any key to continue . . .
```

*10.2.1.5.* *10.2.1.5. Primjer neparne faktorjele*

$$9!! = 1 \times 3 \times 5 \times 7 \times 9 = 945$$

*16. Neparne faktorjele*

```
#include <iostream>
using namespace std;
double neparna(int a){
if(a>1)return(a*neparna(a-2));
else return(1);
}

int main(){
int neparan;
neparan:
cout<<"Upisi neparan broj: ";
cin>>neparan; //broj mora biti neparan
if(neparan%2==0)goto neparan;
cout<<neparan<<"!!(neparne faktorjele) =
"<<neparna(neparan)<<"\n";
return 0;
}
```

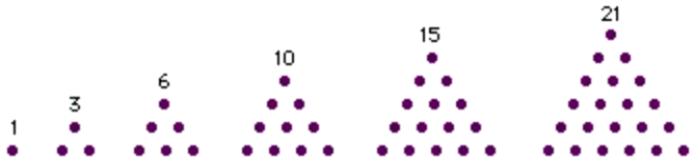
```
C:\windows\system32\cmd.exe
Upisi neparan broj: 10
Upisi neparan broj: 8
Upisi neparan broj: 9
9!!(neparne faktorjele) = 945
Press any key to continue . . .
```

### 10.2.2. 10.2.2. OSTALI POJMOVI

Alternacijska faktorjela, Digama funkcija, Eksponencijska faktorjela, Faktorjelski prirodan broj, Faktorium, Stirlingova aproksimacija, faktorjela Trailingove nule, ….

#### *10.2.2.1. 10.2.2.1. "Trokutni" brojevi (*triangular numbers*)*

Istostranični trokuti čije su stranice sastavljene od točaka. Niz počinje 1 točkom, a svaki sljedeći trokut, u stranicama ima 1 točku više. Niz se sastoji od brojeva (zbroj točaka) {1, 3, 6, 10, 15, 21, …}.




#### *10.2.2.2. 10.2.2.2. Analogna faktorjela zbroja (operator je upitnik "?" umjesto uskličnika "!")*

Kao što se u kombinatorici definirao operator "!" za *množenje* niza brojeva, na sličan način definirao se operator "?" ("*terminal*" *function*) za *zbroj* niza brojeva (autor: *Donald Ervin Knuth*, *The Art of Computer Programming, 3th. ed., 1997*, 1.2.5. *Permutation and Factorials*, page. 78, (Eq. 9 and 10), *Stanford University*)[136].

#### *10.2.2.3. 10.2.2.3. Binomni koeficijenti*

$\binom{n}{k}$… čita se ($n$ nad $k$ ili, $n$ povrh $k$ ili, $n$ iznad $k$), $n \in \aleph, 0 \le k \le n$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \binom{n}{n-k} = \frac{n!}{(n-k)!k!} = \frac{n\cdot(n-1)\cdot(n-2)\cdot\ldots\cdot(n-k+1)}{1\cdot 2\cdot 3\cdot\ldots\cdot(k-1)\cdot k} \qquad (10.9)$$

#### *10.2.2.4. 10.2.2.4. Svojstva binomnih koeficijenata*

$$\binom{n}{0}=1,\ \binom{n}{n}=1,\ \binom{n}{k}=\binom{n}{n-k},\ \binom{n}{1}=n,\ \binom{n}{n-1}=n,\ \binom{n}{k}+\binom{n}{k+1}=\binom{n+1}{k+1},$$

$$\binom{n}{k}=\binom{n+1}{k+1}\cdot\frac{k+1}{n+1},\ \binom{n}{k}=\binom{n-1}{k-1}+\binom{n-1}{k},\ \binom{n}{0}+\binom{n}{1}+\binom{n}{2}+\binom{n}{3}+\ldots+\binom{n}{n-1}+\binom{n}{n}=2^n$$

$$\binom{n}{0}-\binom{n}{1}+\binom{n}{2}-\binom{n}{3}+\ldots+(-1)^n\cdot\binom{n}{n}=0,\ \binom{n}{0}+\binom{n}{2}+\binom{n}{4}+\ldots=2^{n-1},\ \binom{n}{1}+\binom{n}{3}+\binom{n}{5}+\ldots=2^{n-1}$$

#### *10.2.2.5. 10.2.2.5. Primjer 10.2*

*17. Binomni koeficijenti*

| | C:\windows\system32\cmd.exe |
|---|---|
| | |

### 10.2.3. 10.2.3. PERMUTACIJE

#### *10.2.3.1. 10.2.3.1. Permutacije bez ponavljanja*

- Elementi: $a$, $b$, $c$ ($n = 3$):

a b c  b a c  c a b

[136] "*termial*" *progression* … konvergentan zbroj aritmetičkoga niza (*termial arithmetic progression sum*): $n? = 1 + 2 + \ldots + n = \sum_{1\le n\le k} k = (\frac{1}{2})n(n+1)$ pa ova formula vrijedi i za realne i za racionalne brojeve (npr. $(\frac{1}{2})? = \frac{3}{8}$).

a c b  b c a  c b a

$$P_n = n! = \tag{10.10}$$

$$= 3! = 1 \cdot 2 \cdot 3 = 6$$

*10.2.3.2.* *10.2.3.2. Primjer 10.3*

*18. Permutacije bez ponavljanja*

```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;

void main(){
int b;
do{
cout<<"Permutacije bez ponavljanja!\nBroj
elemenata = ";
cin>>b;
vector<int> m_v(b);
for(unsigned
i=0;i<m_v.size();i++)m_v.at(i)=i+1;
//punjenje
sort(m_v.begin(),m_v.begin()+b);
for(unsigned
i=0;i<m_v.size();i++)cout<<m_v.at(i)<<' ';
//ispis prve crte
cout<<endl;
while(next_permutation(m_v.begin(),
m_v.begin()+b)){
for(unsigned
i=0;i<m_v.size();i++)cout<<m_v.at(i)<<' ';
//ostatak ispisa
cout<<endl;}
}while(b!=0);
}
```

```
C:\windows\system32\cmd.exe
Permutacije bez ponavljanja!
Broj elemenata = 3
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
Permutacije bez ponavljanja!
Broj elemenata = 0

Press any key to continue . . .
```

*10.2.3.3.* *10.2.3.3. Permutacije s ponavljanjem*

Neka je zadano $n$-elemenata od kojih su $\alpha$ ($\leq n$) međusobno jednaki:

- Elementi: $a$, $a$, $a$, $b$, $c$. Permutacije od 5 elemenata ($n = 5$) od kojih se 1 element $a$ ponavlja 3 puta.

$$P_n^{'\alpha} = \frac{n!}{\alpha!} = \tag{10.11}$$

$$= \frac{5!}{3!} = \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5}{1 \cdot 2 \cdot 3} = 20$$

Ako među $n$-elemenata ima $\alpha_a$-jednakih, $\alpha_b$-jednakih, …, $\alpha_n$-jednakih:

$$P_n^{'\alpha_1,\alpha_2,\ldots,\alpha_n} = \frac{n!}{\alpha_1!\alpha_2!\ldots\alpha_n!} \tag{10.12}$$

Elementi: $a$, $a$, $b$, $b$, $b$, $c$, $c$: Permutacije od 7 elemenata ($n = 7$) od kojih se ponavljaju:

$\alpha_a = 2$ puta

$\alpha_b = 3$ puta

$\alpha_c = 2$ puta

| | |
|---|---|
| aa bbb cc | aa bcb bc |
| aa bbc bc | aa bcb cb |
| aa bbc cb | aa bcc bb |
| ..... | ..... |

$$P_n^{'\alpha_1,\alpha_2,\alpha_3} = \frac{7!}{2!3!2!} = \frac{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6\cdot 7}{1\cdot 2\cdot 1\cdot 2\cdot 3\cdot 1\cdot 2} = 210$$

*10.2.3.4. 10.2.3.4. Primjer 10.4*

## *19. Permutacije s ponavljanjem*

**Cjelovit skup C++ zadataka i rješenja za sve komleksije, nalazi se u datoteci: Programirane kompleksije.doc**

C:\windows\system32\cmd.exe

### 10.2.4. 10.2.4. KOMBINACIJE

*10.2.4.1. 10.2.4.1. Kombinacije bez ponavljanja*

Kombinacije od $n$-elemenata ($n = 5$) $r$-toga stupnja ($r = 3$) (skupine od $r$-članova).

- Elementi: $a$, $b$, $c$, $d$, $e$:

| | | | | |
|---|---|---|---|---|
| a b c | a b d | a b e | a c d | a c e |
| a d e | b c d | b c e | b d e | c d e |

$$K_n^r = \binom{n}{r} = \frac{n!}{r!(n-r)!} = \binom{n}{n-r} = \tag{10.13}$$

$$= \frac{5!}{3!(5-3)!} = \frac{1\cdot 2\cdot 3\cdot 4\cdot 5}{1\cdot 2\cdot 3\cdot 2!} = \frac{4\cdot 5}{1\cdot 2} = \frac{20}{2} = 10$$

*10.2.4.2. 10.2.4.2. Primjer 10.5*

## *20. Kombinacije bez ponavljanja*

**Cjelovit skup C++ zadataka i rješenja za sve komleksije, nalazi se u datoteci: Programirane kompleksije.doc,**
**Skup C++ zadataka za kombinacije nalazi se u: Rekurzija-analiza-komb.doc i Kombinacije s ponavljanjem n=5, r=3.doc**

C:\windows\system32\cmd.exe

*10.2.4.3. 10.2.4.3. Kombinacije s ponavljanjem*

To su kombinacije u kojima se pojedini elementi ponavljaju.

- Elementi: 1, 2, 3, 4. Kombinacije trećega razreda ($r = 3$) od 4 elementa ($n = 4$):

| | | | | |
|---|---|---|---|---|
| 1 1 1 | 1 2 2 | 1 3 4 | 2 2 4 | 3 3 3 |
| 1 1 2 | 1 2 3 | 1 4 4 | 2 3 3 | 3 3 4 |
| 1 1 3 | 1 2 4 | 2 2 2 | 2 3 4 | 3 4 4 |
| 1 1 4 | 1 3 3 | 2 2 3 | 2 4 4 | 4 4 4 |

$$K_n^{'r} = \binom{n+r-1}{r} = K_{n+r-1}^r = \frac{(n+r-1)!}{r!(n-1)!} = \tag{10.14}$$

$$= \frac{(4+3-1)!}{3!(4-1)!} = \frac{6!}{3!\cdot 3!} = \frac{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6}{1\cdot 2\cdot 3\cdot 1\cdot 2\cdot 3} = 20$$

### 21. Kombinacije s ponavljanjem

**Cjelovit skup C++ zadataka i rješenja za sve komleksije, nalazi se u datoteci:**
**Programirane kompleksije.doc**

C:\windows\system32\cmd.exe

## 10.2.5. 10.2.5. VARIJACIJE

### 10.2.5.1. 10.2.5.1. Varijacije bez ponavljanja

Varijacije od *n*-elemenata *r*-toga stupnja (skupine od *r*-članova). Varijacije su kombinacije bez ponavljanja čiji se elementi međusobno permutiraju.

- Elementi: *a*, *b*, *c*, *d*, *e* ($n = 5$, $r = 3$):

| | | | | |
|---|---|---|---|---|
| **a b c** | **a b e** | **a c e** | **b c d** | **b d e** |
| a c b | a e b | a e c | b d c | b e d |
| b a c | b a e | c a e | c b d | d b e |
| b c a | b e a | c e a | c d b | d e b |
| c a b | e a b | e a c | d b c | e b d |
| c b a | e b a | e c a | d c b | e d b |
| **a b d** | **a c d** | **a d e** | **b c e** | **c d e** |
| a d b | a d c | a e d | b e c | c e d |
| b a d | c a d | d a e | c b e | d c e |
| b d a | c d a | d e a | c e b | d e c |
| d a b | d a c | e a d | e b c | e c d |
| d b a | d c a | e d a | e c b | e d c |

$$V_n^r = r!\,K_n^r = \frac{n!}{(n-r)!} = r!\binom{n}{r} = \tag{10.15}$$

$$= \frac{5!}{(5-3)!} = \frac{1\cdot 2\cdot 3\cdot 4\cdot 5}{1\cdot 2} = 60$$

$$V_n^r = (n-r+1)\,V_n^{r-1} \qquad \binom{n}{0} = 1$$

$$V_n^1 = n$$

### 10.2.5.2. 10.2.5.2. Primjer 10.7

### 22. Varijacije bez ponavljanja

**Cjelovit skup C++ zadataka i rješenja za sve komleksije, nalazi se u datoteci:**
**Programirane kompleksije.doc**

C:\windows\system32\cmd.exe

### 10.2.5.3. 10.2.5.3. Varijacije s ponavljanjem

Varijacije od *n*-elemenata *r*-toga razreda (znači na sve moguće načine svrstati tih *n*-elemenata u skupine od po *r*-članova time da skupine mogu sadržavati i jednake članove.

- Elementi: *a*, *b*, *c*, *d*, *e*. Varijacije od 5 elemenata ($n = 5$) 2. razreda ($r = 2$) s ponavljanjem:

| | | | | |
|---|---|---|---|---|
| a a | b a | c a | d a | e a |
| a b | b b | c b | d b | e b |
| a c | b c | c c | d c | e c |
| a d | b d | c d | d d | e d |
| a e | b e | c e | d e | e e |

$$V_n'^r = n^r = n \cdot V_n'^{r-1} = \tag{10.16}$$

$$= 5^2 = 25.$$

Elementi: 0, 1. Varijacije od 2 elementa ($n = 2$) 7. razreda ($r = 7$) s ponavljanjem:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0000000 | 0010000 | 0100000 | 0110000 | 1000000 | 1010000 | 1100000 | 1110000 |
| 0000001 | 0010001 | 0100001 | 0110001 | 1000001 | 1010001 | 1100001 | 1110001 |
| 0000010 | 0010010 | 0100010 | 0110010 | 1000010 | 1010010 | 1100010 | 1110010 |
| 0000011 | 0010011 | 0100011 | 0110011 | 1000011 | 1010011 | 1100011 | 1110011 |
| 0000100 | 0010100 | 0100100 | 0110100 | 1000100 | 1010100 | 1100100 | 1110100 |
| 0000101 | 0010101 | 0100101 | 0110101 | 1000101 | 1010101 | 1100101 | 1110101 |
| 0000110 | 0010110 | 0100110 | 0110110 | 1000110 | 1010110 | 1100110 | 1110110 |
| 0000111 | 0010111 | 0100111 | 0110111 | 1000111 | 1010111 | 1100111 | 1110111 |
| 0001000 | 0011000 | 0101000 | 0111000 | 1001000 | 1011000 | 1101000 | 1111000 |
| 0001001 | 0011001 | 0101001 | 0111001 | 1001001 | 1011001 | 1101001 | 1111001 |
| 0001010 | 0011010 | 0101010 | 0111010 | 1001010 | 1011010 | 1101010 | 1111010 |
| 0001011 | 0011011 | 0101011 | 0111011 | 1001011 | 1011011 | 1101011 | 1111011 |
| 0001100 | 0011100 | 0101100 | 0111100 | 1001100 | 1011100 | 1101100 | 1111100 |
| 0001101 | 0011101 | 0101101 | 0111101 | 1001101 | 1011101 | 1101101 | 1111101 |
| 0001110 | 0011110 | 0101110 | 0111110 | 1001110 | 1011110 | 1101110 | 1111110 |
| 0001111 | 0011111 | 0101111 | 0111111 | 1001111 | 1011111 | 1101111 | 1111111 |

*10.2.5.4. 10.2.5.4. Primjer 10.8*

Varijacije s ponavljanjem od 2 (binarna) elementa 7. razreda, znači na sve moguće načine svrstati ta 2 elementa (0, 1) u skupine od po 7 članova [vektore] time da te skupine mogu sadržavati i *jednake* članove. Ukupan broj varijacija s ponavljanjem računa se formulom:

$$V_n'^r = n^r = n \cdot V_n'^{r-1} = 2^7 = 128$$

*Zadatak* Izračunajte ukupan broj kodnih vektora dužine 7 napravljenih od 2 binarna elementa? Ispišite na predočniku sve kodne vektore u osam stupaca (jednakoga broja redaka) i odgovarajući broj retka. Izračunajte i ispišite broj redaka. Vektore ispišite tako da nakon punjenja jednoga stupca vektorima, niz se nastavlja na vrhu sljedećega stupca. Na primjer, za varijacije s ponavljanjem od 2 elementa 3. razreda, ispis niza [000, 001, 010, 011, 100, 101, 110, 111] u 2 stupca izgledao bi kao u donjoj tablici pa isto načelo primijenite i u traženome primjeru.

| | |
|---|---|
| 000 | 100 |
| 001 | 101 |
| 010 | 110 |
| 011 | 111 |

*23. Varijacije s ponavljanjem od 2 elementa 7 razreda*

```
#include<iostream>
#include<cmath>
using namespace std;
void dec2bin(int dekadski,int vektori[]);
//dekadski broj<16 u binarni (4 znamenke)
void ispis();
void main(){
int dekadski(0);
int vektori[128];
for(int i=1;i<=8;i++){//8 binarnih znamenaka
for(int j=0;j<16;j++){
dekadski=(i-1)*16+j;//odbrojavanje od 0 do 127
dec2bin(dekadski,vektori);
//getchar();
}}
}
void dec2bin(int dekadski,int vektori[]){
cout<<dekadski<<" = ";
```

```
C:\windows\system32\cmd.exe
```

*23. Varijacije s ponavljanjem od 2 elemenata 7 razreda*

```
int g;
for(int i=6;i>=0;i--){
g=dekadski%2;
dekadski=dekadski/2;
vektori[i]=g;
}
for(int j=0;j<7;j++)cout<<vektori[j];
//cout<<endl;
getchar();
}
```

## 10.3. 10.3. Umnošci vektora

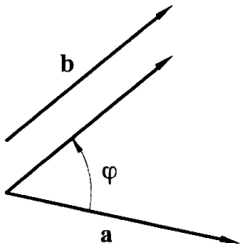
### 10.3.1. 10.3.1. SKALARNI UMNOŽAK

Definira se samo za 2 vektora i kut φ između njih:

$$\vec{a} \cdot \vec{b} := |\vec{a}| \cdot |\vec{b}| \cos \varphi \,. \tag{10.17}$$

Time se definira preslik "·" iz umnoška **v** × **v** u polje realnih brojeva (skalara). Odatle i ime ovomu umnošku.

**Kut među vektorima**. Kut među vektorima je *manji* kut (po apsolutnome iznosu) od dvaju kutova koji zatvaraju zadana dva vektora translatirana[137] u zajednički početak (slika 10.1).




*Slika 10.1: Kut između dvaju vektora*

Označit ćemo ga kao $\varphi = \angle(\vec{a}, \vec{b})$. Prema tomu, kut može poprimiti vrijednost $-\pi < \varphi \leq \pi$.

### 10.3.2. 10.3.2. VEKTORSKI UMNOŽAK

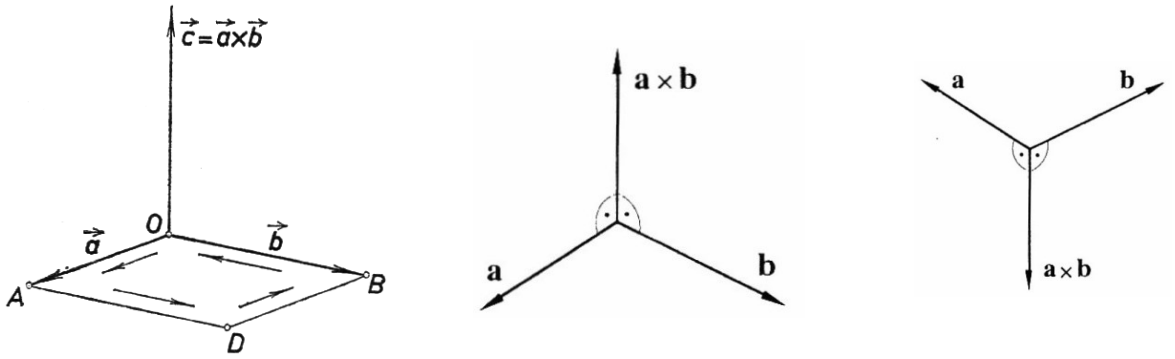
Za zadane *ne-kolinearne* vektore $\vec{a}$ i $\vec{b}$, definiramo *vektorski* (ili *vanjski*) umnožak (*product*):

$$\vec{a} \times \vec{b} \,.$$

To je vektor sljedećih svojstava:

- $|\vec{a} \times \vec{b}| = |\vec{a}|\,|\vec{b}|\,|\sin\varphi|$
- $\vec{a} \times \vec{b}$ je okomit na vektor $\vec{a}$ i na vektor $\vec{b}$.
- Trojka $(\vec{a}, \vec{b}, \vec{a} \times \vec{b})$ čini desni sustav (slika 10.2).

[137] *translacija* … (lat. *translatio*) 1. prijenos, prenošenje, premještanje; 2. prijevod, prevođenje (na drugi jezik); 3. prav. prenošenje, prijenos (nekog prava); 4. fiz. gibanje jednoga tijela kod kojega sve njegove točke opisuju paralelne, podjednake, isto usmjerene putove




*Slika 10.2: Trojka $\vec{a}$, $\vec{b}$, $\vec{a} \times \vec{b}$ čini desni sustav*

Apsolutna vrijednost vektorskoga umnoška $\vec{a} \times \vec{b}$ jednaka je površini paralelograma što ga zatvaraju ta dva vektora. Vektorski umnožak definira se:

$$\vec{a} \times \vec{b} = a \cdot b \sin\varphi \cdot \vec{n}_0, \tag{10.18}$$

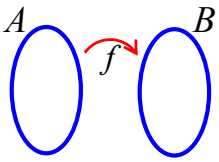
gdje je $\vec{n}_0$ jedinični vektor (zbog naznake da je dobiven umnožak također vektor). Vektor dobiven vektorskim umnoškom 2 ne-kolinearna vektora ($\alpha > 0$), okomit je na ravninu koju oni tvore i čini desni sustav.

## *10.4. 10.4. Funkcijska preslikavanja*

### 10.4.1. 10.4.1. FUNKCIJE

Uvijek kada imamo dva ne-prazna skupa $A$ i $B$, među kojima je uspostavljena takva veza da se svakomu $x \in A$ pridružuje po jedan $y \in B$, kažemo da je zadana funkcija $f$ na $A$ s vrijednostima u $B$, ili da $f$ djeluje s $A$ u $B$.

***Definicija:*** Neka su $A$ i $B$ dva ne-prazna skupa. Funkcijom na skupu $A$ s vrijednostima u skupu $B$, nazivamo svaki postupak kojime se na određen način, svakome elementu skupa $A$ pridružuje *jedan i samo jedan* element skupa $B$ (slika 10.3).




*Slika 10.3: Funkcija*

Da je $f$ funkcija iz $A$ u $B$, označavamo ovako:

$$f : A \to B \text{ ili } A \xrightarrow{f} B$$

a čitamo: "$f$ je funkcija iz $A$ u $B$".

Definicija funkcije uključuje, uređenu trojku:

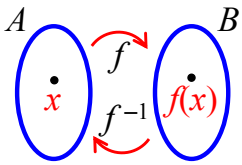
1. Ulazni skup $A$ koji se zove i *domena* ili *područje definicije* funkcije $f$.
2. Izlazni skup $B$, koji se zove i *kodomena* ili *područje vrijednosti* funkcije $f$.
3. Postupak $f$ (radnja, propis) kojime se svakome elementu $x$ iz $A$ pridružuje jedinstven element $y = f(x)$ iz $B$.

Ako se elementu $x$, $x \in A$, propisom $f$, pridružuje element $y$, $y \in B$, pišemo:

$$x \mapsto y \text{ ili } f(x) = y \text{ ili } (x, f(x)).$$

### 10.4.2. 10.4.2. INVERZNA FUNKCIJA

Neka su $A$ i $B$ ne-prazni skupovi, a $f$ bijekcija od $A$ na $B$. Funkciju $f^{-1}$ koja preslikava $B$ u $A$, za koju vrijedi $f^{-1}(y) = x$ onda i samo onda ako je $f(x) = y$, tj. koja svakom $y \in B$ pridružuje onaj element $x = A$ koji se funkcijom $f$ preslikava u $y$, zovemo *inverzna funkcija* funkcije $f$ (slika 10.4).




*Slika 10.4: Inverzna funkcija*

Za kompoziciju funkcija $f^{-1}$ i $f$ vrijedi:

$$f^{-1} \circ f = i_A, \qquad f \circ f^{-1} = i_B$$

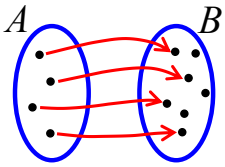
gdje je $i_A$ jedinični element (*identity*) na skupu $A$, a $i_B$ jedinični je element na skupu $B$ (znak $\circ$ označuje kompoziciju dviju funkcija).

*Napomena*: Funkcija $i_R$ zove se *identiteta* ili *jedinični preslik* na skupu realnih brojeva $\Re$ ako je:

$$i_R: \Re \to \Re,\ i_R(r) = r \text{ za svaki } r \in \Re.$$

### 10.4.3. 10.4.3. INJEKCIJSKA FUNKCIJA ILI INJEKCIJA

Neka su $A$ i $B$ skupovi, a $f$ funkcija iz $A$ u $B$. Ako je propis pridruživanja $f$ između elemenata skupa $A$ i skupa $B$ takav, da različitim elementima skupa $A$ pridružuje različite elemente skupa $B$, funkcija $f$ zove se *injekcijska* funkcija ili *injekcija*, dakle iz $x \neq x' \Rightarrow f(x) \neq f(x')$. Uočite da kodomena može (ali ne mora) sadržavati nepridružene elemente (slika 10.5).

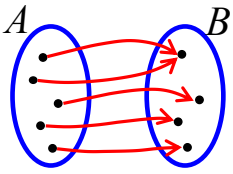



*Slika 10.5: Injekcijska funkcija*

*Napomena.* Znak $\Rightarrow$ čitamo: *slijedi* ili *povlači* (*implicira*).

### 10.4.4. 10.4.4. SURJEKCIJSKA FUNKCIJA ILI SURJEKCIJA

Neka su $A$ i $B$ skupovi, a $f$ funkcija iz $A$ u $B$. Ako je propis pridruživanja $f$ takav da za svaki element $y$ iz skupa $B$ postoji element $x$ iz skupa $A$ takav da bude $f(x) = y$, funkcija $f$ zove se *surjekcijska funkcija* ili *surjekcija* (slika 10.6).

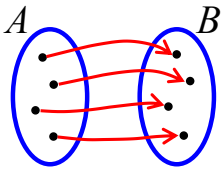



*Slika 10.6: Surjekcijska funkcija*

Svojstvo ovoga preslika je, da su svi elementi kodomene "prekriveni", što znači da u kodomeni nema nepridruženih elemenata. Međutim dva ili više elemenata domene mogu se pridružiti istome elementu kodomene.

### 10.4.5. 10.4.5. BIJEKCIJSKA FUNKCIJA ILI BIJEKCIJA

Neka su $A$ i $B$ skupovi, a $f$ funkcija iz $A$ u $B$. Ako je propis pridruživanja $f$ između elemenata skupa $A$ i skupa $B$ takav da *različitim* elementima skupa $A$ pridružuje *različite* elemente skupa $B$ (preslik *jedan na jedan*), a nema nepridruženih elemenata, funkcija $f$ zove se *bijekcijska* funkcija ili *bijekcija* (slika 10.7).

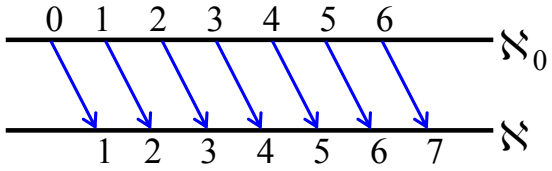



*Slika 10.7: Bijekcijska funkcija*

Kraće se kaže: funkcija $f$: $A \to B$ naziva se *bijekcijska funkcija* ili *bijekcija* ako su ispunjena ova dva uvjeta:

1. $f$ je injekcija,
2. $f$ je surjekcija.

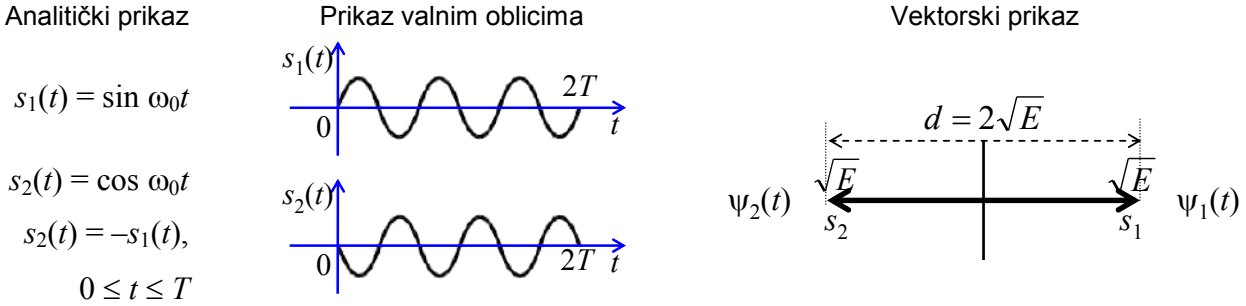
Primjer: Funkcija $f$: $\aleph_0 \to \aleph$ definirana kao: $y = f(x) = x + 1$ je bijekcija, jer svakome elementu iz domene pripada samo jedan element iz kodomene, a različitim elementima domene pripadaju različiti elementi kodomene. Uz to, svaki element iz kodomene "preslik" je samo jednoga elementa iz domene (slika 10.8).




*Slika 10.8: Preslik elemenata među domenama*

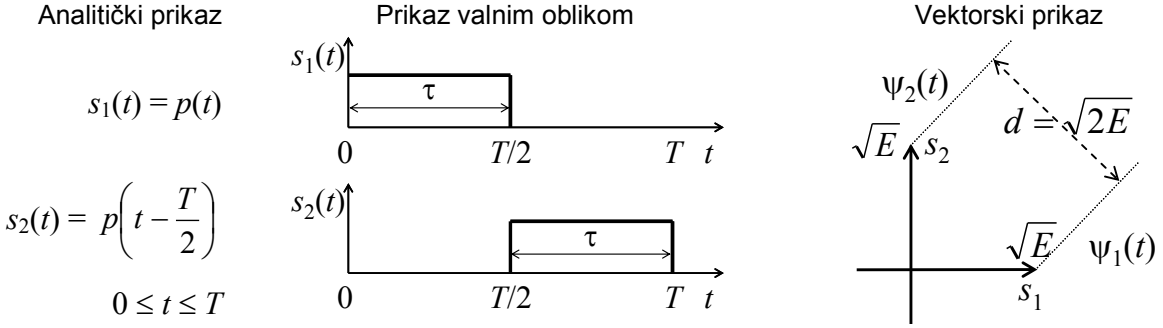
## 10.5. 10.5. Protufazni i okomiti signali

Dva *protu-fazna signala*, zrcalne su slike jedan drugome. Jedan signal negativan je u odnosu na drugi, ili su signali razmaknuti za 180°. Sinusni protufazni signali na slici 10.9, predstavljeni su: analitički, valnim oblicima i vektorski.




*Slika 10.9: Primjer skupa protufaznih signala*

Slika 10.10 prikazuje skup *okomitih* signala koji se sastoje od impulsnih valnih oblika.




*Slika 10.10: Primjer skupa binarnih okomitih signala.*

Za *binarne* okomite signale opisane jednadžbama na slici 10.10, $p(t)$ je impuls trajanja $\tau = T/2$, a $T$ je trajanje (ponavljanja) simbola. Drugi skup okomitih valnih oblika često se koristi u komunikacijskim

sustavima gdje se koriste sin *x* i cos *x*. Općenito, skup signala jednakih energija $s_i(t)$, gdje je $i = 1, 2, \ldots, M$, predstavlja okomit (normaliziran na 1) skup onda i samo onda ako je

$$z_{i,j} = \frac{1}{E}\int_0^T s_i(t)s_j(t)dt = \begin{cases} 1 & \text{za } i = j \\ 0 & \text{za } i \neq j \end{cases} \tag{10.19}$$

gdje se $z_{ij}$ naziva *koeficijent križne korelacije*, a $E$ je energija signala, izražena kao

$$E = \int_0^T s_i^2(t)dt \tag{10.20}$$

Valni oblici prikazani na slici 10.10 pokazuju da se $s_1(t)$ i $s_2(t)$ ne mogu miješati jedan s drugim, jer su pomaknuti u vremenu. Vektorski prikaz pokazuje okomit odnos između pravokutnih signala. Unutarnji - skalarni (ili umnožak točkom) dvaju različitih vektora u okomitome skupu, jednak je nuli. U dvo- ili tro-dimenzijskome Kartezijevome koordinacijskome prostoru, geometrijski možemo opisati vektore signala, kao međusobno okomite. Možemo reći da jedan vektor ima nultu projekciju na drugi, ili da jedan signal ne može ometati drugoga, jer oni ne dijele isti *prostor* signala.

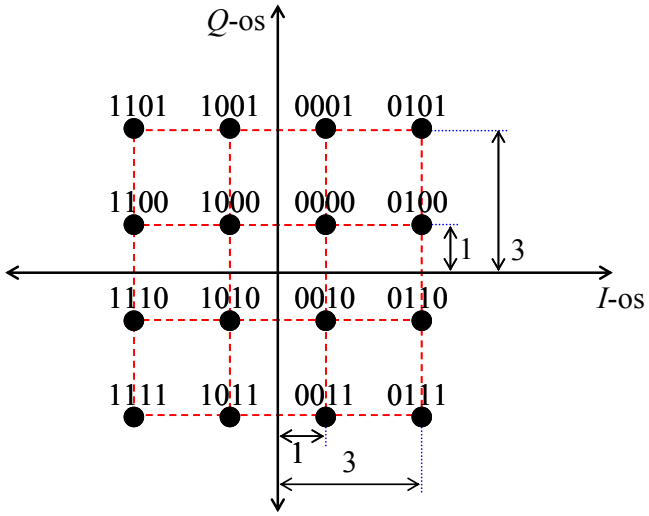
### 10.5.1. 10.5.1. EUKLIDOVA I HAMMINGOVA UDALJENOST

Hammingovu udaljenost već smo upoznali ali ona nije jedina mjera međusobne razmaknutosti binarnih vektora. Druga korištena mjera je Euklidova udaljenost. Definirat ćemo je i objasniti s nekoliko jednostavnih prikaza.

Udaljenost između bilo kojih dviju točaka u Kartezijevome koordinacijskom sustavu zove se *Euklidova udaljenost*. Za točku $p_1$ čije su koordinate $(x_1, y_1)$ i točku $p_2$ čije su koordinate $(x_2, y_2)$, Euklidova udaljenost računa se formulom za pravokutan trokut:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{10.21}$$

Ova udaljenost najkraća je udaljenost između dviju točaka. Za signale, definirat ćemo tu udaljenost u $IQ$[138] ravnini (vidi sliku 10.11).

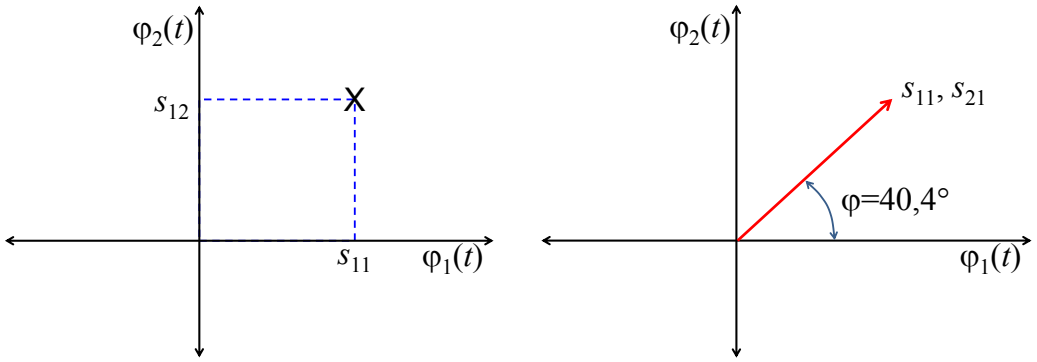



*Slika 10.11: I-Q ravnina*

### 10.5.2. 10.5.2. KONCEPT I I Q KANALA

Definirajmo signal kao vektor. Dva kanonska[139] načina predstavljanja takvoga signala su, prikaz signala: u pravokutnome i u polarnome obliku. Polarne koordinate opisuju signal fazom (kutom) i njegovom pravokutnom projekcijom, kao što su $s_{11}$ i $s_{12}$ na slici 10.12.

[138] Projekcije signala *u fazi* (***I**n-phase*) i kvadraturi (***Q**uadrature*).




(a) *I*-*Q* projekcije (b) Polarni oblik

*Slika 10.12: Vektor signala nacrtan u prostoru signala*

Osi $x$ i $y$ na slici 10.12.a nazivaju projekcije signala *u fazi* (***In-phase***) i kvadraturi (***Quadrature***). Veličina $s_{11}$ je I-projekcija, a $s_{12}$ je $Q$-projekcija signala. Slika 10.12.b prikazuje isti signal u polarnome obliku i njegovu dužinu jednaku amplitudi signala i kutu jednakome njegovoj fazi.

Koeficijent $s_{11}$ predstavlja amplitudu $I$ signala, a $s_{12}$ predstavlja amplitudu $Q$ signala. Ove amplitude kada se nacrtaju na $x$ odnosno $y$ osi, prikazuju vektor signala. Kut vektora signala s $x$-osi tvori fazu toga signala.

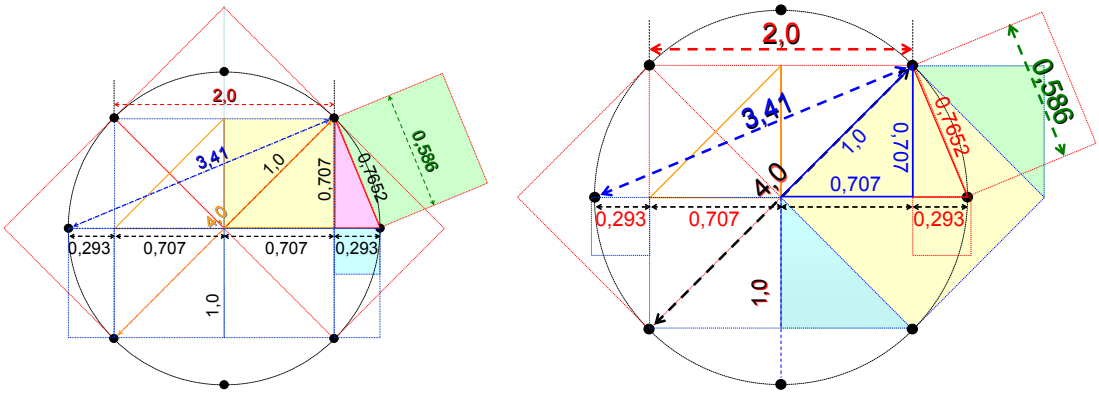
Amplituda signala:

$$S = \sqrt{I^2 + Q^2} \tag{10.22}$$

Faza signala:

$$\varphi = \text{arc tg}\frac{I}{Q} \tag{10.23}$$

Promatramo konstelaciju 8-PSK signala. Radijus je jednak 1 i predstavlja najveću amplitudu. Svaka točka konstelacije, određena je sjedinjenjem amplitude i faze. Razmak između tih točaka može se mjeriti na prethodno opisan način. Navedene udaljenosti su kvadrirane i nazivaju se *kvadrirana Euklidova udaljenost* SED (*Squared Euclidean Distance*). Najmanja od tih udaljenosti zove se minimalna kvadrirana Euklidova udaljenost MSED (*Minimum Squared Euclidean Distance*), označena kao $d_{\min}$ za određenu konstelaciju (slika 10.13).



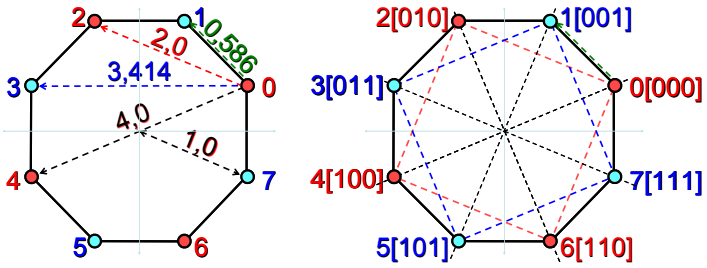

*Slika 10.13: PSK konstelacija i kvadrirane Euklidove udaljenosti između simbola*

[139] *kanonski* … (lat. *canonicus*, grč. *kanon*) koji odgovara crkvenim propisima (kanonima), koji je u skladu s crkvenim propisima ili autoritetom, propisan, vjerodostojan (knjige, spisi); crkveni, papinski; pren. koji služi kao uzor, uzoran, ugledan

### 10.5.3. 10.5.3. RAZMACI IZMEĐU NIZOVA

Spoznali smo da binarni brojevi imaju koncept udaljenosti. Uzmimo dva binarna broja, 0110011 i 1011011. Razmak između njih je broj mjesta u kojima se te dvije brojke razlikuju i taj broj je 4. Ova udaljenost zove se *Hammingova udaljenost* i jednaka je kao i predstavljanje Euklidovoga koncepta udaljenosti. Razlikujemo ove dvije vrste udaljenosti, jedna pripada analognome svijetu realnih brojeva (Euklidova), a druga binarnome svijetu (Hammingova).

Također možemo govoriti o Euklidovoj udaljenosti između nizova uspoređujući udaljenosti između odgovarajućih točaka nizova. Za primjer uzmimo 8-PSK signal koji se sastoji od niza binarnih simbola. U ovome signalu razlikuju se sljedeće udaljenosti: $s_0$ $s_4$ $s_3$ $s_2$ $s_1$ $s_0$. Iskazano bitovima, možemo ih preslikati kao: [000], [100], [011], [010], [001], [000] (skupine po 3 znamenke predstavljaju stanja).




*Slika 10.14: Euklidova i Hammingova udaljenost niza*

Euklidova udaljenost za ovaj niz, udaljenost je između referentnoga niza "sve 0" i svakoga znaka u ovome nizu. Ako je to niz [000], onda je *zbroj kvadriranih Euklidovih udaljenosti* SED (*Sum of the Squared Euclidean Distances*), udaljenost između simbola $s_0$ i svakoga od tih simbola

$s_0$ prema $s_1$ = 0.586 (= 1 – 0,705 = 0,295 $\Rightarrow$ $(0{,}705)^2 + (0{,}295)^2 = 0.497 + 0{,}087 = 0{,}586$)
$s_0$ prema $s_2$ = 2.0,
$s_0$ prema $s_3$ = 3.414
$s_0$ prema $s_4$ = 4
$\sqrt{2}/2 = 1{,}41/2 = 0{,}705 \Rightarrow (0{,}705)^2 + (0{,}705)^2 = 1^2 = 1$

Zbroj kvadriranih Euklidovih udaljenosti SSED (*Sum of the Squared Euclidean Distances*) (označava se kao $d^2_{free}$) svih nizova, jednaka je $d^2_{free}$ = 0,586 + 2,0 + 3,414 + 4,0 = 10,0.

Ova kumulacijska udaljenost opisuje koliko lako (ili teško) možemo zabunom zamijeniti jedan niz drugim nizom. Za referentan niz, mogli smo koristiti bilo koji drugi niz osim niza "sve 0", a rezultati, bili bi isti. Međutim zgodno je i uobičajeno koristiti niz "sve 0".

## 10.6. 10.6. Vrste provjera zalihosti

### 10.6.1. 10.6.1. NAČINI PROVJERE ZALIHOSTI

#### 10.6.1.1. 10.6.1.1. Otkrivanje pogrešaka

Spomenuli smo na početku skripte da je osiguranje cjelovitosti podataka bila jedna od ključnih odgovornosti za okruženje u kojima rade podatkovne komunikacije. To ne znači da su podaci točni. Umjesto toga, to znači da smo na neki način jamčili, uz iznimno visok stupanj vjerojatnosti, da je informacija primljena u istome obliku kao što se poslala.

Točnije, zahtijevaju se dvije zadaće: *otkrivanje*, ako se pojave pogreške pri prijenosu te *ponavljanje slanja* u slučaju otkrivene pogreške. Protokol je odgovoran aktivirati i upravljati ponovnim odašiljanjem. Ovdje ćemo opisati neke pristupe koji su se razvili prvenstveno za otkrivanje pogrešaka.

Svi skupovi kodova koji se koriste u komunikaciji podacima, oblikovani su za korištenje svih svojih bitova da bi predstavili znakove (slova, brojeva i druge posebne znakove). Posljedica ove činjenice je da svaki primljen bajt u takvome kodu, predstavlja (prema definiciji) valjan kod. Kako možemo otkriti je li primljen kod jednak kodu koji se poslao?

Rješenje je da se uz primarne podatke nekako pošalju dodatne informacije, neki podaci o podacima (ponekad nazvani *meta* podaci). Svi pristupi provjeri pogrešaka ovise o slanju tih dodatnih podataka,

osim izvornih podataka pridruženih samoj aplikaciji. Dodatni podaci koji se stvaraju tijekom procesa komunikacije, koriste se za provjeru podataka u pozadini kada se prime, a zatim se odbacuju prije no što informacija dođe do svoga konačnoga odredišta.

S ciljem povećanja pouzdanosti, glavne metode što se koriste za otkrivanje pogrešaka komunikacijskih podataka, uključuju bit pariteta, odnosno:

- *okomitu provjeru zalihosti* VRC (*vertical redundancy checking*),
- *uzdužnu provjeru zalihosti* LRC (*Longitudinal redundancy checking*) i
- *cikličku provjeru zalihosti* CRC *Cyclic redundancy checking*).

Pretpostavimo elektronički prijenos novca. Ako se koristi nepouzdana mreža, a ne primijene se odgovarajuće metode otkrivanja pogrešaka, promjena jedne znamenke ili npr., umetanje decimalne točke umjesto zareza, može rezultirati značajnom novčanom pogreškom.

Šansa da se zaradi novac na ovakav način nije velika, obzirom da je promjena decimalne točke u rasponu do 0 do 126 (ili 254) mogućnosti. Vjerojatnost otkrivanja pogreške *okomitom* provjerom, koristeći paritetne bitove je oko 65%, jer promjena nekoliko pogrešnih bitova, u jednome trenutku promijeni cijeli znak. Bolji način, je *uzdužna* provjera zalihosti. Ona otkriva oko 85% pogrešaka. Metoda *cikličke* provjere zalihosti povećava mogućnost otkrivanja i ispravak takve pogreške na 99.99995%.

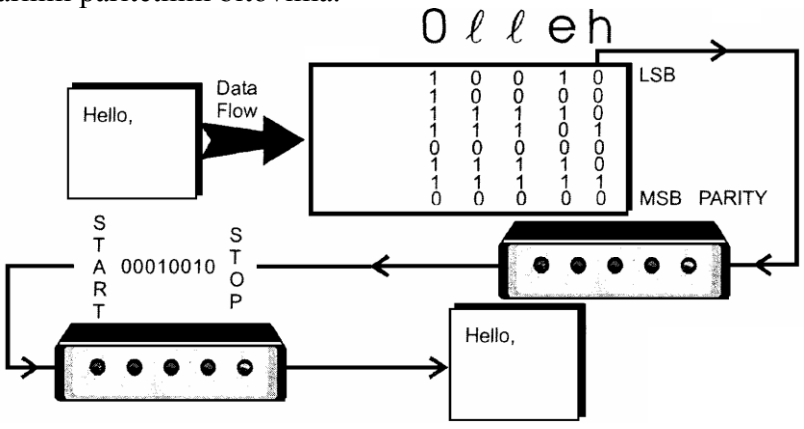
Budući da mreže koje se koriste za slanje novčanih iznosa, uglavnom koriste CRC tehnike, mala je nada da se netko može obogatiti zbog ovakvih pogrešaka. Detaljnije ispitajmo ove metode.

#### *10.6.1.2. 10.6.1.2. Bit pariteta/okomita provjera zalihosti (VRC)*

Otkrivanje pogrešaka paritetnim bitom, jednostavno dodaje jedan "kontrolni" bit svakome poslanome znaku (ili bajtu). Bilo da je bit pariteta postavljen na 0 ili 1 (jedine dvije mogućnosti) računa ga odašiljački digitalni uređaj, a provjerava ga prijemnik u međusobnome suglasju. Ako se izračuni podudaraju, pridružen znak smatra se ispravnim, inače, otkriva se pogreška.

To je mnogo jednostavnije nego što zvuči. Obično se koriste dva pristupa: *paran* i *neparan* paritet Postoje i drugi oblici pariteta, kao što je paritet znaka ili praznine ali nema razlike pri korištenju. Jedini uvjet je da prijemnik i odašiljač koriste isti pristup. Za ilustraciju parnoga pariteta, razmotrimo ASCII niz bitova koji predstavlja malo slovo *a*: [1100001]. Zato što koristimo samo paritet, tražimo da ukupan broj jedinica u nizu bitova koji se šalju prijemniku kao npr. slovo *a*, uključi i osmi kontrolni bit, da bi ukupno imali paran broj jedinica. Njihov položaj u temeljnome bajtu nevažan je. Ako zbrojimo broj jedinica u uzorku od 7 bitova, imamo ih 3, dakle neparan broj.

Stoga postavljamo bit pariteta na 1, što rezultira vektorom [11100001], a to je uzorak od 8 bitova s parnim brojem (tj. četiri) jedinice. Paritetan bit šalje se zadnji. U našim prikazima, oni bitovi sasvim desno, šalju se prvi, tako da se bit pariteta dodaje lijevo. Slika 10.15 prikazuje ASCII prikaz riječi "Hallo", zajedno s parnim paritetnim bitovima.




*Slika 10.15: Provjera okomite zalihosti pokazuje tok podataka. Bit pariteta dodaje se nakon što se generira svaki znak.*

Kao što se može vidjeti na slici, bajtovi su prikazani kao okomiti skupovi brojeva, uz okomitu provjeru zalihosti. Ako znamenke okrenemo okomito, zalihosni bit dodamo okomito (dolje) da bi se provjerila ispravnost prikazanoga bajta. Naravno da je okomito usmjerenje proizvoljno. Međutim, ako se prikaže *uzdužna provjera zalihosti*, vidjet će se da postoji razlog za ovakav prikaz.

Nakon što su se detaljno opisali paritetni bitovi, od njih imamo ograničene koristi u komunikaciji podataka. Provjera parnosti obuhvatit će 100% pogrešaka, gdje je broj neispravnih bitova neparan, a niti jednu od pogrešaka, gdje je broj neispravnih bitova paran. Drugim riječima, ako se pojavi pogreška (i komunikacijske pogreške rijetko utječu samo na jedan bit), postoji samo oko 65% šanse da će ga provjera pariteta otkriti. Vjerojatnost je veća od 50%, jer više ima pogrešaka od jednoga bita, nego bilo koje vrsta višestrukih pogrešaka bajtova, neovisno je li broj višestrukih pogrešaka paran ili neparan.

Provjera pariteta koristi se opsežno unutar računala. To ima smisla, jer je sasvim moguće da će se pojaviti pogreške, jedna po jedna (ako se uopće pojave). Provjera pariteta dobro radi u ovakvome okruženju. Također, neke mreže i dalje koriste komunikacijski program za provjeru pariteta. Ali za prijenos datoteka, koriste se složeniji protokoli.

#### *10.6.1.3. 10.6.1.3. Uzdužna provjera zalihosti (LRC)*

Koncept LRC slijedi izravno iz VRC, daljnjim razvojem VRC. Ovaj primjer koristi bajtove od 8 bitova, a ne od 7 bitova. Ali LRC provjere trebaju istovremeno raditi na skupini bajtova, a ne na jednome bajtu. Za ovaj primjer, zapravo nije bitno što bitovi predstavljaju. Stvorimo niz od osam bajtova, svaki od 8 bitova. Kao što će se vidjeti, iako obrasci bitova nisu važni za primjer, zaista se koristi brojni bajtovi (vidi tablicu 10.1). Ako koristimo samo VRC kao što se opisalo (neparan paritet), proizveden uzorak prikazuje tablica 10.2.

*Tablica 10.1: Postavke za okomitu provjeru pariteta (8 × 8) (paran paritet)*[140]

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

*Tablica 10.2: Kontrolni bit umetne se u VRC (8 × 8) (neparan paritet)*

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Naravno, nije prihvatljivo to što VRC otkriva samo oko 65% pogrešaka. Ali što ako se primijeni *neparna* paritetna provjera bajtova zajedno s okomitom? U tome slučaju, generirat će se u potpunosti ispunjena Tablica 10.3.

*Tablica 10.3: Umeću se okomite i uzdužne provjere* (8 × 9)

| | | | | | | | | | *neparan paritet vodoravno - HRC* |
|---|---|---|---|---|---|---|---|---|---|
| | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | |
| | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | |
| | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | |
| | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | |
| | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | |
| | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | |
| *neparan paritet okomito - VRC* | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | *neparan paritet uzdužno - LRC* |

Dodavanjem i vodoravne i okomite provjere zajedno, naziva se *uzdužna provjera zalihosti* LRC (*longitudinal redundancy checking*). Ona povećava izglede za otkrivanje pogrešaka na oko 85%. Nije loše, iako npr. ne bi željeli povjeriti prijenos našega novca takvome prijenosu. No, tu je još jedan nedostatak LRC. Koristeći bajtove od 8 bitova za svakih od osam podatkovnih bajtova, treba prenositi dodatna dva LRC bajta. Zbog provjere došlo je do 20% dodanoga preteka (2 LRC bajta u odnosu na ukupno 10 bajtova koji se prenose), ne računajući nikakvo narušavanje svojstava zbog vremena

[140] "*Nolite iacere margaritas ante porcos*"… (Dr. Č. B., 2010.)

potrebnoga za izračun bajtova provjere. Ovo nije učinkovit mehanizam za provjeru pogrešaka. U stvari, nema smisla provjera pogrešaka samo jednoga od nekoliko izvora za dodatan prijenos.

LRC ima jednu prednost u odnosu na pristup provjere cikličke zalihosti CRC (*cyclic redundancy checking*), što znači da je za izračun LRC bajtova potrebno daleko manje računalnih resursa nego za izračun CRC (osim ako se CRC ne provodi sklopovski). U stvari, sve do dostupnosti posljednjih generacija računala sa svojim neizmjerno snažnijim CPU, CRC provjera za asinkrone komunikacije podataka u PC okruženju nije bila praktična zbog njezine računalski zahtjevne naravi. Sada, međutim, ona se koristi rutinski.

#### *10.6.1.4. 10.6.1.4. Ciklička provjera zalihosti (CRC)*

Iako praktički niti jedan algoritam provjere pogrešaka ne jamči otkrivanje svakoga mogućega uzorka pogreške, CRC provjera tome je najbliža. Dodatna objašnjenja i primjeri kako CRC radi, zahtijevaju binarnu diobu polinoma opisanu u poglavlju 9.1. Umjesto toga, opisat će se neka od ključnih svojstava ove metode te će se pokazati kako se ona koristi.

Poput prethodno opisanih pristupa otkrivanju pogrešaka, CRC se oslanja na brz izračun (*on-the-fly-calculation*) dodatnoga uzorka bita poznatoga kao *niz provjere okvira* FCS (*frame check sequence*) koji se šalje odmah nakon izvornoga informacijskoga bloka bitova. Oblikovatelj programa ili sklopa, unaprijed izabere duljinu FCS na temelju toga kolika se razina povjerenja zahtijeva u mogućnosti otkrivanja pogreške određenoga prijenosa. Svaki *prasak pogrešaka* ili slučajna pogrešna skupina bitova zbog problema u prijenosu, mora imati duljinu manju od projektirane FCS da bi se otkrila. Često korištene FCS duljine su 12, 16 i 32 bita. Očito, što je veći broj znamenaka FCS, otkrit će se više pogrešaka.

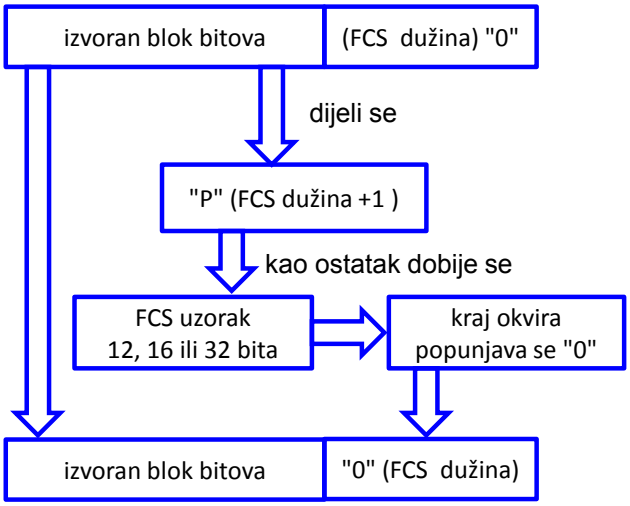
FCS se računa tako da se prvo uzme izvorni uzorak bitova bloka podataka (smatra ga se kao jedan ogroman binaran broj) i na njegov završetak (iza bitova najniže težinske razine) dodaju se neke dodatne binarne nule. Točan broj dodanih nula isti je kao i broj bitova u željenome FCS (umnošku). Jednom izračunat FCS, prekrit će ove nule. Zatim, dobiven binarni broj, uključujući i dodane znamenke, podijelimo posebnim, prethodno odabranim djeliteljem [često poznat iz opisa algoritma kao $G(x)$].

Djelitelj $G(x)$ ima sljedeća svojstva:

- Uvijek je za jedan bit duži od željenoga FCS.
- Njegovi prvi i posljednji bitovi su uvijek 1 (LFSR).
- Izabire se da bude "relativno prost broj" prema FCS, to jest, $G(x)$ podijeljen s FCS uvijek će dati ne-nulti ostatak. U praksi, to znači da je $G(x)$ normalan prost polinom.
- Dijeljenje koristi binarnu diobu, što je puno brži i jednostavniji postupak od dekadskoga dijeljenja. Ostatak diobe (u stanjima LFSR) postaje željeni FCS.

Posebne primjene CRC koristiti posebne djelitelje. Tako protokol provjere pogrešaka CRC-32 u jednome sustavu treba surađivati s CRC-32 protokol u drugome sustavu. Protokol CRC-CCITT (koji koristi uzorak od 17 bitova, stvarajući FCS od 16 bitova) isto tako trebao bi komunicirati s CRC-CCITT primjenom. Izbor određenoga $G(x)$ može se ugoditi prema vrstama najvjerojatnijih pogrešaka koje se pojavljuju u određenome okruženju. Osim ako se ne planira inženjering novoga protokola, ne moramo se brinuti o postupku izbora. To je već učinio oblikovatelj sklopovlja i/ili komunikacijskoga programa.

CRC se obično koristi u blokovima ili okvirima podataka, a ne na pojedinačne bajtu. Ovisno o protokolu koji se koristi, blokovi mogu imati veličinu od nekoliko tisuća bajtova. Dakle, u smislu provjere informacijskih bitova na pogreške, potrebne za određen broj bajtova podataka, CRC zahtijeva daleko manje suvišnoga prijenosa. Npr. CRC-32 šalje 4 riječi od 8 bitova zadužene za provjeru pogrešaka bitova i njima se provjerava tisuće podatkovnih bajtova, a ne samo pojedini bajt. Slika 10.16 prikazuje postupak stvaranja CRC.




*Slika 10.16: Stvaranje CRC ciklusa.*

Binarna dioba vrlo je učinkovita, ali ako se obavlja na svakome bloku koji se prenosi, značajno usporava prijenos. Srećom, za procesore razvijene posljednjih godina, to je izazov. Također, za razliku od većine drugih provjera ispravke pogrešaka digitalnih podataka, prijemni uređaj ili program ne mora preračunati FCS da bi se provjerile pogreške. Umjesto toga, izvorni podaci plus FCS, nanizani su zajedno oblikujući duži uzorak. Zatim se taj uzorak dijeli istim $G(x)$ koji se koristio kao djelitelj tijekom procesa stvaranja FCS. Ako nema ostatka pri ovoj zadnjoj diobi, onda CRC algoritam pretpostavlja da pogreške ne postoje.

A pogreške u mrežama zaista ni ne postoje u 99.99995% slučajeva (ili vremêna)!

# 11. POGLAVLJE 11 - KORIŠTENA LITERATURA


[1] A Commonsense approach to the theory of error correcting codes

[2] Algebraic Codes for Data Transmission

[3] A Mathematical Theory of Communication

[4] All about Digital Modulation

[5] Bežični LAN 802.11 OCR

[6] Communication Systems Engineering 2nd Edition

[7] Communication Technology Update and Fundamentals

[8] CRC.Rf.Transmission.Systems.Handbook

[9] Data Structures and Algorithms in C++ 2ndEd

[10] Digital Communication

[11] Digital Communications 3rd Ed I Glover P Grant Pearson 2010

[12] Digital Communications Fundamentals and Applications 2.Ed

[13] Digital design 5E

[14] Digital Design for the Laboratory

[15] Digital Logic

[16] Digital Systems Design Using VHDL OCR

[17] Digitalne telekomunikacije

[18] Digitalni prijenos informacija

[19] Error Control Coding Second Edition

[20] Error Control Coding Fundamentals and Applications

[21] Error-Correcting Codes

[22] Exploring Digital Logic with Logisim

[23] FDDI Technology and Applications_

[24] Kriptografija

[25] lecture_note_2007

[26] Network Algorithmics

[27] Problem Solving with C++ [7th Edition]\Problem Solving with C++ 7th Ed

[28] Protocols for Authentication and Key Establishment

[29] Šifre

[30] Sigurnost računarskih sistema i mreža

[31] Teorija informacija 1 OCR

[32] Trellis Code Modulation

[33] Trellis and Turbo Coding-U0581065

[34] COM-7001 Turbo Code Error Correction Encoder/Decoder (VIT)

[35] ungerboeck0287

[36] Wireless Communications

[37] Aktualna literatura za ZKS.doc