UNIVERZITET CRNE GORE
ELEKTROTEHNIČKI FAKULTET

Aleksej Vukčević

Projektovanje i analiza hardvera za realizaciju odabranih algoritama za detekciju i korekciju greške

MASTER RAD

Podgorica, novembar 2024.

PODACI I INFORMACIJE O STUDENTU

Ime i prezime: Aleksej Vukčević
Datum rođenja: 11.10.2000.
Mjesto rođenja: Crna Gora, Bar
Naziv završenog osnovnog studijskog programa: Elektronika, Telekomunikacije i Računari
Godina završetka studija: 2022.

INFORMACIJE O MASTER RADU

Naziv master studija: Računari
Naslov rada: Projektovanje i analiza hardvera za realizaciju odabranih algoritama za detekciju i korekciju greške
Fakultet/Akademija na kojem je rad odbranjen: Elektrotehnički fakultet

UDK, OCJENA I ODBRANA MASTER RADA

Datum prijave master rada: 25.06.2024.
Datum sjednice Vijeća na kojoj je prihvaćena tema: 16.09.2024
Mentor: Prof. dr. Nevena Radović
Komisija za ocjenu/odbranu rada:
Prof. dr Veselin Ivanović, predsjednik
Doc. dr Nevena Radović, mentor
Prof. dr Milena Erceg, član
Datum odbrane: 27.10.2025.

Ime i prezime autora: Aleksej Vukčević, BSc

Etička izjava

U skladu sa članom 22 zakona o akademskom integritetu i članom 18 Pravila studiranja na master studijama, pod krivičnom i materijalnom odgovornošću, izjavljujem da je master rad pod naslovom

„Projektovanje i analiza hardvera za realizaciju odabranih algoritama za detekciju i korekciju greške"

moje originalno dijelo.

Podnosilac izjave,

Aleksej Vukčević

Aleksej Vukčević, BSc

U podgorici, dana 05.06.2025 godine

Predgovor

Teorija komunikacija se suočava s problemom detekcije i korekcije grešaka, koji ima veliku praktičnu važnost. Kodovi za korekciju grešaka omogućavaju ne samo otkrivanje, već i ispravljanje grešaka koje nastaju usljed šuma, smetnji ili drugih nepouzdanosti u komunikacionom kanalu, čime se obezbjeđuje pouzdanost prenosa. Algoritmi za detekciju i korekciju grešaka omogućavaju lokalizaciju grešaka, kao i njihovo ispravljanje, bez potrebe za ponovnim prenosom podataka. Ovakvi algoritmi su od suštinskog značaja u sistemima gdje je integritet podataka presudan, bilo da se radi o digitalnim komunikacijama, skladištenju podataka poput dinamičkih RAM memorija i optičkih diskova, ili o sistemima u realnom vremenu gdje gubitak informacija nije prihvatljiv.

Ovaj master rad rezultat je višemjesečnog rada i istraživanja u oblasti digitalnih komunikacija i projektovanja hardverskih sistema. Fokusiran je na dizajn i implementaciju hardvera za realizaciju odabranih algoritama za detekciju i korekciju grešaka u FPGA tehnologiji, sa ciljem postizanja optimalnih performansi u pogledu tačnosti, brzine i iskorišćenosti resursa. Tokom rada posebna pažnja posvećena je poređenju različitih kodova i arhitektura, kao i identifikaciji prednosti i ograničenja svakog od pristupa.

Posebnu zahvalnost dugujem svojoj mentorki, doc. dr Neveni Radović, na nesebičnoj pomoći, stručnom savjetovanju i strpljenju tokom izrade ovog rada. Njen stručni pristup, motivacija i korisni komentari bili su od izuzetne važnosti za uspješnu realizaciju istraživanja.

Sažetak

U matematici, digitalnim komunikacijama i teoriji informacija, detekcija i korekcija grešaka imaju veliku praktičnu važnost u očuvanju integriteta informacija tokom njihovog prenosa kroz kanale zahvaćene šumom. Kodiranje kanala predstavlja metodu detekcije i korekcije ovih grešaka, koja osigurava da informacija ostane netaknuta tokom prenosa od izvora do odredišta. Ovakva vrsta kodiranja zasniva se na dodavanju redudancije informacionom signalu, prema unaprijed definisanom algoritmu. Postoje različiti algoritmi za detekciju i korekciju grešaka, a neki od najčešće korišćenih su: pravougaoni, trougaoni, Hamming-ov, *Bose, Chaudhuri and Hocquenghem* (BCH), Reed-Solomon i konvolucioni. Ovaj master rad se fokusira na dizajn i hardversku implementaciju ovih algoritama u *Field-Programmable Gate Array* (FPGA) tehnologiji. Predložena rješenja obuhvataju kodere i dekodere za svaki od navedenih algoritama, čime je pokriven kompletan dizajn sistema za detekciju i korekciju grešaka. Svi kodovi su napisani korišćenjem *VHSIC Hardware Description Language* (VHDL) programskog jezika, u okviru Quartus II razvojnog okruženja. Simulacije i verifikacije su izvedene korišćenjem ModelSim Altera alata, dok se vizuelizacija generisanih netlisti obavlja pomoću Netlist Viewer alata.

Ključne reči: Kodiranje kanala, Algoritmi za detekciiju i korekciju grešaka, FPGA, Reed-Solomon kod, BCH kod, Hamming-ov kod, pravougaoni kod, trugaoni kod, konvolucioni kod.

Abstract

In mathematics, digital communications, and information theory, error detection and correction play a crucial role in maintaining the integrity of information during its transmission through noisy channels. Channel coding is a method for detecting and correcting these errors, ensuring that the information remains intact during transmission from the source to the destination. This type of coding is based on adding redundancy to the information signal according to a predefined algorithm. There are various algorithms for error detection and correction, some of the most commonly used being: rectangular, triangular, Hamming, *Bose, Chaudhuri and Hocquenghem* (BCH), Reed-Solomon, and convolutional codes. This master thesis focuses on the design and hardware implementation of these algorithms using *Field-Programmable Gate Array* (FPGA) technology. Proposed solutions include encoders and decoders for each of the mentioned algorithms, covering the complete system design for error detection and correction. All the codes are written using the *VHSIC Hardware Description Language* (VHDL) programming language in the Quartus II development environment. Simulations and verifications were carried out using the ModelSim Altera tool, while the visualization of the generated netlists is performed using the Netlist Viewer tool.

Keywords: Channel coding, Error detection and correction algorithms, FPGA, Reed-Solomon code, BCH code, Hamming code, Rectangular code, Triangular code, Convolutional code.

Sadržaj

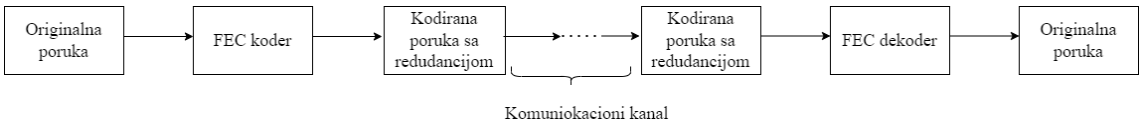
Popis slika

Popis tabela

1 Uvod

Digitalna komunikacija je prisutna u svakom aspektu naših života: mobilne komunikacije, radio komunikacije, satelitski prenosi podataka, mrežni prenosi podataka, transferi računarskih fajlova, itd. U digitalnoj komunikaciji signali, odnosno nosioci podataka, koji su predstavljeni u binarnom obliku, se prenose kroz komunikacioni kanal od pošiljaoca (izvora) do prijemnika. Tokom prenosa mogu se javiti greške, uzrokovane različitim faktorima. Pod greškom podrazumijevamo stanje u kojem izlazna informacija ne odgovara ulaznoj informaciji, što znači da se bit „0“ može promijeniti u „1“ ili bit „1“ u „0“. Ovakve greške mogu značajno uticati na tačnost i performanse sistema. U cilju unapređenja pouzdanosti prenosa podataka, neophodno je obaviti deketkciju i korekciju ovih grešaka, [1–3].

Kako bi se ovo postiglo, predložene su razne tehnike korekcije grešaka unaprijed (engl. Forward Error Correction, FEC). FEC koder dodaje redudanciju podacima prije prenosa. Po prijemu, prijemnik detektuje i ispavlja greške. Broj grešaka koje se mogu detekovati i ispraviti na prijemnoj strani zavisi od ograničenja korišćenog algoritma. Opšti koncept FEC-a prikazan je na slici 1, [4].



Slika 1: FEC koncept

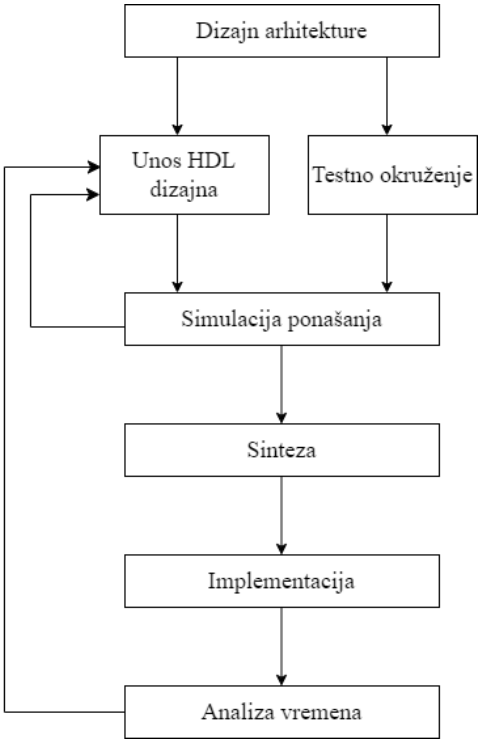
FEC algoritmi za detekciju i korekciju grešaka igraju ključnu ulogu u osiguravanju integriteta podataka tokom njihovog prenosa putem različitih komunikacionih kanala. Ovi algoritmi imaju širok spektar primjena i prioriteti koji moraju biti zadovoljeni variraju u zavisnosti od konkretnih zahtjeva sistema i specifičnosti aplikacija, [3, 5, 6].

U okviru drugog poglavlja master rada su definisani algoritmi za detekciju i korekciju greške koji se najčešće srijeću u praksi: pravougaoni, trougaoni, Hammingov, *Bose, Chaudhuri and Hocquenghem* (BCH), Reed-Solomon i konvolucioni. Za svaki od algoritama je detaljno objašnjen princip kodiranja i dekodiranja, te potkrijepljen odgovarajućim primjerima.

Pregled nekih od postojećih rješenja za realizaciju algoritama navedenih u poglavlju 2 biće dat u trećem poglavlju rada, [2, 7–11].

Cilj istraživanja ovog master rada je projektovanje hardverskih rješenja za odabrane, najčešće korišćene algoritame za detekciju i korekciju grešaka korišćenjem

Field-Programmable Gate Array (FPGA) tehnologije. FPGA čipovi su unaprijed izrađeni silicijumski uređaji koji se mogu električki programirati da postanu gotovo bilo koji oblik digitalnog kola ili sistema. FPGA uređaji pružaju brojne prednosti u odnosu na tehnologije sa fiksnim funkcijama kao što su *Application Specific Integrated Circuit* (ASIC) uređaji , koji su dizajnirani za specifičnu primjenu i nakon što se proizvedu, ne mogu se mijenjati. S druge strane, FPGA uređaji mogu se konfigurisati u relativno kratkom vremenskom periodu, a često i rekonfigurisati ako se napravi greška. FPGA se sastoji od niza konfigurabilnih logičkih blokova, programabilnih međuspojnica i ulazno-izlaznih blokova. Navedene karakteristike čine pomenute uređaje mnogo fleksibilnijim sa aspekta vrsta dizajna koje se mogu na njima implementirati. Njihovom upotrebom je omogućena implementacija paralelnih struktura, sa vremenom odziva manjim nego kod sistema sa mikroprocesorom. Proces implementacije dizajna u FPGA tehnologiji može se podijeliti u nekoliko faza, kao što je prikazano na slici 2, [12, 13].

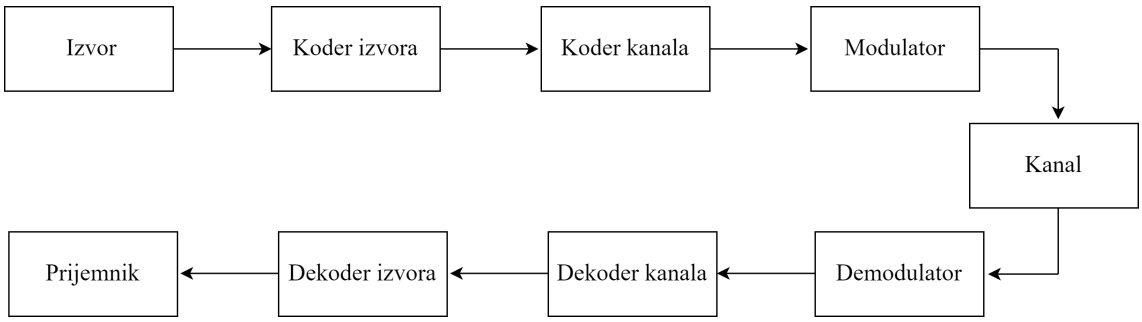


Slika 2: FPGA tok dizajna

U okviru četvrtog poglavlja biće predstavljenja i analizirana projektovana rješenja za hardversku implementaciju kodera i dekodera za svaki od algoritama definisanih u poglavlju 2, sa posebnim osvrtom na utrošene resurse i vrijeme izvršavanja. U posljednjem poglavlju biti izneseni odgovarajući zaključci sa pogledom na kompletan rad„ te istaknuti ostvareni doprinosi zajedno sa idejama za buduća naučna istraživanja.

2 Algoritmi za detekciju i korekciju greške

Digitalni komunikacioni sistem se koristi za prenos signala koji nosi informacije od izvora do korisničke destinacije putem komunikacionog kanala. Informacioni signal se obrađuje u digitalnom komunikacionom sistemu kako bi se formirale diskretne poruke, što čini informacije pouzdanim za prenos. Kodiranje kanala je važna operacija obrade signala za efikasan prenos digitalnih informacija kroz kanal. Kodiranjem kanala broj simbola u izvornoj kodiranoj poruci se povećava na kontrolisan način kako bi se omogućila dva osnovna cilja na strani prijemnika: detekcija grešaka i korekcija grešaka. U cilju ispravljanja grešaka koje su se dogodile i pravilnog dekodiranja poslate poruke, na prijemnoj strani je neophodno postojanje odgovarajućeg dekodera, [5,14,15]. Digitalni komunikacioni sistem ima tri osnovne operacije obrade signala: kodiranje izvora, kodiranje kanala i modulaciju. Digitalni komunikacini sistem prikazan je u blok dijagramu ispod, slika 1.



Slika 3: Blok dijagram digitalnog komunikacionog sistema

U kodiranju izvora, koder mapira digitalne podatke generisane na izlazu izvora u drugi signal u digitalnom obliku. Cilj je eliminisati ili smanjiti redundantnost kako bi se obezbijedila efikasna reprezentacija izlaza izvora. Pošto je mapiranje izvora kodera jednoznačno, dekoder izvora na drugom kraju jednostavno vrši obrnuto mapiranje, čime korisniku isporučuje reprodukciju originalnog digitalnog izlaza izvora. Primarna prednost primjene kodiranja izvora je smanjenje zahtjeva za propusnim opsegom. U kodiranju kanala, cilj kodera je da mapira ulazni digitalni signal u ulaz kanala, dok je cilj dekodera da mapira izlaz kanala u izlazni signal na način da se minimalizuje uticaj šumova u kanalu. Uloga kodera i dekodera kanala je da obezbijede pouzdanu komunikaciju preko kanala zahvaćenog šumom. Ova funkcija se ostvaruje pomoću kodova za detekciju i korekciju grešaka, koji dodaju redundantnost na propisan način u koder kanala i koriste je u dekoderu kako bi se što tačnije rekonstruisao originalni ulaz kodera. Danas imamo na raspolaganju veliki broj kodova za detekciju i korekciju greške različitih karakteristika, a oni se mogu podijeliti u dvije glavne grupe, [5,14]:

- **Blok kodovi**

  Blok kodovi su kodovi za ispravljanje grešaka u kojima se poruka dijeli na blokove fiksne veličine, pri čemu svaki blok sadrži k informacionih simbola unaprijed određene veličine. Svaki blok se kodira kako bi formirao kodnu riječ veličine n simbola. Kodna riječ, dakle, sadrži k informacionih simbola, sa dodatih $n-k$ paritetnih simbola.

- **Konvolucioni kodovi**

  Konvolucioni kodovi su kodovi za ispravljanje grešaka u kojima se poruka dijeli na simbole fiksne veličine, pri čemu svaki simbol sadrži m bitova. Korišćenjem unaprijed određenog algoritma, k uzastopnih simbola se kodira kako bi formirali kodnu riječ od n bitova gdje je $n > m$. Dakle, svaka kodna riječ zavisi od k prethodnih simbola. Kodne riječi se takođe nadovezuju jedna na drugu kako bi formirale neprekidan i teoretski beskonačan tok kodnih simbola.

Linearni blok kodovi kao što su pravougaoni i trougaoni kod sposobni su da detektuju i isprave jednu grešku u kodiranoj poruci, pa su korisni u jednostavnijim sistemima. Još jedan primjer linearnog blok koda je Hamingov kod. Hamingov kod se može koristiti za detekciju dvije i ispravljanje jedne greške u poruci. Zbog jednostavnosti Hamingovog koda, on se široko koristi u računarskoj memoriji, kompresiji podataka i drugim primjenama u telekomunikacijama, [6, 16]. Sa druge strane složeniji blok kodovi kao što su BCH i Reed-Solomon kodovi su u stanju da isprave više od jedne greške u kodiranoj poruci. BCH i Reed-Solomon kodovi su ciklični kodovi, što znači da imaju dodatnu osobinu da je svaka ciklična promjena riječi takođe kodna riječ. BCH kodovi se široko koriste u mobilnim komunikacijama, računarskim mrežama, satelitskoj komunikaciji, kao i u sistemima za pohranu podataka poput računarskih memorija ili kompaktnih diskova, [12, 15]. Reed-Solomon kodovi se koriste u komunikacionim sistemima i sistemima za pohranu zbog njihove sposobnosti da isprave serijske i nasumične greške. Neke od glavnih primjena Reed-Solomon kodova uključuju uređaje za pohranu poput kompakt diskova i DVD-ova, bežične i mobilne komunikacije, digitalnu televiziju, komunikacije putem električnih mreža, kao i brojne druge aplikacije kao što su medicinska dijagnostika i vojne svrhe, [17, 18]. Konvoluciono kodiranje, zajedno sa odgovarajućim algoritmom dekodiranja je tehnika za ispravljanje grešaka koja je posebno pogodna za kanale u kojima je preneseni signal uglavnom oštećen aditivnim bijelim Gaussovim šumom. Jedna od najčešće korišćenih tehnika dekodiranja konvolcionog koda je Viterbijev algoritam, jer ima fiksno vrijeme dekodiranja i dobro je prilagođen implementaciji u hardverskim dekoderima. Ova vrsta kodiranja se danas koristi u mobilnim telefonima i bežičnim komunikacijama, [11, 19, 20].

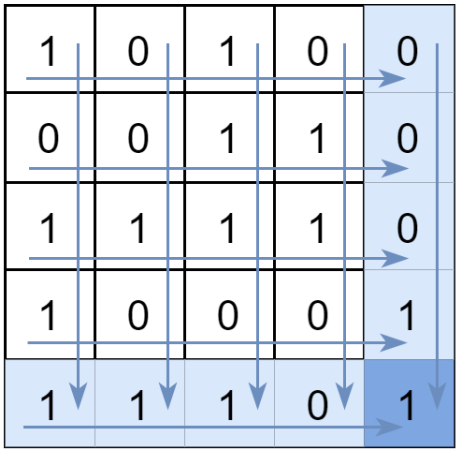
2.1 Pravougaoni kod

Pravougaoni kod spada u klasu linearnih blok kodova i u stanju je ispraviti jednu grešku u poruci, [3]. Pored informacionih, ovaj kod sadrži i kontrolne bitove (bitove parnosti), koji se generišu primjenom XOR operacije (sabiranje po modulu 2) nad odgovarajućim informacionim bitovima.

$\oplus$	0	1
0	0	1
1	1	0

Slika 4: Rezultat XOR operacije nad binarnim simbolima 0 i 1

Ovo je binarni kod dužine $n = pq$, čije su riječi napisane (ili se, barem, smatraju) kao matrica dimenzije $(p \times q)$. Dodaje se bit provjere parnosti svakom redu kao i svakoj koloni, transformišući originalnu matricu dimenzija $(p \times q)$ u matricu dimenzija $[(p+1) \times (q+1)]$, [3, 21]. Ovakav kod pripada klasi blok kodova, koji se obično označavaju kao (n,k). Oznaka n označava broj bita u kodnoj riječi, a oznaka k označava broj informacionih bitova u poruci. Broj bitova parnosti (redundantnih bita) obično se označava kao $m = n - k$, [5]. Na slici 3, prikazana je šema pravougaonog koda (25,16). Vidimo da imamo $k = 16$ informacionih bita, dok je ukupan broj bita u kodnoj riječi $n = 25$. Plavom bojom označena su polja u kojima su upisani biti parnosti. Posljednja kolona predstavlja provjere parnosti po vrstama, dok posljednja vrsta predstavlja provjere parnosti po kolonama.



Slika 5: Ilustracija pravougaonog koda (25,16)

Što se tiče posljednjeg bita (na slici 3 prikazan tamnijom bojom), on se može smatrati bitom za provjeru parnosti kako za posljednju kolonu tako i za posljednju vrstu. Kada je ovaj bit postavljen, sve vrste i sve kolone u matrici sadrže paran broj jedinica, čime se osigurava parnost za cijelu tabelu. Može se pokazati da je ovaj bit zapravo bit parnosti za sve informacione bitove. Konkretno, ukupan broj jedinica u bitovima parnosti u posljednjoj koloni (isključujući posljednji bit) je paran (ili neparan) ako je ukupan broj jedinica među informacionim bitovima paran (ili neparan). Isto pravilo važi i za posljednju kolonu. Stoga je ukupan broj bitova parnosti u posljednjoj vrsti i koloni (isključujući posljednji bit) uvijek paran, što znači da je provjera parnosti za cijelu tabelu ekvivalentna provjeri parnosti samo informacionih bitova, [5].

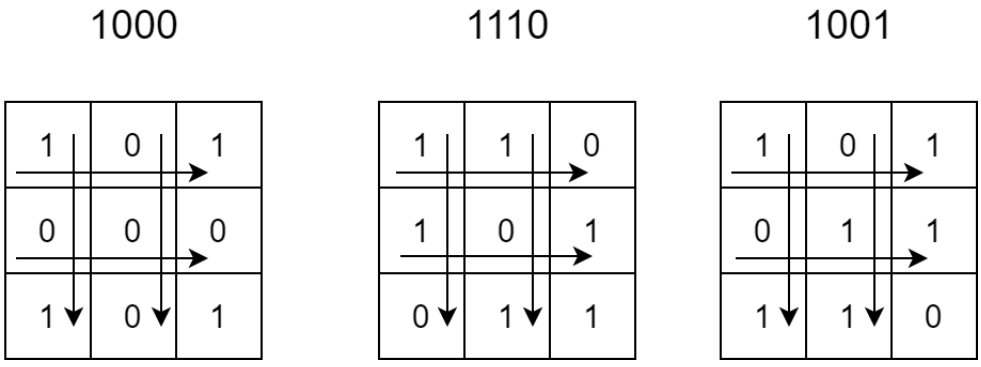
Odnos broja informacionih bita i broja bita u kodnoj riječi se naziva kodni odnos i on se računa po formuli, [3, 5]:

$$R = \frac{k}{n} \tag{1}$$

Kodni odnos može imati vrijednost između 0 i 1. Dvije ekstremne vrijednosti odgovaraju trivijalnim kodovima koji ne nose informacije ($R = 0$), i kodovima u kojima je svaka riječ kodna riječ koji ne pružaju zaštitu od šuma ($R = 1$). Generalno, kodni odnosi blizu 0 su tipični za kodove koji nisu veoma efikasni sa stanovišta vremena i/ili prostora koje zahtijevaju, ali često pružaju dobru zaštitu od šuma. Kodovi sa kodnim odnosom blizu 1 su veoma efikasni kodovi, ali često pružaju slabiju zaštitu od šuma, [3]. Kod ilustrovan na slici 3 ima kodni odnos : $R = 16/25 = 0.64$, odnosno 0.64 bita gubimo u poruci. Pokažimo sada na primjeru kako se pravougaonim kodom formiraju kodne riječi, a nakon toga ćemo unijeti greške u kodne riječi i ilustrovati na koji način se obavlja detekcija i korekcija greške.

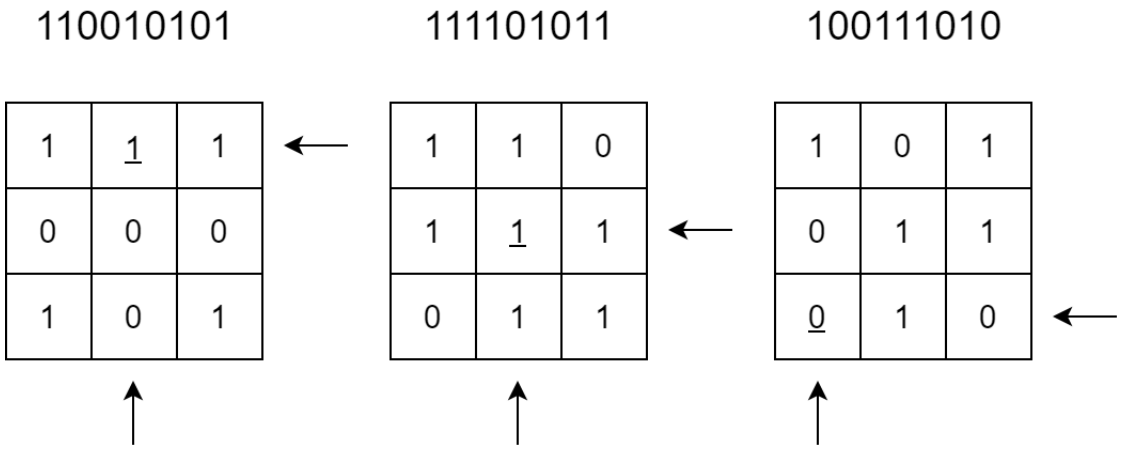
Primjer 1. Pravougaonim kodom (9,4) kodirati poruke:1000, 1110 i 1001, a zatim unijeti greške u kodirane poruke i izvršiti dekodiranje.

Izvršimo prvo kodiranje ove tri četvorke bita:



Slika 6: Kodiranje poruka: 1000, 1110 i 1001 pravougaonim kodom (9,4)

Pitanje je konvencije kojim će se redom slati bitovi kodnih riječi u kanal. Uzmimo da se na početku kodne riječi nalaze informacioni biti, a zatim biti parnosti. Kodne riječi za poruke: 1000, 1110 i 1001 su: 100010101, 111001011 i 100111110, respektivno. Pretpostavimo sada da su se prilikom prenosa kroz kanal javile greške u kodnim riječima, pa su se na ulazu dekodera našle poruke (podvučeni su bitovi na kojima se desila greška): 1$\underline{1}$0010101, 111$\underline{1}$01011, 100111$\underline{0}$10. Obavimo sada dekodiranje:

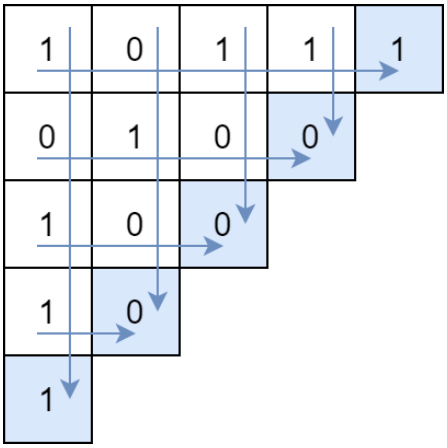


Slika 7: Dekodiranje poruka: 110010101, 111101011 i 100111010 pravougaonim kodom (9,4)

U prvoj tabeli, kontrolni bit u prvoj vrsti ukazuje na prisustvo greške u toj vrsti. Takođe, u ovoj tabeli kontrolni bit u drugoj koloni ukazuje na grešku u toj koloni, što znači da se greška desila na presjeku prve vrste i druge kolone. U drugoj tabeli, kontrolni biti ukazuju da se greška dogodila na presjeku druge vrste i druge kolone. U trećoj tabeli detektovana je greška u prvoj koloni, ali nije detektovana greška ni u jednoj od dvije vrste koje korespondiraju informacionim bitima. Međutim, na osnovu provjere parnosti u posljednjoj vrsti, možemo zaključiti da se greška dogodila na kontrolnom bitu u prvoj koloni. Takođe, na osnovu kontrolnog bita za čitavu tabelu možemo zaključiti da nema grešaka na informacionim bitima. Kako se radi o binarnom kodu, jednom kada nađemo poziciju greške, možemo je jednostavno ispraviti negacijom bita na poziciji na kojoj je detektovana greška. Preostaje nam da iz kodnih riječi izdvojimo informacione bite, kako bi dobili originalne poruke koje su se našle na ulazu kodera. Kako smo ranije definisali da se u kanal prvo šalju informacioni, a zatim kontrolni bitovi, na izlaz dekodera treba poslati prva četiri bita ispravljene kodne riječi. Nakon izdvajanja informacionih bita dobijamo poruke: 1000, 1110 i 1001, čime smo završili proces dekodiranja.

2.2 Trougaoni kod

Trougaoni kod, kao i pravougaoni, spada u klasu linearnih blok kodova koji su u stanju ispraviti jednu grešku u poruci. Kodna riječ je sastavljena od informacionih i kontrolnih bita. Kontrolni biti se generišu primjenom XOR operacije nad odgovarajućim informacionim bitima (Slika 2). Pretpostavimo sada da kanalom treba prenijeti sekvencu od deset informacionih bita. Kod pravougaonog koda, ove bite bi mogli rasporediti u matricu dimenzija (2×5), a kada dodamo još i kontrolne bite dobili bi matricu dimenzija (3×6), pa je riječ o pravougaonom kodu (18, 10), koji ima redudanciju od osam bita. Kodni odnos ovakvog koda iznosi $R = 10/18 = 0.55$. Formirajmo sada od ovih deset informacionih bita trougao sa redovima od po četiri, tri, dva i jednog informacionog bita, a zatim dodajmo i kontrolne bite.

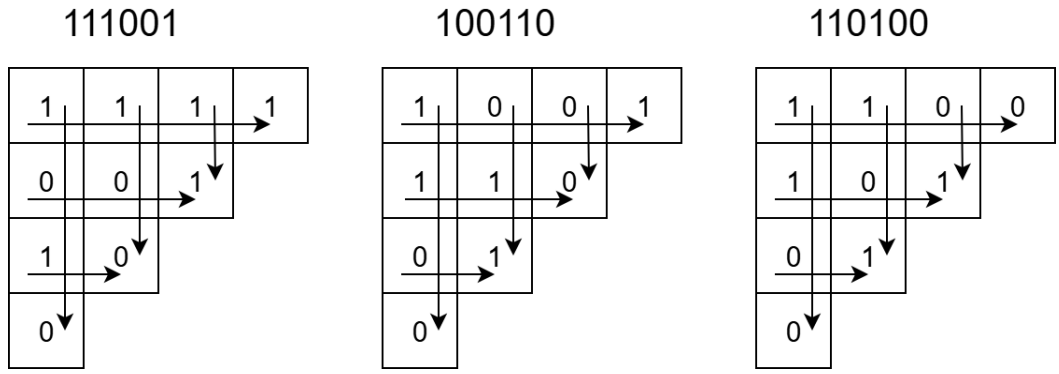


Slika 8: Ilustracija trougaonog koda (15,10)

Kod ovakvog trougaonog koda kontrolni biti se nalaze na hipotenuzi pravouglog trougla. Ovakva struktura je grafički prikazana na slici 6. Polja u kojima su upisani kontrolni biti označena su plavom bojom, dok strelice pokazuju koji kontrolni biti provjeravaju koje informacione. Dakle, svaki informacioni bit je provjeravan sa dva kontrolna bita, tako da se nedvosmisleno može utvrditi njegova koordinata. Kontrolni biti su provjeravani samo jednom. Kada jedna provjera parnosti nije zadovoljena, došlo je do greške na kontrolnom bitu, [5]. Na ovaj način ostvarena je mogućnost ispravke jedne greške u poruci od deset bita na osnovu dodavanja pet redudantnih bita, dok je u slučaju pravougaonog koda to ostvareno dodavanjem osam redudantih bita. Redukovanje redudancije ima dvije koristi: smanjuje se potrebna količina memorijskih resursa i smanjuje se vjerovatnoća javljanja više grešaka. Ovakav kod ima kodni odnos $R = 10/15 = 0.67$. Pokažimo sada na primjeru kako se trougaonim kodom formira kodna riječ i obavlja detekcija i korekcija greške.

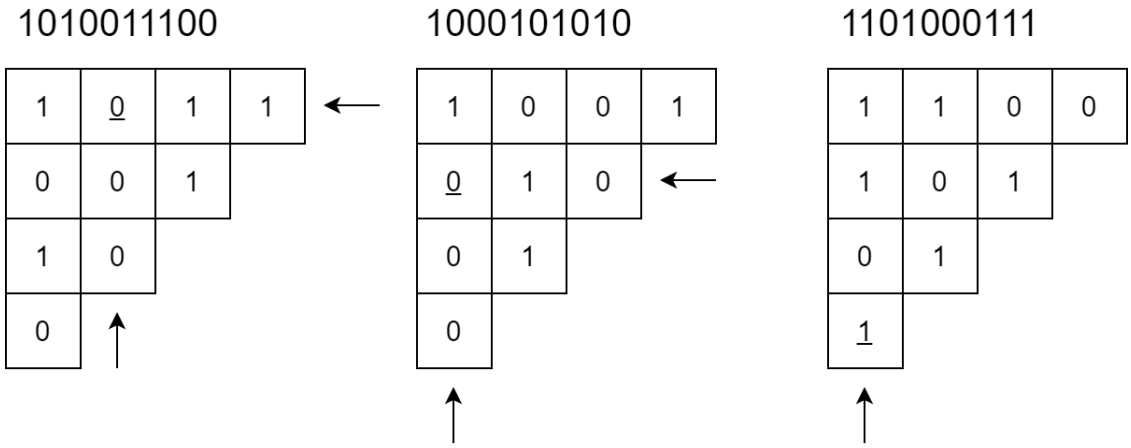
Primjer 2. Trougaonim kodom (10, 6) kodirati poruke: 111001, 100110 i 110100, a zatim unijeti greške u kodirane poruke i izvršiti dekodiranje

Izvršimo prvo kodiranje ove tri poruke:



Slika 9: Kodiranje poruka: 111001,100110 i 110100 trougaonim kodom (10, 6)

Pretpostavimo da se na početku kodne riječi nalaze informacioni, a zatim kontrolni biti. Kodne riječi za poruke 111001, 100110 i 110100 su: 1110011100, 1001101010 i 1101000110, respektivno. Pretpostavimo da je došlo do grešaka u prenosu kodiranih poruka, pa su primljene poruke (podvučene su pozicije na kojima su se dogodile greške): 1$\underline{0}$10011100, 100$\underline{0}$101010 i 1101000111. Izvršimo sada dekodiranje:



Slika 10: Dekodiranje poruka: 1010011100, 1000101010 i 1101000111 trougaonim kodom (10, 6)

Kod prve dvije poruke, dva bita parnosti ukazuju na grešku, pa je pozicija greške na presjeku ta dva bita parnosti. U trećoj poruci samo jedan bit parnosti ukazuje na grešku, pa zaključujemo da se greška dogodila upravo na tom bitu parnosti. Nakon ispravljanja grešaka negacijom bita na poziciji na kojoj je greška detektovana, ostaje samo da se izdvoje informacioni biti, čime je proces dekodiranja završen.

2.3 Hamingov kod

Hamingov kod je linearni kod za detekciju i korekciju greške nazvan po svom pronalazaču, Ričardu Hamingu. Haming je uveo koncept broja lokacija na kojima se dvije kodne riječi razlikuju i broja modifikacija neophodnih da se jedna kodna riječ pretvori u drugu u studiji objavljenoj 1950. godine. Ovaj koncept je danas poznat kao Hamingova distanca. Minimalna Hamingova distanca je fundimentalna granica za kodove i obilježava se sa d_{min}. Uloga minimalne distance d_{min} za detekciju i ispravljanje grešaka prikazana je u tabeli 1. Hamingov kod pripada grupi kodova koji se obično obilježavaju sa (n,k), gdje je n broj bita u kodnoj riječi, a k je broj informacionih bita. Broj redudantnih bita, tj. bita parnosti se obično obilježava sa m, [2, 14, 22–25].

Tabela 1: Uloga minimalne distance d_{min} za detekciju i ispravljanje grešaka

Detekcija do s grešaka po riječi	$d_{\min} \geq (s+1)$
Ispravljanje do t grešaka po riječi	$d_{\min} \geq (2t+1)$
Ispravljanje do t grešaka i detekcija $s > t$ grešaka po riječi	$d_{\min} \geq (t+s+1)$

Broj bita parnosti m, koji je potreban da bi se detektovala i ispravila jedna greška u nizu podataka dužine n je dat jednačinom:

$$m = \log_2 n + 1. \tag{2}$$

Još jedna verzija Hamingovog koda je prošireni Hammingov kod koji koristi još jedan dodatni bit parnosti. Ovakav kod je u stanju da detektuje dvije greške i ispravi jednu grešku u poruci. Dodatni bit parnosti se primjenjuje na sve bitove nakon što su dodati bitovi parnosti Hamingovog koda. Ovaj dodatni bit parnosti predstavlja paritet cijele kodne riječi. Ako se desi jedna greška, paritet se mijenja. Ako se dese dvije greške, paritet ostaje isti. Generalno, broj bita parnosti, m, potreban za detekciju dvije greške ili detekciju i ispravljanje jedne greške u nizu podataka dužine n, dat je sljedećom jednačinom, [5, 6, 14]:

$$m = \log_2 n + 2. \tag{3}$$

Opšti algoritam za Hamingov kod je sljedeći, [5, 14, 16]:

1. m paritetnih bitova se dodaje na podatke od k bita, formirajući kodnu riječ od $n = k + m$ bitova.

2. Pozicije bitova su numerisane redom od 1 do $k + m$.

3. One pozicije koje su numerisane kao stepeni broja dva, rezervisane su za bite parnosti, a preostale pozicije su za informacione bite.

4. Bitovi parnosti se računaju primjenom XOR operacije nad određenim kombinacijama informacionih bita. Kombinacije informacionih bitova su prikazane na slici 9.

5. Kako bi se detektovala greška potrebno je provjeriti sve bitove parnosti :

   - $C1$ = XOR bitova na pozicijama: (1, 3, 5, 7, 9, 11, 13, . . .)
   - $C2$ = XOR bitova na pozicijama: (2, 3, 6, 7, 10, 11, . . .)
   - $C4$ = XOR bitova na pozicijama: (4, 5, 6, 7, 13, . . .)
   - $C8$ = XOR bitova na pozicijama: (8, 9, 10, 11, 12, 13, . . .)

6. Ukoliko su sve provjere ($C1, C2, C4, C8$) jednake nuli, to znači da nema greške u kodnoj riječi. Međutim, ako je bilo koja od provjera jednaka jedinici, to ukazuje na prisustvo greške. Pozicija greške se može odrediti sabiranjem pozicija bitova parnosti koji ukazuju na grešku. Na primjer, ako su C1 i C2 jednaki jedan, a C4 i C8 jednaki nula, pozicija greške će biti 1 + 2 = 3. Ova metoda omogućava precizno lociranje i ispravljanje jedne greške unutar kodne riječi.

Pozicija / Paritet	1	2	3	4	5	6	7	8	9	10	11	12	13	(k+m)
	P1	P2	D1	P3	D2	D3	D4	P4	D5	D6	D7	D8	D9	...
P1	⊕		⊕		⊕		⊕		⊕		⊕		⊕	
P2		⊕	⊕			⊕	⊕			⊕	⊕			⊕
P3				⊕	⊕	⊕	⊕					⊕	⊕	⊕
P4								⊕	⊕	⊕	⊕	⊕	⊕	⊕
m														

Slika 11: Metod računanja bitova parnosti, pri čemu se oznake $P1, P2, P3$ i $P4$ odnose na paritetne bitove, a oznake $D1, D2, ..., D9$ na informacione bitove. Oznake k i m odnose se na broj informacionih bitova i broj bitova parnosti, respektivno.

Pokažimo sada na primjeru kako se Hamingovim kodom obavlja detekcija i korekcija greške:

Primjer 3. Proširenim Hamingovim kodom (8, 4) kodirati poruku 1010, a zatim unijeti grešku u kodiranu poruku i obaviti dekodiranje. Nakon toga demonstrirati kako se Hamingovim kodom obavlja detekcija dvije greške u poruci.

Kodna riječ je :

$$P_1\ P_2\ 1\ P_3\ 0\ 1\ 0\ P_4.$$

Prvi bit parnosti kontroliše bite na pozicijama: 1, 3, 5, 7. Drugi bit parnosti kontroliše bite na pozicijama: 2, 3, 6, 7. Treći bit parnosti kontroliše bite na pozicijama: 4, 5, 6, 7. Konačno, posljednji bit parnosti postavlja se tako da bude zadovoljena provjera parnosti za čitavu riječ. Sada dobijamo:

$$\begin{aligned} P_1 &= 1 \oplus 0 \oplus 0 = 1, \\ P_2 &= 1 \oplus 1 \oplus 0 = 0, \\ P_3 &= 0 \oplus 1 \oplus 0 = 1, \\ P_4 &= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0. \end{aligned}$$

Kodna riječ je : 10110100.
Pretpostaivmo da je u kanalu došlo do greške na poziciji 5, pa je dobijena poruka (podvučena je pozicija na kojoj je došlo do greške) : 1011$\underline{1}$100. Obavimo sada dekodiranje:

Potrebno je provjeriti sve bitove parnosti:

$$\begin{aligned} &\text{Pozicija 1:} \quad 1 \oplus 1 \oplus 1 \oplus 0 = 1 \quad \times \\ &\text{Pozicija 2:} \quad 0 \oplus 1 \oplus 1 \oplus 0 = 0 \quad \checkmark \\ &\text{Pozicija 4:} \quad 1 \oplus 1 \oplus 1 \oplus 0 = 1 \quad \times \\ &\text{Pozicija 8:} \quad 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 \oplus 0 = 1 \quad \times \end{aligned}$$

Bit na poziciji 8 ukazuje na to da je došlo do greške u kodnoj riječi. Kako provjera parnosti nije zadovoljena za bite na pozicijama 1 i 4, zaključujemo da je do greške došlo na poziciji 1+4=5. Negacijom bita na poziciji 5 dobijamo ispravnu kodnu riječ: 10110100. Nakon izdvajanja informacionih bita dobijamo poruku: 1010, čime je proces dekodiranja završen.

Pretpostavimo sada situaciju u kojoj su se desile dvije greške u kanalu i to na pozicijama 1 i 6, pa je kodna riječ koju treba dekodirati (podvučene su pozicije na kojima je došlo do greške): $\underline{0}$0110$\underline{0}$00. Izvršimo sada provjeru bitova parnosti:

$$\begin{aligned} &\text{Pozicija 1:} \quad 0 \oplus 1 \oplus 0 \oplus 0 = 1 \quad \times \\ &\text{Pozicija 2:} \quad 0 \oplus 1 \oplus 0 \oplus 0 = 1 \quad \times \\ &\text{Pozicija 4:} \quad 1 \oplus 0 \oplus 0 \oplus 0 = 1 \quad \times \\ &\text{Pozicija 8:} \quad 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 \oplus 0 = 0 \quad \checkmark \end{aligned}$$

Prva tri bita parnosti ukazuju na to da se greška dogodila (na poziciji 7, što je pogrešno), a posljednji bit ukazuje na to da nema greške. Iz ovakve situacije možemo zaključiti da su se desile dvije greške tokom prenosa.

2.4 BCH kod

BCH kodovi spadaju u klasu cikličnih kodova za ispravljanje slučajnih grešaka. Ovo su polinomijalni kodovi koji funkcionišu u Galoovim poljima (ili konačnim poljima). Dakle, BCH kodovi se oslanjaju na moćne i elegantne algebarske strukture koje se zovu konačna polja.

Polje je skup elemenata u kojem je moguće sabirati, oduzimati, množiti i dijeliti elemente polja, pri čemu se uvijek dobija drugi element unutar skupa. Konačno polje je polje koje sadrži konačan broj elemenata. Dobro poznat primjer polja je beskonačno polje realnih brojeva. Može se pokazati da skup cijelih brojeva {0, 1, 2, ..., p-1}, gdje je p prost broj, zajedno sa sabiranjem i množenjem po modulu p, formira polje. Takvo polje se zove konačno polje reda p, ili polje Galoa, u čast francuskom matematičaru Evaristu Galoau (fr. Evarist Galois). Galoovo polje reda p se često označava sa GF(p), [1, 8, 12, 26].

BCH kodovi mogu biti definisani sa dva parametra: veličinom kodne riječi n i brojem grešaka koje kod može ispraviti t.

$$\text{Dužina bloka: } n = 2^m - 1,$$
$$\text{Broj informacionih bitova: } k \geq n - mt,$$
$$\text{Minimalno rastojanje: } d_{\min} \geq 2t + 1,$$

gdje se parametar m odnosi na stepen Galoovog polja GF(2^m), a parametar k se odnosi na broj informacionih bitova.

Generatorski polinom koda je definisan u smislu njegovih korijena iz Galoovog polja GF(2^m). Neka je α primitivni element iz GF(2^m). Generatorski polinom $g(x)$ koda je polinom najnižeg stepena nad GF(2). Kodna riječ se dobija množenjem informacionog polinoma $i(x)$ i generatorskog polinoma $g(x)$. Neka je $m_i(x)$ minimalni polinom od α^i, tada se generatorski polinom $g(x)$ može izračunati kao, [5, 7]:

$$g(x) = m(x)m_3(x)m_5(x)...m_{2t-1}(x) = m(x)\sum_{i=2}^{t} m_{2i-1}(x). \tag{4}$$

U ovom radu se razmatra BCH (15, 5, 3) kod, gdje je n = 15, k = 5 i t = 3. Stoga, generatorski polinom sa $\alpha, \alpha^2, \ldots, \alpha^{2t}$ kao korijenima se dobija množenjem sljedećih minimalnih polinoma:

$$m_1(x) = 1 + x + x^4,$$
$$m_3(x) = 1 + x + x^2 + x^3 + x^4,$$
$$m_5(x) = 1 + x + x^2.$$

Zamjenjujući $m_1(x)$, $m_3(x)$ i $m_5(x)$ u jednačinu (4), dobija se generatorski polinom:

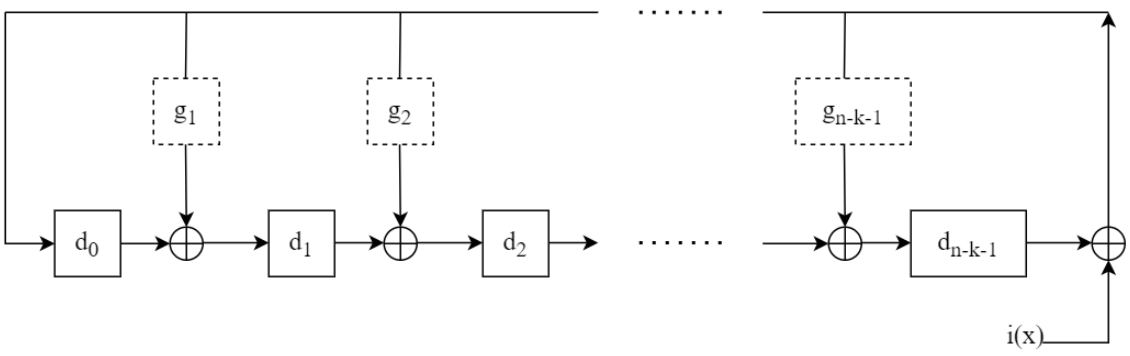
$$g(x) = 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}.$$

Da bismo konstruisali BCH kodove nad GF(2^4), treba da pronađemo elemente GF(2^4) generisane sa $p(x) = 1 + x + x^4$, kao što je dato u tabeli ispod, [7, 14] .

Tabela 2: Elementi GF(2^4) generisani polinomom: $p(x) = 1 + x + x^4$

Stepeni primitivnih elemenata	**Binarna reprezentacija**	**Polinimijalna forma**
α^0	0001	1
α^1	0010	α
α^2	0100	α^2
α^3	1000	α^3
α^4	0011	$\alpha + 1$
α^5	0110	$\alpha^2 + \alpha$
α^6	1100	$\alpha^2 + \alpha^3$
α^7	1011	$1 + \alpha + \alpha^3$
α^8	0101	$\alpha^2 - 1$
α^9	1010	$\alpha + \alpha^3$
α^{10}	0111	$1 + \alpha + \alpha^2$
α^{11}	1110	$\alpha^3 + \alpha + \alpha^2$
α^{12}	1111	$1 + \alpha + \alpha^2 + \alpha^3$
α^{13}	1101	$1 + \alpha^2 + \alpha^3$
α^{14}	1001	$1 + \alpha^3$
α^{15}	1	1

BCH kodovi se implementiraju kao ciklični kodovi, što znači da je digitalna logika koja implementira algoritme za kodiranje organizovana u linearni pomjerački registar koji imitira ciklična pomijeranja i polinomske aritmetike potrebne za opis cikličnih kodova, slika 10, [27].



Slika 12: Pomjerački registar za generisanje kodne riječi BCH (n, k) koda

Algoritam za dekodiranje BCH kodova sastoji se iz tri glavna koraka, [7]:

1. Računanje vrijednosti sindroma S_j, $j = 1,2,...,2t$,
2. Određivanje polinoma lokatora greške $\sigma(x)$,
3. Traženje korijena od $\sigma(x)$ i ispravljanje greške .

Pretpostavimo da je kodna riječ $c(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1}$ prenijeta i da su greške u prenosu rezultovale sljedećim primljenim vektorom: $r(x) = c(x) + e(x)$, gdje je $e(x)$ obrazac greške.

Dalje pretpostavimo da obrazac greške ima v grešaka na pozicijama $X^{l_1}, X^{l_2}, \ldots, X^{l_v}$, odnosno: $e(x) = x^{l_1} + x^{l_2} + \cdots + x^{l_v}$ gdje $0 \leq l_1 < l_2 < \cdots < l_v < n$. Pošto su $\alpha, \alpha^2, \ldots, \alpha^{2t}$ korijeni kodne riječi, tj. $c(\alpha^j) = 0$ za $1 \leq j \leq 2t$, onda važi: $r(\alpha^j) = e(\alpha^j)$, $j = 1,2,\ldots,2t$.

Prvi korak dekodiranja koda je računanje $2t$ sindromskih komponenti iz primljenog vektora $r(x)$. Ove sindromske komponente se mogu dobiti supstitucijom elemenata polja $\alpha, \alpha^2, \ldots, \alpha^{2t}$ u primljeni polinom $r(x)$, [12, 13].

$$S_j = r(\alpha^j) = e(\alpha^j). \tag{5}$$

Iz jednačine (5) vidimo da sindrom S zavisi samo od obrasca greške $e(x)$. Ukoliko su svi sindromi jednaki nuli, onda zaključujemo da nema greške u kodiranoj poruci.

Postoji više algoritama koji se mogu koristiti za računanje koeficijenata polinima lokatora greške $\sigma(x)$, a u narednom dijelu biće razmatran *Peterson-Gorenstein-Zierler* (PGZ) algoritam. PGZ algoritam sastoji se iz više koraka, [28]:

1. Naći najveći cijeli broj $i \leq t$ takav da je matrica M nesingularna:

$$M = \begin{pmatrix} S_1 & S_2 & \cdots & S_i \\ S_2 & S_3 & \cdots & S_{i+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_i & S_{i+1} & \cdots & S_{2i-1} \end{pmatrix}. \tag{6}$$

   Matrica M je singularna za $i > v$. Stoga, PGZ algoritam prvo računa broj grešaka v. Počevši od $i = t$ računa se determinanta det(M). Ako je det(M)=0, algoritam smanjuje matricu M sve dok determinanta matrice M bude nenulta, [29].

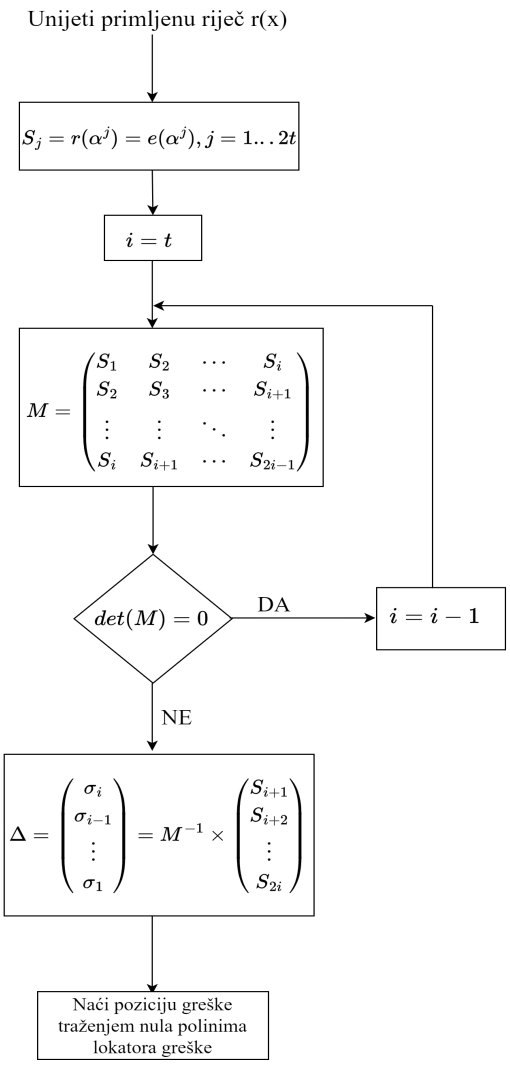
2. Naći inverznu matricu $M^{-1} = \frac{1}{det(M)} * adj(M)$ i sindromski vektor S:

$$S = \begin{pmatrix} S_{i+1} \\ S_{i+2} \\ \vdots \\ S_{2i} \end{pmatrix}. \tag{7}$$

3. Kako bi našli koeficijente polinoma lokatora greške potrebno je riješiti sistem: $\Delta = M^{-1} \times S$, gdje je Δ vektor koeficijenata polinoma lokatora greške, [30].

$$\Delta = \begin{pmatrix} \sigma_i \\ \sigma_{i-1} \\ \vdots \\ \sigma_1 \end{pmatrix} = M^{-1} \times S. \tag{8}$$

Na kraju, Chienova pretraga traži korijene polinoma lokatora greške, [1]. Računanje $\sigma(\alpha^i)$, za $i = 0..., n-1$ može se izvršavati paralelno koristeći jednostavne logičke operacije. Pozicija greške može se utvditi računanjem recipročne vrijednosti korijena polinoma lokatora geške. Ova vrijednost se dobija oduzimanjem stepena korijena polinoma lokatora greške od maksimalnog stepena n, [29]. Na slici 11 prikazana je šema algoritma za dekodiranje BCH koda.

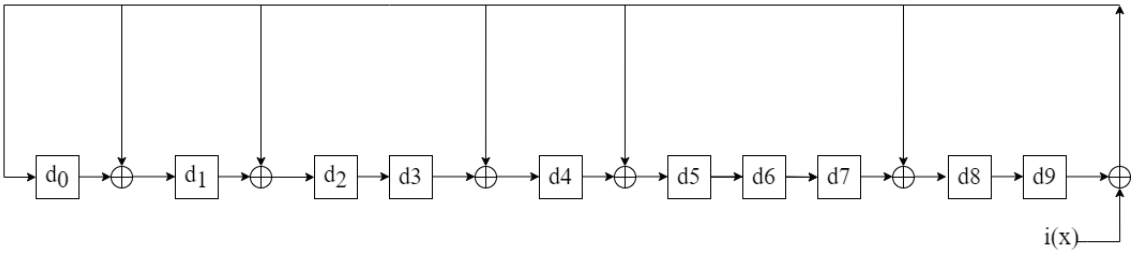


Slika 13: Algoritam dekodiranja BCH koda

Pokažimo sada na primjeru kako se obavlja detekcija i korekcija greške BCH kodom.

Primjer 4. BCH kodom (15, 5, 3) kodirati poruku: 01011. Nakon toga unijeti greške na pozicijama 4 i 10 u kodiranoj poruci i obaviti dekodiranje.

Linearni pomjerački registar za generisanje kodne riječi BCH (15, 5, 3) koda prikazan je na slici 12.



Slika 14: Linearni pomjerački registar za generisanje kodne riječi BCH (15, 5, 3) koda

Prvih pet bita u kodnoj riječi su informacioni biti. Nakon toga dodaju se kontrolni biti postavljajući ih u krajnjih lijevih $(n-k)$ ćelija registra. Sadržaji registra nakon prolaska svakog pojedinačnog informacionog bita prikazani su u tabeli 3.

Tabela 3: Sadržaji registra tokom generisanja BCH koda

Ulazna poruka	**Sadržaji registra**
11010	0000000000
1101	0000000000
110	1110110010
11	0111011001
1	0011101100
-	1111000100

Konačno, nakon prolaska svih informacionih bita kroz linearni pomjerački registar, formiramo kodnu riječ: 010110010001111. Nakon uvođenja grešaka na pozicije 4 i 10 u kodiranoj poruci dobijamo poruku (podvučene su pozicije na kojima je došlo do greške): 0100<u>1</u>0010<u>1</u>01111. Obavimo sada dekodiranje:

Primljena kodirana poruka se može zapisati kao:

$$r(x) = x^{13} + x^{10} + x^7 + x^5 + x^3 + x^2 + x^1 + 1.$$

Prvi korak u dekodiranju poruke BCH kodom jeste računanje sindroma S_j, $j = 1, 2, ..., 2t$, gdje je t broj grešaka koje kod može ispraviti. Ovo se može uraditi zamjenom elemenata polja GF(2^4) u primljeni polinom $r(x)$. Binarna reprezentacija ovih

elemenata data je u tabeli 2. Kako se radi o kodu koji može ispraviti tri greške, potrebno je izračunati šest sindroma:

$S_1 = \alpha^{13} + \alpha^{10} + \alpha^7 + \alpha^5 + \alpha^3 + \alpha^2 + \alpha + 1 = \alpha^3,$
$S_2 = \alpha^{11} + \alpha^5 + \alpha^{14} + \alpha^{10} + \alpha^6 + \alpha^4 + \alpha^2 + 1 = \alpha^6,$
$S_3 = \alpha^9 + 1 + \alpha^6 + 1 + \alpha^9 + \alpha^6 + \alpha^3 + 1 = \alpha^{14},$
$S_4 = \alpha^7 + \alpha^{10} + \alpha^{13} + \alpha^5 + \alpha^{12} + \alpha^8 + \alpha^4 + 1 = \alpha^{12},$
$S_5 = \alpha^5 + \alpha^5 + \alpha^5 + \alpha^{10} + 1 + \alpha^{10} + \alpha^5 + 1 = 0,$
$S_6 = \alpha^3 + 1 + \alpha^{12} + 1 + \alpha^3 + \alpha^{12} + \alpha^6 + 1 = \alpha^{13}.$

Iz dobijenih vrijednosti sindroma može se zaključiti da postoji makar jedna greška u dobijenoj poruci. Sljedeći korak jeste konstruisanje matrice M, dimenzija $(i \times i)$, počevši od $i = t$:

$$M = \begin{pmatrix} S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \\ S_3 & S_4 & S_5 \end{pmatrix} = \begin{pmatrix} \alpha^3 & \alpha^6 & \alpha^{14} \\ \alpha^6 & \alpha^{14} & \alpha^{12} \\ \alpha^{14} & \alpha^{12} & 0 \end{pmatrix}.$$

PGZ algoritam prvo računa broj grešaka u kodnoj riječi, a potom pozicije tih grešaka. Broj grešaka u kodnoj riječi je najveće $i \leq t$ za koje važi da je $\det(M) \neq 0$.

$$\det(M) = \begin{vmatrix} \alpha^3 & \alpha^6 & \alpha^{14} \\ \alpha^6 & \alpha^{14} & \alpha^{12} \\ \alpha^{14} & \alpha^{12} & 0 \end{vmatrix} = \alpha^3 \begin{vmatrix} \alpha^{14} & \alpha^{12} \\ \alpha^{12} & 0 \end{vmatrix} + \alpha^6 \begin{vmatrix} \alpha^6 & \alpha^{12} \\ \alpha^{14} & 0 \end{vmatrix} + \alpha^{14} \begin{vmatrix} \alpha^6 & \alpha^{14} \\ \alpha^{14} & \alpha^{12} \end{vmatrix} = 0.$$

Kako važi da je $\det(M) = 0$, za $i = 3$, zaključujemo da ne postoje tri greške u kodiranoj poruci. Sada dekrementiramo i za jedan i ponovo konstruišemo matricu M:

$$M = \begin{pmatrix} S_1 & S_2 \\ S_2 & S_3 \end{pmatrix} = \begin{pmatrix} \alpha^3 & \alpha^6 \\ \alpha^6 & \alpha^{14} \end{pmatrix}.$$

Tražimo determinantu ovakve matrice M:

$$\det(M) = \begin{vmatrix} \alpha^3 & \alpha^6 \\ \alpha^6 & \alpha^{14} \end{vmatrix} = \alpha^2 + \alpha^{12} = \alpha^7.$$

Najveće i, $i \leq t$ za koje je važi da je: $\det(M) \neq 0$ je 2. Stoga, zaključujemo da postoje dvije greške u kodiranoj poruci. Sada je potebno naći inverznu matricu M i sindromski vektor S:

$$M^{-1} = \frac{1}{det(M)} * adj(M),$$

$$adj(M) = \begin{pmatrix} \alpha^{14} & \alpha^{6} \\ \alpha^{6} & \alpha^{3} \end{pmatrix} \Rightarrow M^{-1} = \frac{1}{\alpha^{7}} * \begin{pmatrix} \alpha^{14} & \alpha^{6} \\ \alpha^{6} & \alpha^{3} \end{pmatrix} = \begin{pmatrix} \alpha^{7} & \alpha^{14} \\ \alpha^{14} & \alpha^{11} \end{pmatrix},$$

$$S = \begin{pmatrix} S_3 \\ S_4 \end{pmatrix} = \begin{pmatrix} \alpha^{14} \\ \alpha^{12} \end{pmatrix}.$$

Kako bi pronašli koeficijente polinoma lokatora greške potrebno je pomnožiti vektor S matricom M^{-1}:

$$\Delta = \begin{pmatrix} \sigma_2 \\ \sigma_1 \end{pmatrix} = M^{-1} \times S = \begin{pmatrix} \alpha^{7} & \alpha^{14} \\ \alpha^{14} & \alpha^{11} \end{pmatrix} \times \begin{pmatrix} \alpha^{14} \\ \alpha^{12} \end{pmatrix} = \begin{pmatrix} \alpha \\ \alpha^{3} \end{pmatrix}.$$

Sada možemo zapisati polinom lokator greške na sljedeći način:

$$\sigma(x) = 1 + \sigma_1 x + \sigma_1 x^2 = 1 + \alpha^3 x + \alpha x^2.$$

Korijeni polinoma $\sigma(x)$ su: α^{10} i α^4. Konačno, pozicije grešaka se mogu utvrditi računanjem recipročnih vrijednosti korijena polinoma lokatora greške, a to su α^5 i α^{11}. Sada možemo zapisati obrazac greške kao:

$$e(x) = x^{11} + x^5.$$

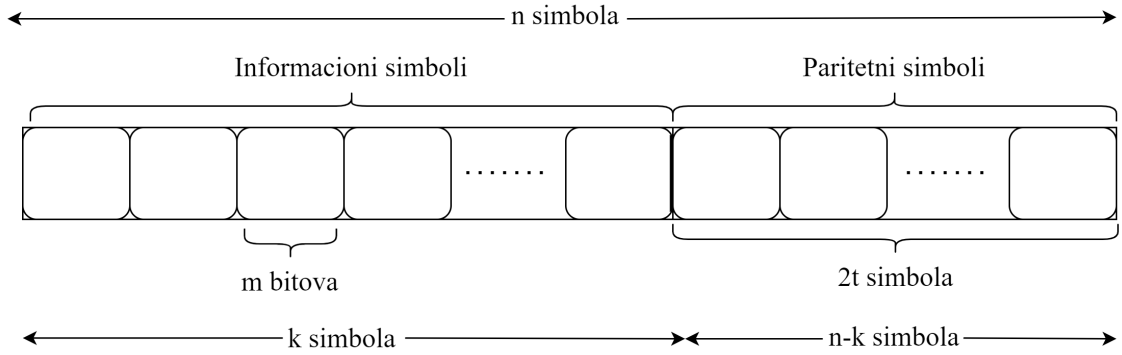
Ispravljena kodirana poruka se dobija na sljedeći način:

$$c(x) = r(x) + e(x) = x^{13} + x^{11} + x^{10} + x^7 + x^3 + x^2 + x^1 + 1.$$

Kodirana poruka $c(x)$ se može zapisati u binarnom obliku kao : 010110010001111. Nakon izdvajanja informacionih bita, dobijamo poruku: 01011, čime je proces dekodiranja završen.

2.5 Reed-Solomon kod

Reed-Solomon kodiranje je preslikavanje iz vektorskog prostora dimenzije m preko konačnog polja k u vektorski prostor veće dimenzije n preko istog polja k. Ukoliko se tokom prenosa ne dogodi više od $\frac{n-m}{2}$ grešaka, dekodiranjem je moguće u potpunosti ispraviti greške. Reed-Solomon kodovi su linearni blok kodovi i oni su potklasa nebinarnih BCH kodova. Ovakvi kodovi obično se označavaju kao RS (n, k, t), [4, 31]. Svaki simbol (blok bitova) ima m bitova. Ukupan broj simbola korišćenih u kodu se označava sa n, a broj simbola korišćenih za skladištenje korisnih informacija sa k. Tada, $m \times n$ predstavlja broj informacionih bitova koje treba kodirati. Preostali simboli, $(n-k)$, koriste se kao paritetni simboli. Za datu širinu simbola, n ima maksimalnu vrijednost od $2^m - 1$. Kapacitet korekcije, t, dat je brojem paritetnih simbola podijeljenim sa dva, tj. $2t = n - k$, [15, 18, 32–38].



Slika 15: Kodna riječ Reed-Solomon koda

Svaki blok poruke je ekvivalentan polinomu poruke stepena $(k-1)$, označen kao:

$$m(x) = m_0 + m_1 x + m_2 x^2 + m_3 x^3 + \cdots + m_{k-1} x^{k-1}, \tag{9}$$

gdje su koeficijenti $m_0, m_1, m_2, \ldots, m_{k-1}$ polinoma $m(x)$ simboli bloka poruke. Ovi koeficijenti su elementi GF(2^m). Dakle, sekvenca informacija se preslikava u apstraktni polinom tako što se koeficijenti postavljaju tako da budu jednaki vrijednostima simbola. RS koder koristi GF aritmetičke operacije za dodavanje paritetnih simbola. Galoovo polje je objašnjeno u prethodnoj sekciji. Objašnjenje za generisanje paritetnih simbola i polinoma kodne riječi (kodirani izlaz) je dato ispod, a više detalja je dostupno u literaturi [9, 17, 39].

1. Generatorski polinom se definiše na sljedeći način:

$$g(x) = g_0 + g_1 x + g_2 x^2 + \cdots + g_{2t-1} x^{2t-1} + x^{2t}, \tag{10}$$

gdje su $g_0, g_1, \ldots, g_{2t-1}$ koeficijenti iz GF(2^m).

2. Paritetni polinom (redundantni polinom) se definiše na sljedeći način:

$$p(x) = \left(x^{2t} \cdot m(x)\right) \bmod g(x), \tag{11}$$

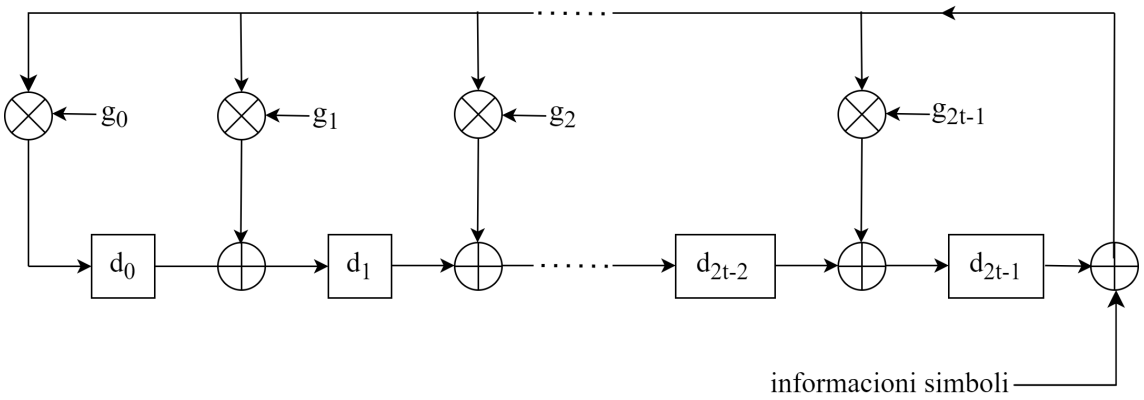
gdje je $g(x)$ generatorski polinom, a $m(x)$ polinom poruke.

3. Polinom kodne riječi je konačno:

$$c(x) = x^{2t} m(x) + p(x). \tag{12}$$

Kodiranje RS kodova se vrši u sistematskom obliku. Kodna riječ se formira jednostavno dodavanjem paritetnih (ili redundantnih) simbola na kraj k-simbola. Konkretno, kodna riječ sa k simbola u bloku poruke sastoji se od k uzastopnih koeficijenata polinoma poruke, dok su $2t$ paritetni simboli koeficijenti (iz GF(2^m)) redundantnog polinoma. Koristeći polinomsku notaciju, informacija se pomijera ulijevo množenjem sa x^{2t}, ostavljajući kodnu riječ u obliku definisanom jednačinom (12), [9, 32].

RS koder je u suštini linearni pomjerački registar sa $2t$ ćelija, pri čemu je svaka ćelija širine m bitova, slika 14. Koeficijenti kojima se množi su koeficijenti RS generatorskog polinoma. Opšta ideja je konstrukcija polinoma, čiji će koeficijenti biti simboli takvi da generatorski polinom dijeli polinom podataka bez ostatka, [37].

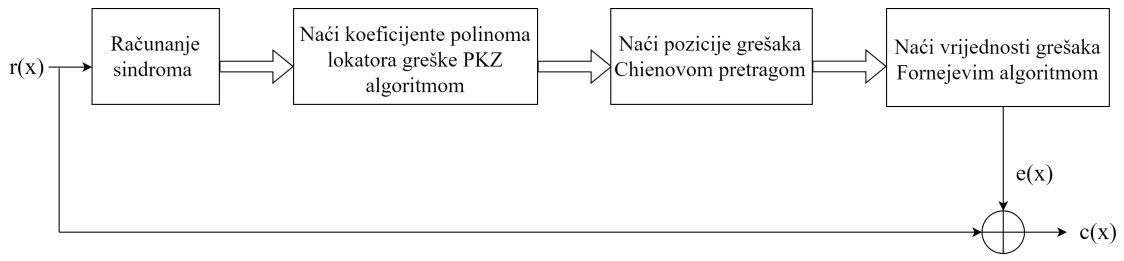


Slika 16: Linearni pomjerački registar za generisanje kodne riječi Reed-Solomon (n,k,t) koda

Kada RS dekoder ispravi simbol, on zamijeni netačan simbol tačnim, bilo da je greška prouzrokovana oštećenjem jednog bita ili svih bitova. Dakle, čak i ukoliko je simbol oštećen u svim svojim bitnim pozicijama, proces dekodiranja može ispravno rekonstruisati tačan simbol. Ovo daje RS kodovima ogromnu prednost u odnosu na binarne kodove kada je u pitanju otpornost na impulsne šumove, [34].

RS tehnika dekodiranja uključuje sljedeće korake, [31]:

1. Računanje vrijednosti sindroma,
2. Pronalaženje koeficijenata polinoma lokatora greške $\sigma(x)$,
3. Pronalaženje korijena polinoma $\sigma(x)$ i računanje lokacija grešaka,
4. Računanje vrijednosti grešaka,
5. Ispravljanje primljene kodne riječi.



Slika 17: Arhitektura Reed-Solomon dekodera

Poruka kodirana RS kodom se može zapisati kao:

$$c(x) = c_0 + c_1x + c_2x^2 + \cdots + c_{n-1}x^{n-1}. \tag{13}$$

Tada se primljeni polinom $r(x)$ može zapisati kao:

$$r(x) = r_0 + r_1x + r_2x^2 + \cdots + r_{n-1}x^{n-1}. \tag{14}$$

Primljeni polinom je povezan sa polinomom greške $e(x)$ i polinomom kodne riječi $c(x)$ na sljedeći način:

$$r(x) = c(x) + e(x), \tag{15}$$

gdje se obrazac greške $e(x)$ može zapisati kao:

$$e(x) = e_0 + e_1x + e_2x^2 + \cdots + e_{n-1}x^{n-1}. \tag{16}$$

Prvi korak RS dekodiranja je računanje $2t$ sindromskih komponenti:

$$S_i = r(\alpha^j), \quad j = 1,2,3,\ldots,2t. \tag{17}$$

gdje su $\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t}$ korijeni $g(x)$, elementi GF(2^m). Da bi polinom $r(x)$ bio validna kodna riječ, svaka sindromska komponenta S_j treba biti jednaka nuli. Ako bar jedna sindromska komponenta nije jednaka nuli, onda zaključujemo da postoji bar jedna greška u primljenoj poruci, [9].

Sljedeći korak je računanje polinoma lokatora greške $\sigma(x)$. Koeficijenti polinoma lokatora greške računaju se PKZ algoritmom, [30]. Nakon toga potrebno je naći korijene polinoma $\sigma(x)$ Chienovom pretragom, [4]. Postupak računanja koeficijenata i korijena polinoma lokatora greške detaljno je opisan u prethodnoj sekciji. Konačno, potrebno je izračunati vrijednosti grešaka $e_0, e_1, ..., e_i$, gdje je i broj grešaka u kodiranoj poruci. Postoji više algritama za računanje vrijednosti grešaka, a u narednom dijelu biće razmatran Fornejev algoritam, [15].

Prvi korak u Fornejevom algoritmu je konstruisanje matrice P, [4]:

$$P = \begin{pmatrix} y_1 & y_2 & \cdots & y_i \\ y_1^2 & y_2^2 & \cdots & y_i^2 \\ \vdots & \vdots & \ddots & \vdots \\ y_1^i & y_2^i & \cdots & y_i^i \end{pmatrix}, \tag{18}$$

gdje vrijednosti $y_1, y_2, ..., y_i$ predstavljaju korijene polinoma lokatora greške. Sada je potrebno riješiti Fornejev indentitet, [15]:

$$\begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_i \end{pmatrix} = P^{-1} \times \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_i \end{pmatrix}. \tag{19}$$

Sada obrazac greške možemo zapisati kao:

$$e(x) = e_0 + e_1 x + e_2 x^2 + \cdots + e_i x^i. \tag{20}$$

Ispravljenu kodiranu poruku dobijamo na sljedeći način:

$$c(x) = r(x) + e(x). \tag{21}$$

Pokažimo sada na primjeru kako se obavlja detekcija i korekcija greške Reed-Solomon kodom:

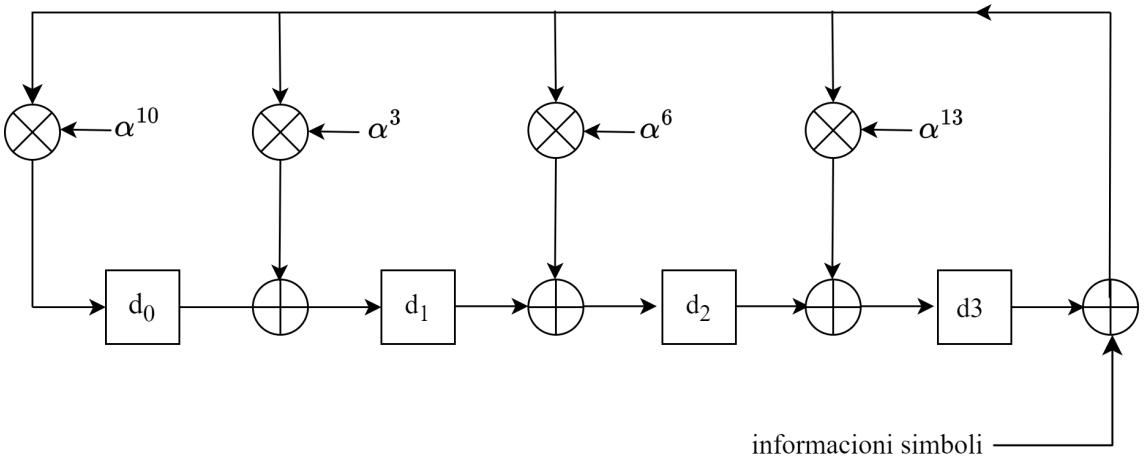
Primjer 5 Poruku $x + \alpha^6$ kodirati Reed-Solomon (15, 11, 2) kodom. Nakon toga, unijeti greške u kodiranu poruku i obaviti dekodiranje.

Generatorski polinom RS (15, 11, 2) koda je:

$$g(x) = \alpha^{10} + \alpha^3 x + \alpha^6 x^2 + \alpha^{13} x^3 + x^4,$$

gdje su $\alpha^0, \alpha^1, ..., \alpha^{15}$, elementi GF($2^4$) generisani polinomom: $1 + x + x^4$. Binarne reprezentacije ovih elemenata date su u Tabeli 2.

Predmetni Reed-Solomon koder koristi simbole dužine četiri bita i prima poruku dužine 44 bita (11 simbola). Ovakav koder na svom izlazu treba da generiše kodiranu poruku dužine 60 bita (15 simbola). Informacioni polinom: $m(x) = x + \alpha^6$ se može zapisati u binarnom obliku kao : 0000 0000 0000 ... 0001 1100. Dakle, prvih devet simbola imaju vrijednost 0000, dok su posljednja dva simbola opisana polinomom $m(x)$. Linearni pomjerački registar za generisanje opisanog RS koda prikazan je na slici 16, dok su sadržaji registra tokom generisanja kodne riječi dati u tabeli 4.



Slika 18: Linearni pomjerački registar za generisanje kodne riječi Reed-Solomon (15, 11, 2) koda

Tabela 4: Sadržaji registra tokom generisanja RS kodne riječi

Informacioni simboli	**Sadržaji registra**
$\alpha^0\ \alpha^6 0000000000000$	0000
$\alpha^0\ \alpha^6 000000000000$	0000
$\alpha^0\ \alpha^6 0000000000$	0000
$\vdots$	$\vdots$
$\alpha^0\ \alpha^6$	$\alpha^{10}\ \alpha^3\ \alpha^6\ \alpha^{13}$
α^0	$\alpha^{10}\ \alpha^{12}\ \alpha^2\ \alpha^0$

Dakle, početni sadržaji registra su 0000 i ovakvo stanje će ostati nepromijenjeno sve dok na ulaz linearnog pomjeračkog registra ne dođe prvi nenulti simbol. Nakon prolaska svih informacionih simbola kroz pomjerački registar, kodnu riječ možemo zapisati kao:

$$c(x) = x^5 + x^4\alpha^6 + x^3 + x^2\alpha^2 + x\alpha^{12} + \alpha^{10}.$$

Prvih jedanaest simbola kodne riječi, na pozicijama: $x^{14}, x^{13}, x^{12}, ..., x^4$, su informacioni simboli, dok su preostale pozicije namijenjene za redudantne simbole. Pretpostavimo da je došlo do grešaka tokom prenosa kodne riječi, pa je dobijena poruka (podvučene su pozicije na kojima je došlo do greške):

$$c(x) = x^5 + x^4\alpha^6 + \underline{x^3\alpha} + x^2\alpha^2 + \underline{x\alpha^{10}} + \alpha^{10}.$$

Prvi korak dekodiranja RS koda je računanje $2t$ sindromskih komponenti:

$S_1 = \alpha^5 + \alpha^{10} + \alpha^4 + \alpha^4 + \alpha^{11} + \alpha^{10} = \alpha^3,$
$S_2 = \alpha^{10} + \alpha^{14} + \alpha^7 + \alpha^6 + \alpha^{12} + \alpha^{10} = 1,$
$S_3 = 1 + \alpha^3 + \alpha^{10} + \alpha^8 + \alpha^{13} + \alpha^{10} = 1,$
$S_4 = \alpha^5 + \alpha^7 + \alpha^{13} + \alpha^{10} + \alpha^{14} + \alpha^{10} = \alpha^{14}.$

Sada je potrebno naći koeficijente polinoma lokatora greške upotrebom PKZ algoritma. Kako se radi o RS (15, 11, 2) kodu, početna vrijednost parametra i je 2, a matrica M će imati oblik:

$$M = \begin{pmatrix} S_1 & S_2 \\ S_2 & S_3 \end{pmatrix} = \begin{pmatrix} \alpha^3 & 1 \\ 1 & 1 \end{pmatrix}.$$

Tražimo determinantu ovakve matrice M:

$$\det(M) = \begin{vmatrix} \alpha^3 & 1 \\ 1 & 1 \end{vmatrix} = \alpha^3 + \alpha^0 = \alpha^{14}.$$

Kako determinanta matrice M ima nenultu vrijednost za $i = 2$, možemo zaključiti da postoje greške na dva različita simbola u primljenoj poruci. Sada računamo inverznu matricu M^{-1} i sindromski vektor S:

$$M^{-1} = \frac{1}{det(M)} * adj(M),$$

$$adj(M) = \begin{pmatrix} 1 & 1 \\ 1 & \alpha^3 \end{pmatrix} \Rightarrow M^{-1} = \frac{1}{\alpha^{14}} * \begin{pmatrix} 1 & 1 \\ 1 & \alpha^3 \end{pmatrix} = \begin{pmatrix} \alpha & \alpha \\ \alpha & \alpha^4 \end{pmatrix},$$

$$S = \begin{pmatrix} S_3 \\ S_4 \end{pmatrix} = \begin{pmatrix} 1 \\ \alpha^{14} \end{pmatrix}.$$

Množenjem matrice M^{-1} i sindromskog vektora S dobijamo koeficijente polinoma lokatora greške na sljedeći način:

$$\Delta = \begin{pmatrix} \sigma_2 \\ \sigma_1 \end{pmatrix} = M^{-1} \times S = \begin{pmatrix} \alpha & \alpha \\ \alpha & \alpha^4 \end{pmatrix} \times \begin{pmatrix} 1 \\ \alpha^{14} \end{pmatrix} = \begin{pmatrix} \alpha^4 \\ \alpha^9 \end{pmatrix}.$$

Sada polinom lokator greške možemo zapisati u obliku:

$$\sigma(x) = 1 + \sigma_1 x + \sigma_1 x^2 = 1 + \alpha^9 x + \alpha^4 x^2.$$

Chienovom pretragom traže se korijeni polinoma $\sigma(x)$ i utvrdjuju se lokacije grešaka. Korijeni polinoma lokatora greške su: α^{14} i α^{12}. Recipročne vrijednosti korijena polinoma $\sigma(x)$, tj. lokacije grešaka su: $y_1 = \alpha$ i $y_2 = \alpha^3$. Vrijednosti grešaka računaju se Fornejevim algoritmom. Prvi korak je konstruisanje matrice P:

$$P = \begin{pmatrix} y_1 & y_2 \\ y_1^2 & y_2^2 \end{pmatrix} = \begin{pmatrix} \alpha & \alpha^3 \\ \alpha^2 & \alpha^6 \end{pmatrix}.$$

Kako bi riješili Fornejev indentitet potrebno je izračunati matricu P^{-1}:

$$P^{-1} = \frac{1}{det(P)} * adj(P),$$

$$adj(P) = \begin{pmatrix} \alpha^6 & \alpha^3 \\ \alpha^2 & \alpha \end{pmatrix} \Rightarrow P^{-1} = \frac{1}{\alpha^{13}} * \begin{pmatrix} \alpha^6 & \alpha^3 \\ \alpha^2 & \alpha \end{pmatrix} = \begin{pmatrix} \alpha^8 & \alpha^5 \\ \alpha^4 & \alpha^3 \end{pmatrix}.$$

Vrijednosti grešaka se računaju na sljedeći način:

$$\begin{pmatrix} e_1 \\ e_2 \end{pmatrix} = P^{-1} \times \begin{pmatrix} S_1 \\ S_2 \end{pmatrix} = \begin{pmatrix} \alpha^8 & \alpha^5 \\ \alpha^4 & \alpha^3 \end{pmatrix} \times \begin{pmatrix} \alpha^3 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha^3 \\ \alpha^4 \end{pmatrix}.$$

Sada kada znamo lokacije i vrijednosti grešaka, polinom greške možemo zapisati kao:

$$e(x) = \alpha^3 x + \alpha^4 x^3.$$

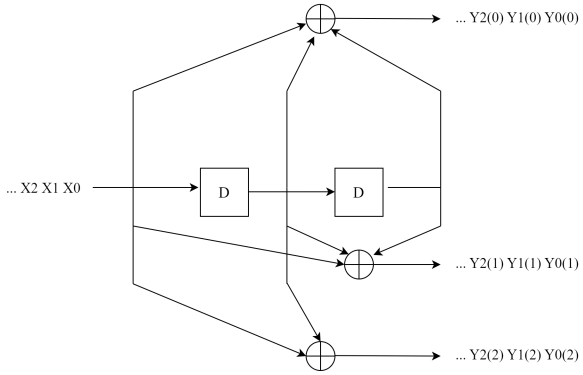
Originalna kodna riječ se dobija sabiranjem polinoma greške i primljenog polinoma:

$$c(x) = r(x) + e(x) = x^5 + x^4\alpha^6 + x^3 + x^2\alpha^2 + x\alpha^{12} + \alpha^{10}.$$

Nakon izdvajanja informacionih simbola dobijamo poruku: $x + \alpha^6$, čime je proces dekodiranja uspješno obavljen.

2.6 Konvolucioni kod i Viterbijev algoritam

Konvolucioni kod je vrsta koda za ispravljanje grešaka koja se značajno razlikuje od blok kodova. Prva razlika se ogleda u tome što kodna riječ nije sastavljena od odvojenih sekcija podataka i blokova, već su redudantni bitovi raspoređeni kroz kodirane podatke. Druga razlika je to što koder konvolucionog koda sadrži memoriju i n izlaza kodera u bilo kojem trenutku ne zavisi samo od k ulaza u tom trenutku, već i od m prethodnih ulaza, [40]. Ukoliko se informacioni bitovi direktno pojavljuju u kodiranoj poruci, onda govorimo o sistematskim kodovima. U suprotnom, imamo slučaj nesistematskog koda. Konvolucioni kodovi se obično označajavaju sa (n, k, m), gdje je n broj izlaznih elemenata (kodiranih), k je broj ulaznih elemenata (nekodiranih), a m je dubina memorije koda, što predstavlja broj ćelija registra. Ukupan broj bita koji utiče na generisanje jednog izlaznog bita naziva se ograničenje koda i označava se sa $L = (m + 1)$. Kodni odnos ($R = k/n$) predstavlja broj prenesenih bitova po ulaznom bitu. Na primjer, kodni odnos R=1/2 znači da se za svaki ulazni bit generišu dva bita na izlazu, [10, 11, 19, 20, 41]. Konvolucioni koder se implementira kao konačni automat, sa pomjeračkim registrom koji ima L-1 ćelija, slika 17.



Slika 19: Konvolucioni (3,1,2) nesistematski koder, pri čemu su sa X označeni informacioni biti na ulazu pomjeračkog registra, dok su sa Y označeni izlazni paritetni biti. Ćelije pomjeračkog registra označene su sa D.

Kodni odnos kodera prikazanog na slici 17 je R=1/3, a generatorski polinomi ovakvog kodera su:

$$G_1 = 1 + x + x^2,$$
$$G_2 = 1 + x + x^2,$$
$$G_3 = 1 + x.$$

Dakle, konvolucioni koder je Mealy-eva mašina, jer je izlaz funkcija trenutnog ulaza i trenutnog stanja. U Mealy-evoj mašini, postoji jedan ili više pomjeračkih registara i više XOR kapija. Povorka informacionih bitova ulazi u pomjerački registar sa jednog kraja i pomjera se na drugi kraj. XOR kapije su povezane sa nekim ćelijama pomjeračkih registara, kao i sa trenutnim ulazom, kako bi generisale izlaz, [11, 42].

Kodiranje konvolucionih kodova je trivijalan postupak, ništa složeniji od kodiranja blok kodova. Međutim, postupci dekodiranja su znatno složeniji. Postoje tri postupka za dekodiranje konvolucionih kodova: majoritetna logika, sekvencijalno dekodiranje i Viterbijev algoritam, [5]. U narednom dijelu biće razmatran Viterbijev algoritam. Viterbijev algoritam je poznat kao algoritam za dekodiranje konvolucionih kodova sa maksimalnom vjerovatnoćom za procjenu i pronalaženje najvjerovatnije putanje preživljavanja u trelis dijagramu, pritom ostvarujući mogućnost ispravljanja grešaka koje su se dogodile u prenosu. Ovaj algoritam se zasniva na računanju Hammingove distance za svaku granu, a putanja koja je najvjerovatnija kroz trelis će minimalizovati tu metriku. Za binarne kodere, odgovarajući trelis je sastavljen od $j = S$ nivoa sa 2^{L-1} čvorova u svakom nivou, gdje je S broj informacionih bitova, a L je ograničenje koda, [41, 43]. Za sekvencu od S informacionih bitova i koder sa memorijom m, algoritam se može podijeliti na sljedeća tri koraka (pretpostavljajući da je koder inicijalno u stanju svih nula na nivou $j = 0$), [19]:

1. Počevši od nivoa $j = m$, izračunati metriku za svaku pojedinačnu putanju koja ulazi u svako stanje kodera. Sačuvati putanju i njenu metriku za svako stanje. Ako postoje najmanje dvije putanje koje vode do istog stanja, kao preživjelu označiti onu sa najmanjom vrijednosti metrike;

2. Povećati nivo j za 1. Izračunati metriku za sve putanje koje ulaze u svako stanje dodavanjem metrike ulaznih grana metrički povezanog preživjelog iz prethodne vremenske jedinice. Za svako stanje identifikovati putanju sa najnižom metrikom kao preživjelu iz koraka 2. Sačuvati preživjelu putanju i njenu metriku;

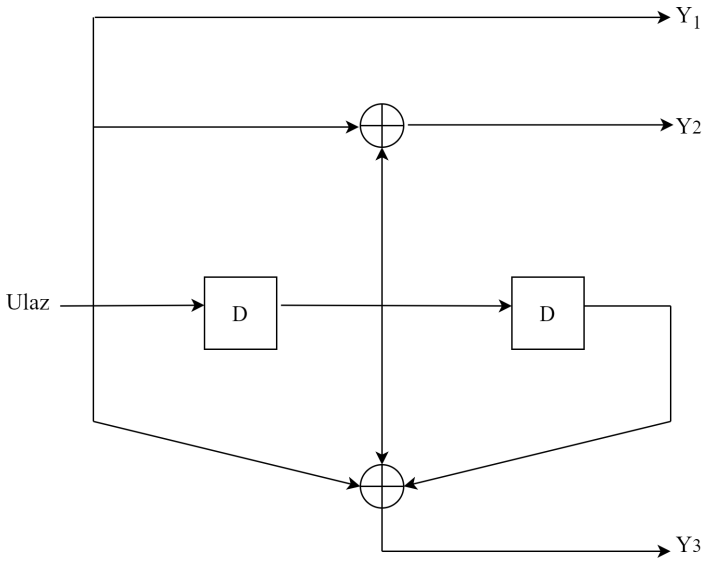
3. Ako je nivo $j < S$, ponovite korak 2. U suprotnom, zaustaviti algoritam i proći kroz trelis prateći najvjerovatniju putanju preživljavanja, kako bi se ispravile eventualne greške.

Velika prednost Viterbijevog algoritma je što (za kod sa ograničenjem L i kodnim odnosom $R = k/n$) broj operacija izvršenih pri dekodiranju S bitova iznosi $S2^{n(L-1)}$, što je linearno po S. Međutim, broj operacija izvršenih po dekodiranom bitu je eksponencijalna funkcija ograničenja L. Ova eksponencijalna zavisnost od L ograničava upotrebu Viterbijevog algoritma kao praktične tehnike dekodiranja na relativno kratke i ograničene kodove, [19, 44, 45].

Pokažimo sada na primjeru kako se generiše kodna riječ konvolucionim kodom i kako se obavlja proces dekodiranja Viterbijevim algoritmom.

Primjer 6 Sistematskim konvolucionim (3, 1, 2) kodom kodirati poruku 0110100. Zatim unijeti greške u kodiranu poruku i obaviti dekodiranje Viterbijevim algoritmom.

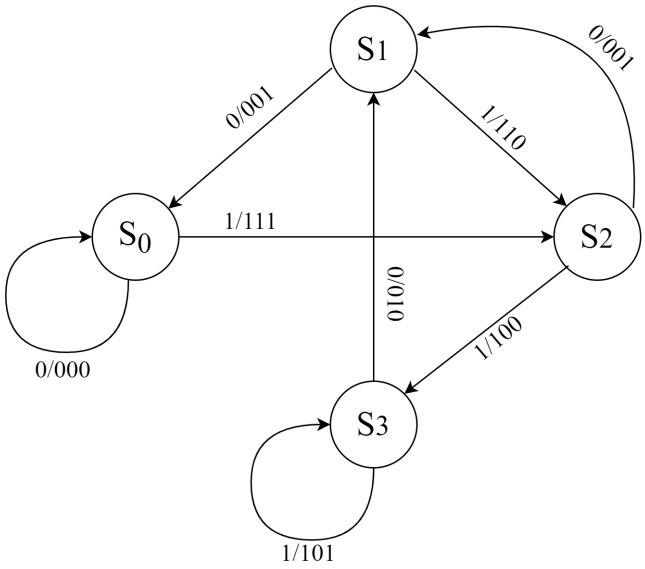
Predmetni kod ima kodni odnos $R = 1/3$, što znači da će se za svaki informacioni bit generisati tri bita na izlazu kodera. Kako se radi o sistematskom kodu, informacioni biti će se direktno pojavljivati u kodiranoj poruci. Memorijska dubina ovakvog kodera je $m = 2$, dok je ograničenje koda $L = 3$. Linearni pomjerački registar za generisanje kodne riječi konvolucionim (3, 1, 2) sistematskim kodom prikazan je na slici 18. Koder se inicijalno nalazi u stanju 00, tj. u stanju S_0. Informacioni bitovi dolaze jedan po jedan na ulaz pomjeračkog registra kako bi se generisali izlazni elementi dužine tri bita. Prvi bit svakog izlaznog elementa je informacioni bit, dok se preostala dva bita generišu primjenom XOR operacije nad odgovarajućim sadržajima registara. Kako je konvolucioni koder Mealy-eva mašina, on može biti opisan i tabelom stanja i dijagramom stanja, a izlaz zavisi od trenutnog ulaza i trenutnog stanja.



Slika 20: Konvolucioni (3,1,2) sistematski koder

Tabela 5: Tabela stanja sistematskog konvolucionog (3, 1, 2) kodera

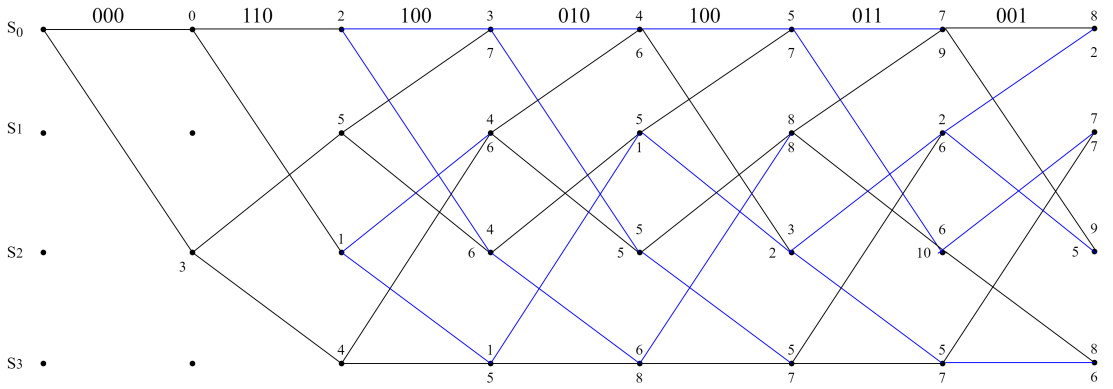
Ulaz	**Trenutno stanje**		**Sljedeće stanje**	**Izlaz**
0	S_0:	00	00	000
1		00	10	111
0	S_1:	01	00	001
1		01	10	110
0	S_2:	10	01	011
1		10	11	100
0	S_3:	11	01	010
1		11	11	101



Slika 21: Dijagram stanja sistematskog konvolucionog (3, 1, 2) kodera

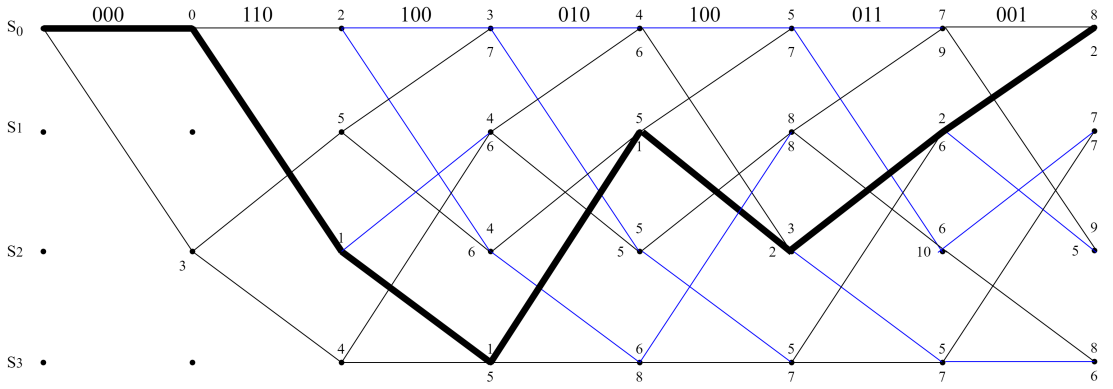
Nakon prolaska svih informacionih bitova kroz linearni pomjerački registar sa slike 18, kodirana poruka je : 000 111 100 010 110 011 001. Sada pretpostavimo da je došlo do grešaka prilikom prenosa, pa je primljena poruka (podvučene su pozicije na kojima je došlo do greške): 000 11$\underline{0}$ 100 010 1$\underline{0}$0 011 001. Obavimo dekodiranje:

Na slici 20 prikazan je trelis dijagram Viterbijevog algoritma za dekodiranje predmetnog koda. Metrika koja se koristi u Viterbijevom algoritmu je Hamingova distanca između primljenog elementa dužine tri bita i elementa koji se dobija prelaskom iz trenutnog stanja u sljedeće stanje. Za svaki čvor računamo vrijednost metrike tako što sabiramo vrijednosti metrika grana koje vode do tog čvora. Ukoliko dvije grane vode do jednog čvora, kao preživjelu označavamo onu koja ima manju vrijednost metrike. Preživjele grane su prikazane plavom bojom na slici 20.



Slika 22: Trelis dijagram za dekodiranje primljene poruke Viterbijevim algoritmom

Kako bi se ispravile greške u prenosu, potrebno pronaći najvjerovatniju putanju preživljavanja u trelis dijagramu. Ovo se postiže prolaskom kroz trelis dijagram unazad, počevši od čvora sa najmanjom metrikom na posljednjem nivou. Prilikom prolaska kroz trelis koriste se samo one grane koje su označene kao preživjele (osim za prvih m nivoa, koji imaju jedinstvene putanje). Na slici 21 prikazan je predmetni trelis dijagram, pri čemu je najvjerovatnija putanja preživljavanja označena podebljanim linijama.



Slika 23: Najvjerovatnija putanja preživljavanja u trelis dijagramu

Ispravljena kodna riječ se dobija praćenjem najvjerovatnije putanje preživljavanja u trelis dijagramu. Na prvom nivou trenutno stanje je S_0, a sljedeće stanje je takođe S_0, pa je odgovarajući element kodne riječi 000. Na drugom nivou prelazi se iz stanja S_0 u stanje S_2, što odgovara elementu 111. Ovaj postupak se ponavlja sve do posljednjeg nivoa, nakon čega možemo zapisati kodnu riječ kao: 000 111 100 010 110 011 001. Izdvajanjem informacionih bitova dobijamo poruku: 0110100, čime je proces dekodiranja završen.

3 Pregled postojećih rješenja

U ovom poglavlju analizirana su različita postojeća rješenja za implementaciju algoritama za detekciju i korekciju grešaka, korišćenjem FPGA tehnologije. Opisani su principi rada kodera i dekodera za različite vrste kodova, uključujući BCH, Reed-Solomon-ov, Hamming-ov i konvolucione kodove. Pored toga, razmatrane su metode za generisanje kodnih riječi, kao i specifične implementacije na FPGA platformama Xilinx i Altera. Algoritmi za detekciju i korekciju grešaka se izvršavaju na FPGA čipu, umjesto na računaru. Na taj način se dobija vrlo mali odzivni vremenski interval u poređenju sa odzivnim vremenom računara.

U [2] je implementiran Hammingov kod na Xilinx platformi koristeći metodu provjere parnosti. U realizaciji se 7 informacionih bita prenosi zajedno sa 4 redundantna bita. Poseban paritetni bit se koristi za detekciju parnog broja grešaka. Sekvenca informacionih bita se primjenjuje kao ulaz u kolo kodera, koji vrši ekskluzivno ili (XOR) operacije nad njom i na taj način generiše potrebne paritetne bite. Paritetni biti i informacioni biti zajedno formiraju kodnu riječ. Kolo dekodera se sastoji od generatora provjere bita, dekodera 3/8 i XOR kola. Kodna riječ se primjenjuje kao ulaz dekodera, a zatim se kontrolni biti generišu pomoću generatora provjere bita. Ti biti se dovode na ulaz dekodera 3/8 koji aktivira onu XOR kapiju koja ukazuje na poziciju greške. U zavisnosti od algoritma za ispravku, greška će biti ili detektovana ili ispravljena. Implementirani dizajn obavlja detekciju i korekciju greške za vrijeme trajanja jednog ciklusa taktnog signala trajanja 1 μs.

U [7] je implementiran BCH kod (15, 7) koji je u stanju da ispravi dvije greške u kodiranoj poruci. Sistem je simuliran korišćenjem Xilinx 12.1 ISE simulatora, dok je sinteza izvršena korišćenjem Cadence RTL kompajlera. Implementirani koder prima poruku dužine 7 bita i na svom izlazu generiše kodnu riječ dužine 15 bita. Koder je realizovan pomoću linearnog pomjeračkog registra i koristi serijsku ulaznu arhitekturu, pa je potrebno petnaest ciklusa taktnog signala kako bi se generisala jedna kodna riječ. Dekoder prima kodiranu poruku dužine 15 bita i na osnovu nje računa vrijendosti sindroma $s(x)$. Koeficijenti polinoma lokatora greške se računaju Petersonovim algoritmom, dok se nule polinoma lokatora greške, tj. pozicije greške, računaju Chien pretragom. Koristeći informacije o lokaciji grešaka, greške će biti ispravljene jednostavnim invertovanjem (pretvaranjem 1 u 0 i 0 u 1) na odgovarajućim lokacijama u primljenoj kodnoj riječi od 15 bita. Prednosti implementiranog dizajna uključuju nisku potrošnju energije (Slika 24) i efikasnu upotrebu FPGA resursa. Međutim, primjena serijske arhitekture dovodi do sporije obrade podataka, što rezultuje vremenom detekcije i korekcije grešaka od 100 ns.

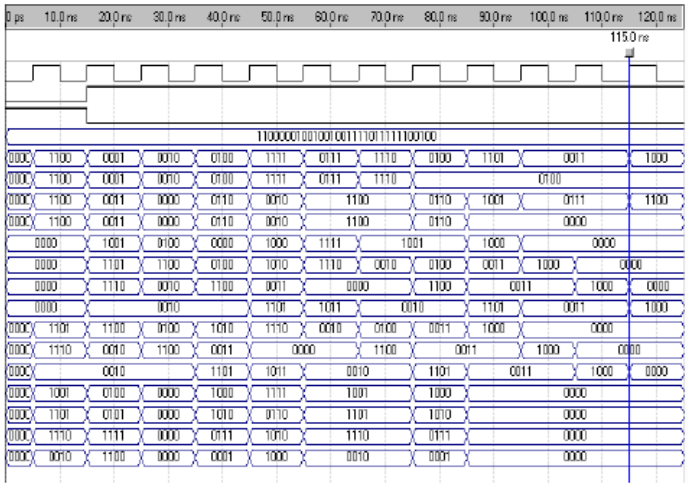
Naziv modula	Snaga (nW)	Površina (μm²)
Koder	151867.464	6961
Dekoder	477932.501	1051

Slika 24: Proračun snage i površine BCH (15,7) kodera i dekodera implementiranog na 180nm tehnologiji

U [8] je implementiran BCH algoritam na čipu EP1C12Q240C8 iz famiije Cyclone. Implementiran je BCH dekoder (15, 5, 3), koji je u stanju da ispravi 3 greške u kodiranoj poruci dužine 15 bita. Implementirani dekoder koristi serijsku ulaznu i serijsku izlaznu arhitekturu, pa je za detekciju i korekciju grešaka porebno 15 ciklusa taknog signala. BCH dekoder je implementiran pomoću kola za izračunavanje sindroma, Berlekamp-Massey algoritma i kola za Chien pretragu. Uprkos efikasnosti u korekciji grešaka, glavni nedostatak ovog dizajna je relativno dugo vrijeme obrade zbog serijske arhitekture, koje iznosi 1.09 μs. Međutim, ova mana je nadoknađena jednostavnošću dizajna, koji zahtijeva minimalne FPGA resurse. Implementacija koristi ukupno 187 logičkih elemenata, od kojih su 96 kombinaciona sa registrima, 12 isključivo registarska, a 79 kombinaciona bez registara.

U [12] implementiran je BCH (15, 5, 3) kod na FPGA čipu XC3S700A-4FG484. Implementirani dizajn prima informacionu poruku dužine 5 bita i generiše kodnu riječ dužine 15 bita. Dekoder je u stanju da ispravi do 3 greške u kodiranoj poruci. Kodna riječ se generiše pomjeranjem svih informacionih bita kroz linearni pomjerački registar. Kako dizajn koristi serijsku arhitekturu, za generisanje kodne riječi je potrebno 15 ciklusa taktnog signala. Dekoder prima kodiranu poruku dužine 15 bita i na svom izlazu generiše originalnu sekvencu informacionih bita, pritom ispravljajućći do 3 greške u kodiranoj poruci. Za računanje koeficijenata polinoma lokatora greške korišćen je PGZ algoritam, dok se korijeni polinoma lokatora greške traže Chienovom pretragom. Dekoder, takođe koristi serijsku ulaznu i serijsku izlaznu arhitekturu pa je i za proces dekodiranja potrebno 15 ciklusa taktnog signala, što odgovara vremenu od 10 μs. Prednost ovog dizajna je mala potrošnja FPGA resursa, koristeći samo 147 logičkih elemenata i 154 LUT-a.

U [9] je implementiran Reed-Solomon (12, 8) kod na FPGA čipu EP2S60F672C3. Kodna riječ ovakvog koda sačinjena je od 12 simbola, od kojih su 8 informacioni simboli, a 4 kontrolni simboli. Svaki simbol sačinjen je od 4 bita, a ovakav kod je u stanju da ispravi grešku na 2 simbola, tj. 8 bita. Koder je realizovan pomoću linearnog pomjeračkog registra i potrebno je dvanaest ciklusa taktnog signala kako bi se generisala jedna kodna riječ (slika 25).



Slika 25: Rezlutati simulacije implementiranog kodera

Dekoder je sačinjen od više modula, a to su: modul za računanje sindroma, modul za traženje koeficijenata polinoma lokatora, modul za traženje pozicija grešaka i modul za računanje vrijednosti grešaka. Za detekciju i korekciju greške potrebno je 12 ciklusa taktnog signala (ukupno 120 ns). Implementirani Reed-Solomon (12,8) dekoder karakteriše visoka brzina obrade, ali uz značajnu složenost i veliku potrošnju FPGA resursa. Dizajn koristi 12,484 ALU jedinica i 12,484 registara, što omogućava brzu i efikasnu obradu podataka, ali povećava hardversku zahtijevnost.

U radu [10] autori predstavljaju dizajn i implementaciju konvolucionog kodera i Viterbijevog dekodera na FPGA uređaju Spartan 3. Implementacija je izvedena korišćenjem Verilog HDL jezika, a funkcionalna verifikacija i simulacija obavljene su pomoću ModelSim 10.1b, dok je sinteza izvršena na Xilinx ISE platformi. Konvolucioni koder realizovan je pomoću pomjeračkih registara i mod-2 sabirača, čime se implementira proces kodiranja sa kodnim odnosom $R = 1/2$ i ograničenjem koda $L = 3$. Ovakav koder generiše dva izlazna bita za svaki ulazni bit, što omogućava povećanu otpornost na šum u komunikacionim kanalima. Struktura kodera opisana je pomoću tabele stanja, dijagrama stanja i trelis dijagrama. Viterbijev dekoder implementiran je kako bi se omogućila efikasna detekcija i korekcija grešaka na prijemnoj strani sistema. Arhitektura dekodera sastoji se od jedinice za metriku grane , jedinice za računanje metrika puta i memorije za preživljavanje. Dekodiranje se realizuje determinističkim pristupom, pri čemu se primljeni podaci upoređuju sa idealnim izlazima kodera koristeći Hamingovu ili Euklidsku udaljenost. Sinteza dizajna na FPGA platformi pokazala je da implementacija koristi minimalnu količinu resursa, što omogućava povećanu efikasnost i brže procesiranje signala. Za proces kodiranja potrebno je 1.5 ns, dok je za proces dekodiranja potrebno 400 ns. Analiza iskorišćenosti FPGA resursa pokazuje da dizajn koristi samo 1% dostupnih

logičkih elemenata i 2% 4-input LUT-ova, što potvrđuje njegovu visoku efikasnost i minimalnu potrošnju resursa.

U [11] implementiran je konvolucioni algoritam na Xilinix platformi. Realizovani su konvolucioni koderi sa kodnim odnosima 1/2 i 1/3 pomoću Finite state machine (FSM) pristupa. Konvolucini koder sa kodnim odnosom 1/2 generiše dva bita na izlazu, za svaki ulazni bit, dok koder sa kodnim odnosom 1/3 generiše tri bita na izlazu za svaki bit na ulazu. Prikazan je i trelis dijagram sa 4 stanja, na kojem se vide tranzicije za svako stanje. U radu su date i RTL šeme realizovanih kodera kao i rezlutati simulacija. Konvolucioni koderi implementirani su na FPGA uređaju Spartan 3E, pri čemu je za proces kodiranja koderom sa kodnim odnosom 1/2 porebno 4ns, dok je za koder sa kodnim odnosom 1/3 potrebno 7.8ns. Dizajn ostvaruje minimalnu upotrebu FPGA resursa, što se može videti u tabelama 6 i 7 koje pokazuju zanemarljivo iskorišćenje logičkih elemenata (Flip-Flop-ova i LUT-ova), dok je veća potrošnja uočena kod I/O resursa i globalnih taktnih ciklusa kod kodnog odnosa 1/3.

Tabela 6: Poređenje iskorišćenosti FPGA resursa za kodne odnose 1/2 i 1/3

Resurs	**Kod. odnos 1/2**	**Kod. odnos 1/3**
Odabrani uređaj	3S100EVQ100-5	
Broj logičkih blokova	2 od 960 (0%)	2 od 960 (0%)
Broj Flip-Flop-ova	4 od 1920 (0%)	3 od 1920 (0%)
Broj 4-input LUT-ova	2 od 1920 (0%)	2 od 1920 (0%)
Broj I/O pinova	4	6
Broj bonded IOBs	4 od 66 (6%)	6 od 66 (9%)
Broj GCLK-ova	1 od 24 (4%)	1 od 24 (4%)

U [36] implementiran je Reed-Solomon (15, 9, 3) kod na FPGA čipu EPF10K30RI2404. Koder je ralizovan pomoću linearnog pomjeračkog registra i potrebno je 9 ciklusa taktnog signala za generisanje kodne riječi. Dekoder na svom ulazu prima kodnu riječ dužine 15 simbola i na svom izlazu generiše originalne informacione simbole, pritom ispravljajući greške na najviše 3 simbola u kodiranoj poruci. Za računanje koeficijenata polinoma lokatora greške korišćen je Euklidov algoritam, dok je za računanje vrijednosti grešaka korišćen Fornejev algoritam. Za proces kodiranja potrebno je 10 ns, dok je za detekciju i korekciju grešaka potrebno 5 ciklusa taktnog singla, što odgovara vremenu od 50 ns. Kompletan dizajn koristi 1728 logičkih elemenata i 189 I/O pinova.

U [41] implementiran je konvolucioni koder (2, 1, 7) i odgovarajući Viterbijev dekoder na FPGA čipu XC2V2000fg676 iz familije Xinlinx virtex2 . Implementira-

ni koder ima kodni odnos 1/2, što znači da se za svaki bit na ulazu generišu dva bita na izlazu kodera. Koder se sastoji od šest pomjeračkih registara i dvije XOR kapije . Pri početku kodiranja, svi ovi registri se resetuju radi inicijalizacije. Svaki pomijerački registar je ekvivalentan flip-flopu. Ovih šest flip-flopova su povezani u seriju kako bi se izvršavala operacija pomjeranjaa i ažuriranja pod uticajem taktnog signala. XOR kapije se koriste kao sabirači za dobijanje kodiranih podataka. Predloženi Viterbijev dekoder funkcioniše tako što dva binarna bita ulaze sa svakim taktnim impulsom, a modul za računanje Hamingove distance računa 64 različite grupe distanci. *Add-Compare-Select* (ACS) moduli dodaju prethodnu akumuliranu distancu novom putu i biraju manju vrijednost kao preživjeli put. Preživjeli putevi svih stanja se čuvaju u RAM-u. Kada se dostigne dubina dekodiranja, bira se minimalna akumulirana distanca, koja se prosljeđuje modulu za vraćanje unazad, koji daje konačne dekodirane rezultate. Za proces detekcije i korekcije grešaka potrebno je 550 ns. Prednost ovog dizajna je niska potrošnja FPGA resursa, kako dizajn koristi samo 2.94 % ukupnih logičkih elemenata.

U [16] implementiran je Hammingov kod na FPGA čipu XC3S700A iz porodice Spartan 3A, korišćenjem VHDL koda. Arhitektura je simulirana sa različitim tehnologijama (16nm, 22nm, 32nm i 45nm) uz pomoć TANNER EDA alata radi proučavanja ukupne disipacije snage kola. Analiza pokazuje da sa smanjenjem dužine kanala dolazi do smanjenja disipacije snage od 12,65 % i 2,37 % u 16nm tehnologiji u poređenju sa 22nm, kod kola kodera i dekodera. Koder prima poruku dužine četiri bita i primjenom XOR operacije nad odgovarajućim bitima generiše kodnu riječ dužine sedam bita. Dekoder na svom ulazu prima kodnu riječ i u stanju je da tektektuje i ispravi jednu grešku u kodiranoj poruci. Generišu se kontrolni bitovi pomoću generatora kontrolnih bitova kako bi se provjerili paritetni bitovi. Ti kontrolni bitovi lociraju grešku u kodnoj riječi pomoću kola dekodera. Pozicija greške se utvrđuje provjerom kontrolnih bitova, dok se greška ispravlja jednostavnom inverzijom bita na poziciji na kojoj je greška detektovana. Za proces detekcije i korekcije greške imoplementiranim dizjanom potrebana je 1 μs. Glavna prednost ovog dizajna ogleda se u veoma maloj potrošnji energije i FPGA resursa (tabela 7).

Tabela 7: Rezime simulacije kodera i dekodera

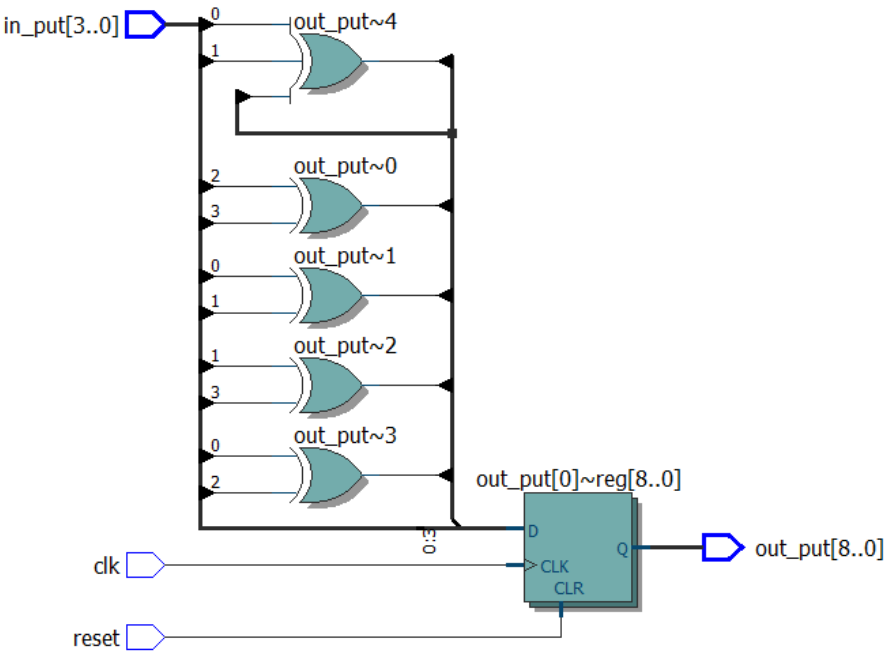
	Iskorišćenost uređaja		
	Br. logičkih blokova	**Br. 4-ulaznih LUT-ova**	**Br. povezanih IOB-ova**
Koder	3	5	20
Dekoder	11	21	20

4 Hardverska implementacija algoritama za detekciju i korekciju greške

U okviru ove sekcije biće dizajniran i projektovan hardver za realizaciju odabranih, najčešće korišćenih algoritama za detekciju i korekciju greške. Svi kodovi će biti napisani u VHDL programskom jeziku, u okviru Quartus II razvojnog okruženja, i implementirani na FPGA čipu EP3C5E144C7 iz familije Cyclone III. Funkcionalnost implementiranih algoritama biće testirana kroz simulacije pomoću ModelSim Altera alata, gdje će se generisati različiti testni slučajevi za verifikaciju njihovog rada. Sva rješenja će biti evaluirana u pogledu performansi, uključujući brzinu, korišćenje resursa čipa na kome je izvršena implementacija, kašnjenje sistema, kao i zahtijevanu snagu. Netlist Viewer alat će se koristiti za vizualizaciju generisanih netlisti, omogućavajući detaljno ispitivanje i analizu strukture dizajna, dok će se PowerPlay Power Analyzer alat koristiti za analizu upotrijebljene snage.

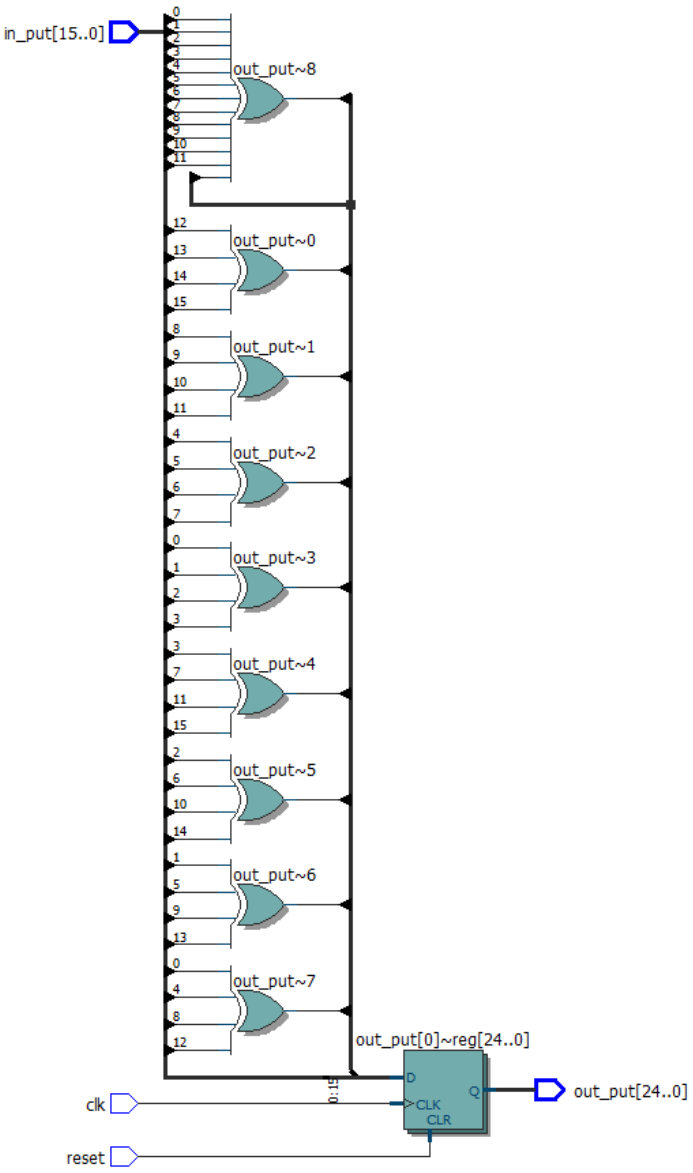
4.1 Pravougaoni kod

Pravougaoni koder (9,4) realizovan je tako da se na izlaz prvo šalju informacioni, a zatim kontrolni biti. Signal in_put predstavlja ulaznu sekvencu informacionih bita, dok signal out_put predstavlja izlaznu kodiranu poruku. Za slučaj pravougaonog kodera (9, 4), ulazni signal in_put je dužine četiri bita.

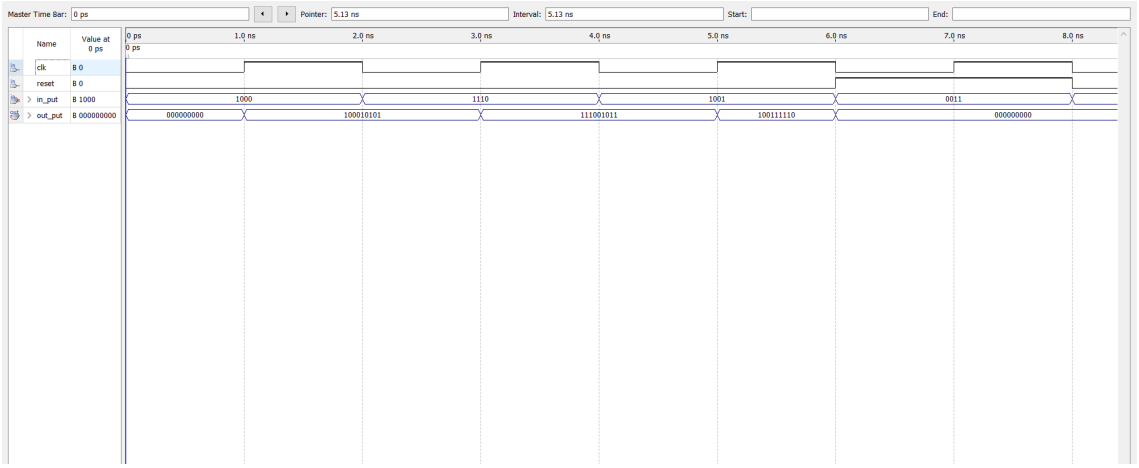


Slika 26: RTL šema pravougaonog kodera (9, 4)

Odgovarajući informacioni biti dovode se na ulaze XOR kapija, kako bi se generisali kontrolni biti. Na slici 22, pored XOR kapija, postoji i jedan D flip-flop, koji osigurava sinhronizaciju izlaznog signala *out_put* sa kontrolnim signalima *clk* i *reset*. Na prvi ulaz D flip-flopa dovodi se kodirana poruka, dok se na preostala dva ulaza dovode kontrolni signali *reset* i *clk*. Signal *clk* predstavlja taktni signal i kodirana poruka će se proslijediti na izlaz flip-flopa na svakoj uzlaznoj ivici ovog signala. Signal *reset* se koristi za postavljanje izlaznog signala *out_put* u poznato stanje. Kada je ovaj signal na visokom nivou, izlaz se resetuje na logičku nulu. Za generisanje jednog kontrolnog bita potrebna je jedna XOR kapija, pa bi u slučaju pravougaonog kodera (25,16) imali devet ovakvih kapija (slika 23).



Slika 27: RTL šema pravougaonog kodera (25, 16)



Slika 28: Rezultati simulacije za pravougaoni koder (9, 4)

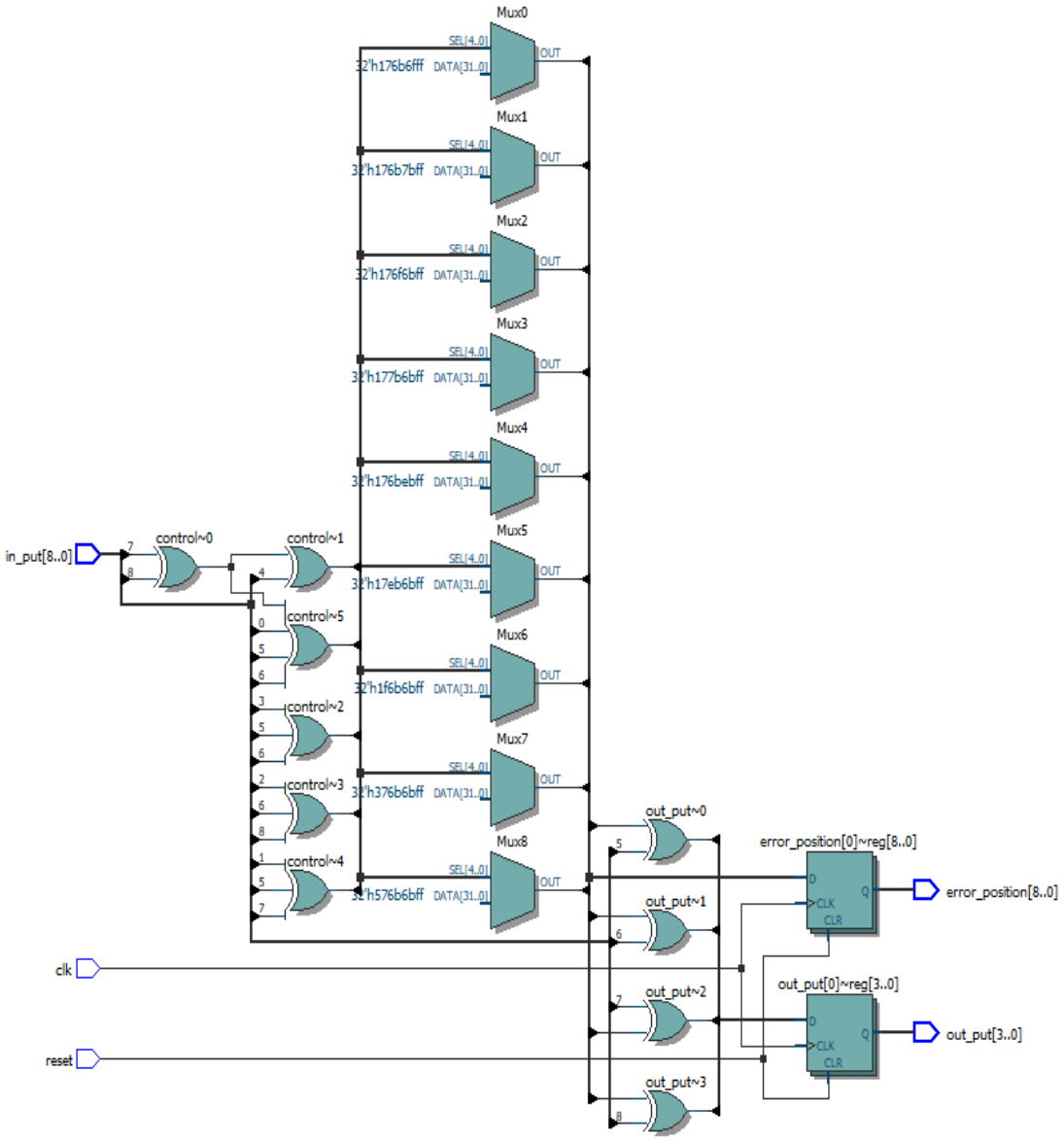
Kako bi se verifikovala tačnost implementiranog kodera, generisano je više ulaza koji su testirani kroz simulaciju, slika 24. Testni podaci su isti kao oni korišćeni u primjeru 1, poglavlje 2. Izlazni signal out_put se generiše na svakoj uzlaznoj ivici taktnog signla clk, a ulazni podaci se učitavaju na svakoj silaznoj ivici ovog signala. Nakon trećeg ciklusa taktnog signala, izlaz je postavljen na logičku nulu jer je signal $reset$ u visokom stanju, tj. na logičkoj jedinici.

Rezultati analize snage pokazuju da ukupna termička disipacija snage iznosi 56.38 mW. Ova vrijednost se sastoji od nekoliko komponenti: dinamička termička disipacija jezgra je 0.00 mW, što znači da nema dodatne disipacije usljed promjena u logičkim stanjima. S druge strane, statička termička disipacija jezgra iznosi 46.10 mW, što predstavlja osnovnu disipaciju kada sistem miruje. Konačno, I/O termička disipacija snage je 10.28 mW, koja se odnosi na disipaciju usljed ulazno-izlaznih operacija. Ovi podaci pružaju sveobuhvatan pregled termičkih karakteristika sistema.

Tabela 8: Upotreba resursa čipa EP3C5E144C7 iz familije Cyclone III, koji je korišćen za hardversku implementaciju pravougaonog kodera (9,4)

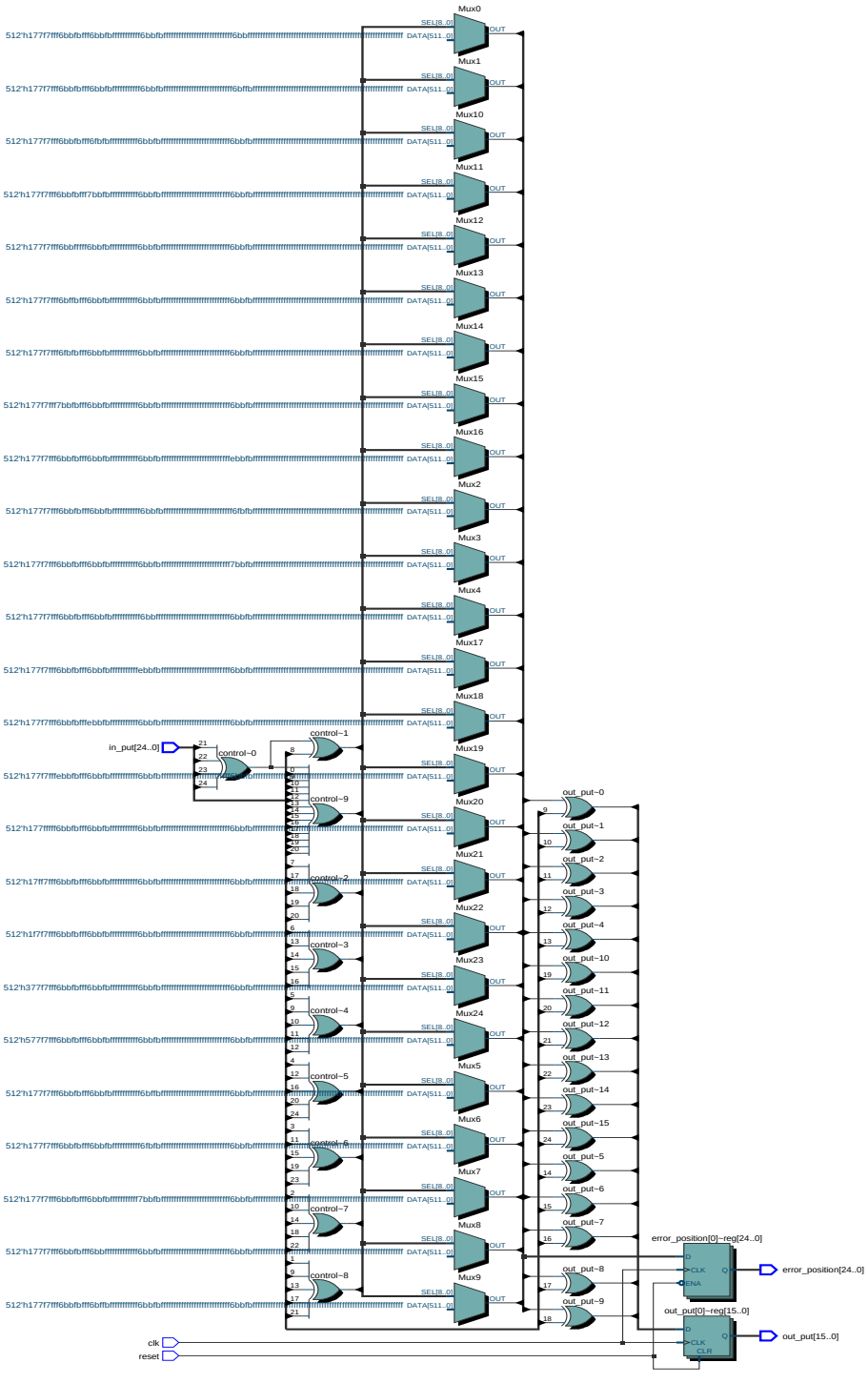
Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C5E144C7	9 / 5,136 (< 1 %)	5 / 5,136 (< 1 %)	9 / 5,136 (< 1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
15 / 95 (16 %)	0	0 / 423,936 (0 %)	0 / 46 (0 %)	0 / 2 (0 %)

Pravougaoni dekoder (9,4) je u stanju da ispravi jednu grešku u kodiranoj poruci. Signal *in_put* predstavlja kodiranu poruku koja se našla na izlazu kodera, dok signal *out_put* predstavlja dekodiranu poruku, tj. originalnu sekvencu informacionih bita. Signal *error_position* ukazuje na poziciju greške i imaće vrijednost 1 na onom mjestu na kojem je greška detektovana. Varijabla *control* koristi se za provjeru parnosti i detekciju greške. U zavisnosti od vrijednosti ove varijable, multipleksori na odgovarajuće mjesto signala *error_position* postavljaju jedinicu ili nulu. Ispravljena poruka dobija se kao rezultat XOR operacije nad signalom *errror_position* i ulaznim signalom *in_put*.

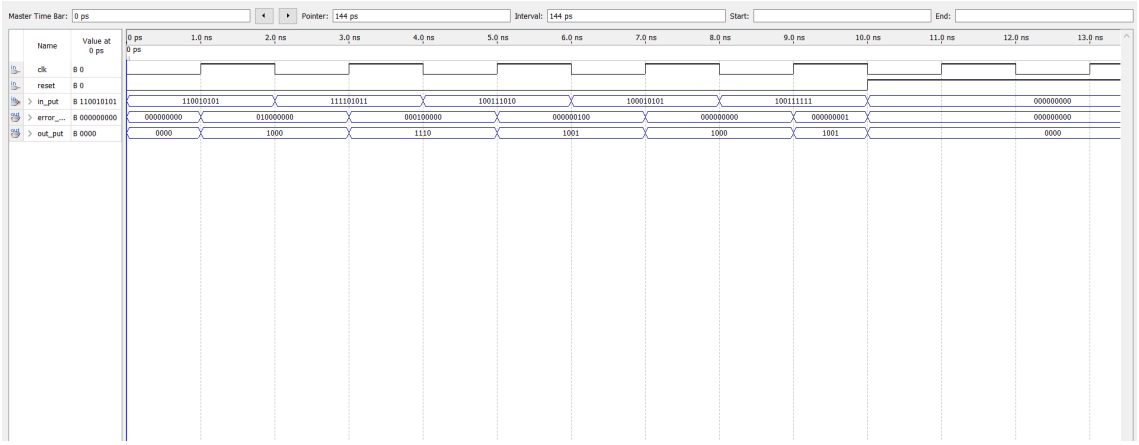


Slika 29: RTL šema pravougaonog dekodera (9, 4)

Za provjeru pozicije greške, odnosno za generisanje svakog pojedinačnog bita signala $error\_position$, potreban je jedan multipleksor. Upravo zbog ovoga u dizajnu pravougaonog dekodera (25, 16) postoji 25 multilpleksora, jer postoji 25 različitih pozicija na kojima može doći do greške (slika 26).



Slika 30: RTL šema pravougaonog dekodera (25, 16)



Slika 31: Rezultati simulacije za pravougaoni dekoder (9, 4)

D flip-flopovi osiguravaju da se izlazni signali generišu na uzlaznim ivicama taktnog signala clk, slika 27. Ulazni signali se učitavaju na svakoj silaznoj ivici taktnog signala clk. Detekcija i korekcija greške obavljaju se u jednom ciklusu taktnog signala. Prva tri testna podatka predstavljaju podatke iz primjera 1, poglavlje 2, gdje su greške prisutne na pozicijama 2, 4 i 7 (tj. greška na poziciji 2 u prvom signalu, na poziciji 4 u drugom signalu i na poziciji 7 u trećem signalu). Nakon korekcije, signal out_put predstavlja informacione bitove bez grešaka. U četvrtom taktu nije došlo do greške, pa svi biti signala $error\_position$ imaju vrijednost logičke nule. U petom taktu desila se greška na posljednjem bitu, što zaključujemo iz signala $error\_position$, koji ima vrijednost 1 na poziciji greške. Nakon petog takta, signal $reset$ je na visokom nivou, pa su svi izlazi resetovani.

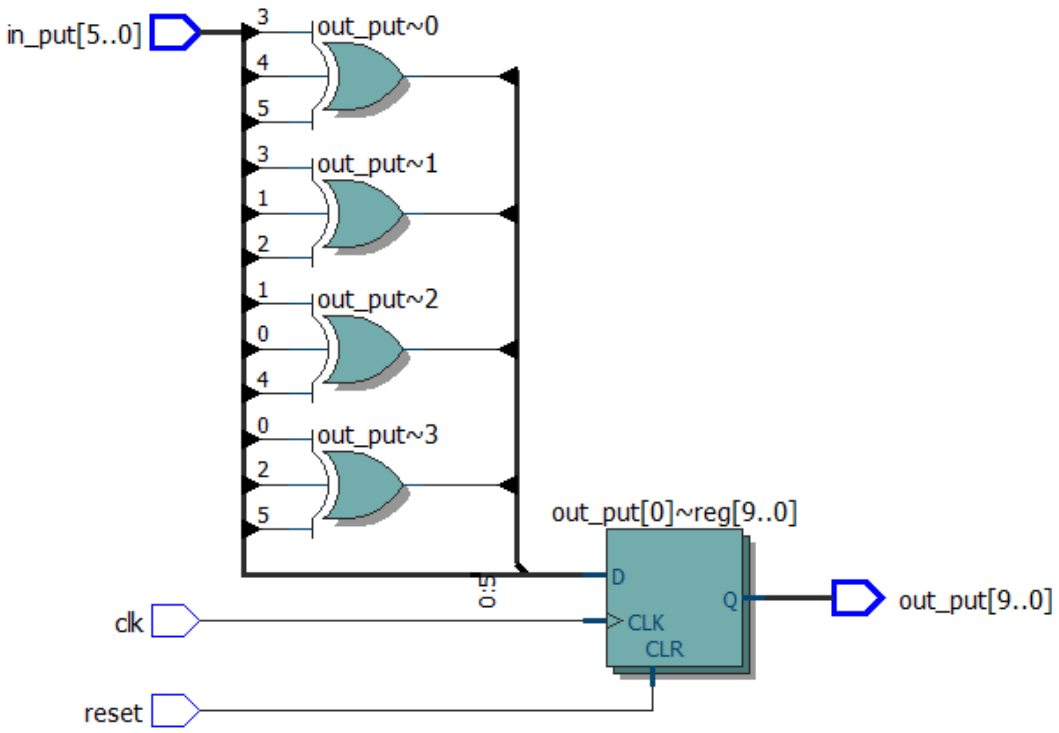
Rezultati analize snage pokazuju da ukupna termička disipacija snage iznosi 57.23 mW. Dinamička termička disipacija jezgra je 0.00 mW, statička termička disipacija jezgra je 46.10 mW, a I/O termička disipacija snage iznosi 11.12 mW.

Tabela 9: Upotreba resursa čipa EP3C5E144C7 iz familije Cyclone III, koji je korišćen za hardversku implementaciju pravougaonog dekodera (9,4)

Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C5E144C7	33 / 5,136 (< 1 %)	33 / 5,136 (< 1 %)	13 / 5,136 (< 1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
24 / 95 (25 %)	0	0 / 423,936 (0 %)	0 / 46 (0 %)	0 / 2 (0 %)

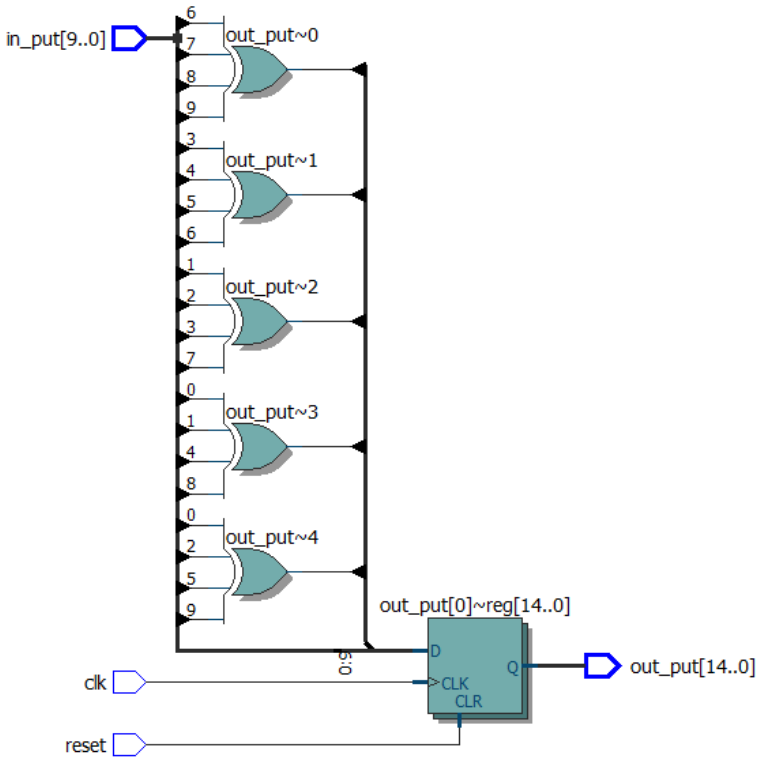
4.2 Trougaoni kod

Trougaoni koder (10, 6) realizovan je tako da se na izlaz prvo šalju informacioni, a zatim kontrolni biti. Ulazni signal in_put je vektor dužine šest bita i on predstavlja sekvencu informacionih bita koju treba kodirati. Signal out_put predstavlja izlaznu kodiranu poruku dužine deset bita, koja pored šest informacionih sadrži i četiri kontrolna bita.



Slika 32: RTL šema trougaonog kodera (10, 6)

Signal clk predstavlja taktni signal i on kontroliše glavni proces. Kodirana poruka se generiše na svakoj uzlaznoj ivici ovog signala. Signal *reset* se koristi za postavljanje izlaza u poznato stanje. Ukoliko je ovaj signal na visokom nivou, izlaz će biti resetovan. Odgovarajući informacioni biti dovode se na ulaze XOR kapija kako bi se generisali kontrolni biti, slika 28. D flip-flop osigurava da se izlaz generiše na uzlaznim ivicama taktnog signala, samo ukoliko je signal *reset* na nivou logičke nule. Za generisanje svakog pojedinačnog kontrolnog bita potrebna je jedna XOR kapija. Upravo zbog ovoga u dizajnu trougaonog kodera (15, 10) postoji pet ovakvih kapija, slika 29.



Slika 33: RTL šema trougaonog kodera (15, 10)

Kako bi se verifikovala tačnost implementiranog kodera, više različitih vrijednosti ulaznog signala *in_put* će biti testirano kroz simulaciju, slika 30. Prva tri testna podatka predstavljaju poruke čije je kodiranje demonstrirano u primjeru 2, poglavlje 2. Kodirane poruke se generišu na svakoj uzlaznoj ivici taktnog signala *clk*. Nakon četvrog takta, signal *reset* je postavljen u visoko stanje, što znači da je izlaz resetovan sve dok ovaj signal ponovo ne dođe na nivo logičke nule.

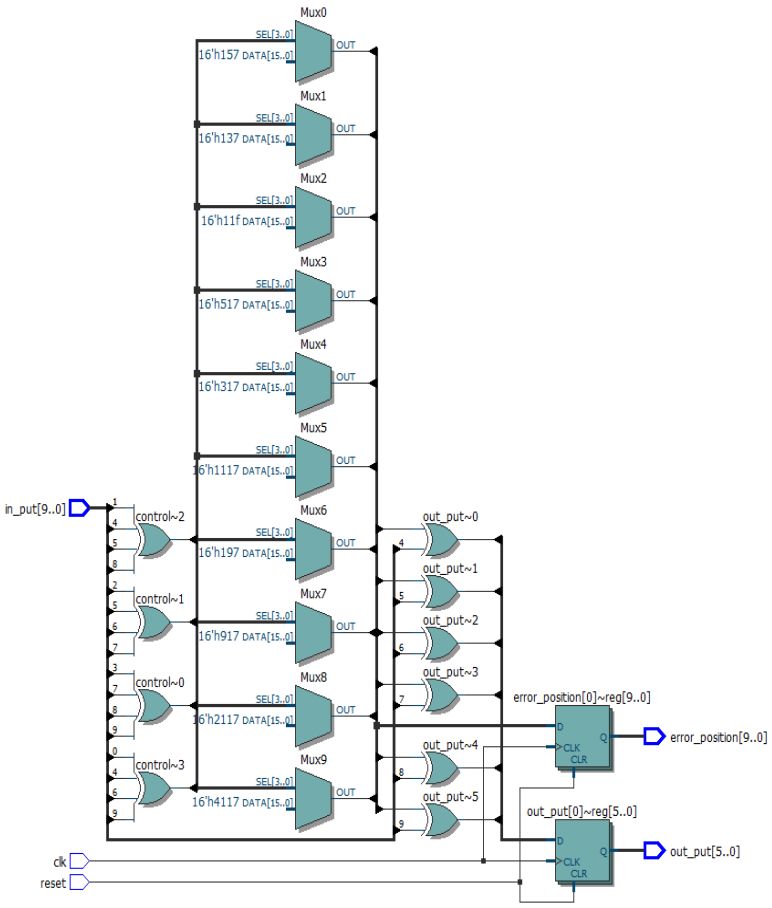
Slika 34: Rezultati simulacije za trougaoni koder (10, 6)

Rezultati analize snage pokazuju da ukupna termička disipacija snage iznosi 56.66 mW. Dinamička termička disipacija jezgra je 0.00 mW, statička termička disipacija jezgra je 46.10 mW, a I/O termička disipacija snage iznosi 10.56 mW.

Tabela 10: Upotreba resursa čipa EP3C5E144C7 iz familije Cyclone III, koji je korišćen za hardversku implementaciju trougaonog kodera (10, 6)

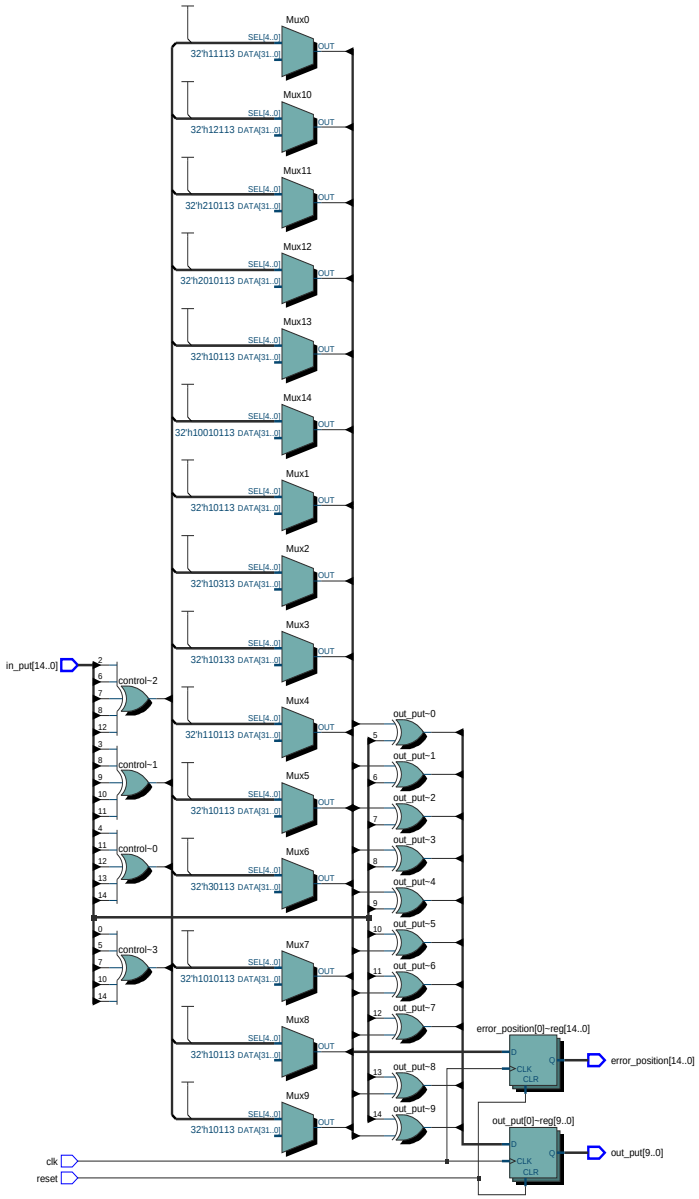
Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C5E144C7	10 / 5,136 (< 1 %)	4 / 5,136 (< 1 %)	10 / 5,136 (< 1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
18 / 95 (19 %)	0	0 / 423,936 (0 %)	0 / 46 (0 %)	0 / 2 (0 %)

Trougaoni dekoder (10, 6) je u stanju da detektuje i ispravi jednu grešku u kodiranoj poruci dužine deset bita.



Slika 35: RTL šema trougaonog dekodera (10, 6)

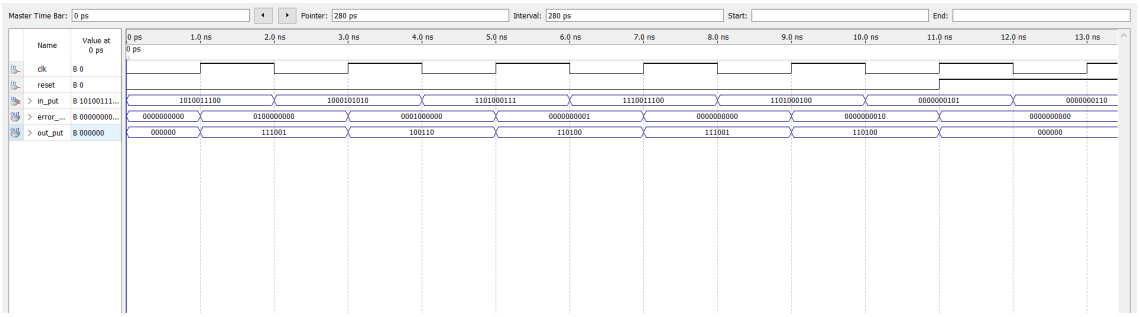
Na slici 31 prikazana je RTL šema ovakvog dekodera. Ulazni signal in_put je dužine deset bita i on predstavlja prethodno kodiranu poruku. Varijabla $control$ predstavlja bite parnosti koji uz pomoć XOR kapija provjeravaju sve informacione bite. Signal $error\_position$ ukazuje na poziciju greške i ovaj signal će imati vrijednost 1 na onom mjestu na kojem je greška detektovana. Ukoliko se greška nije desila, svi bitovi ovog signala će imati vrijedost 0. Izlazni signal out_put predstavlja originalnu poruku nakon detekcije i korekcije greške. Ovaj signal je dužine šest bita i on se dobija kao rezlutat XOR operacije nad ulaznim signalom in_put i signalom $error\_position$.



Slika 36: RTL šema trougaonog dekodera (15, 10)

Za generisanje svakog pojedinačnog bita signala *error_position* potreban je jedan multipleksor 2/1 koji odlučuje da li na to mjesto treba postaviti 1 ili 0. Zbog ovoga u dizajnu trougaonog dekodera (15, 10) postoji petnaest multipleksora, jer postoji petnaest pozicija na kojima može doći do greške, slika 32.

Na slici 33 prikazani su rezultati simulacija za trougaoni dekoder (10, 6). Prva tri testna podatka predstavljaju podatke iz primjera 2, poglavlje 2, gdje su greške prisutne na pozicijama 2, 4 i 10 (tj. greška na pozicija 2 u prvom signalu,na poziciji 4 u drugom signalu i na poziciji 10 u trećem signalu), što se može zaključiti i na osnovu signala *error_position* koji ima vrijednost logičke jedinice na ovim pozicijama. U četvrtom taktu nije došlo do greške, pa signal *error_position* ima vrijednost 0 na svakoj poziciji. U šestom taktu je signal *reset* na visokom nivou, pa su i svi izlazi resetovani. Detekcija i korekcija greške izvršavaju se u jednom ciklusu taktnog signala *clk*. Izlazni signal *out_put* koji predstavlja dekodiranu poruku nakon korekcije greške se takođe generiše na svakoj uzlaznoj ivici taktnog singala.



Slika 37: Rezultati simulacije za trougaoni dekoder (10, 6)

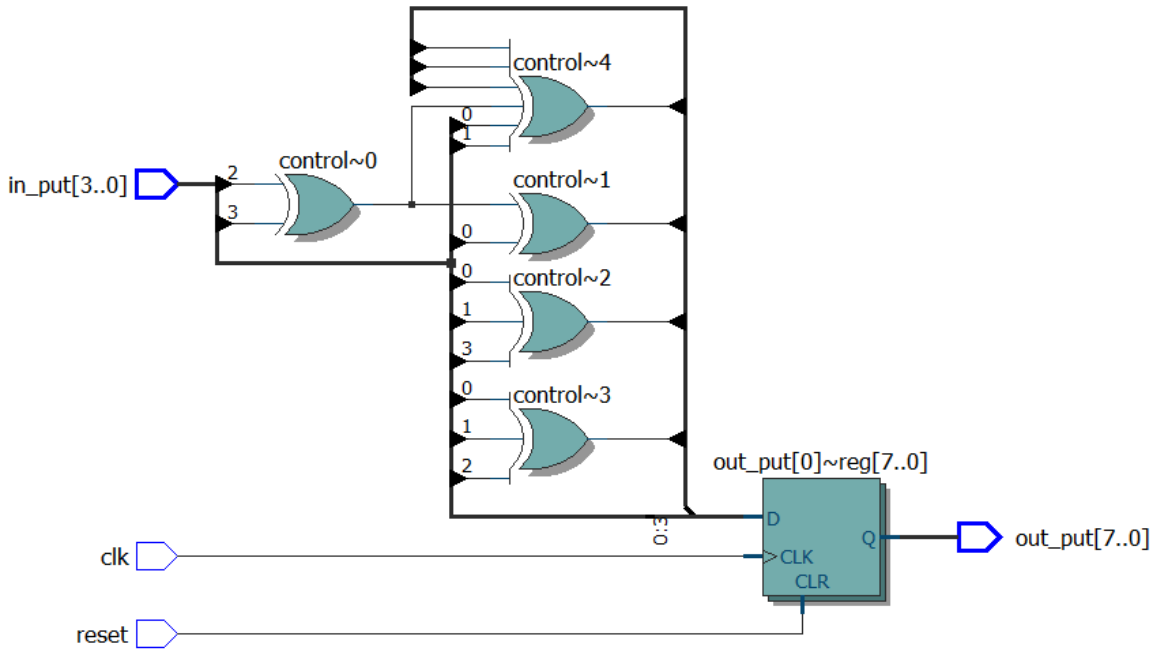
Ukupna termička disipacija snage iznosi 57.60 mW. Dinamička termička disipacija jezgra je 0.00 mW, statička termička disipacija jezgra je 46.10 mW, a I/O termička disipacija snage iznosi 11.50 mW.

Tabela 11: Upotreba resursa čipa EP3C5E144C7 iz familije Cyclone III, koji je korišćen za hardversku implementaciju trougaonog dekodera (10, 6)

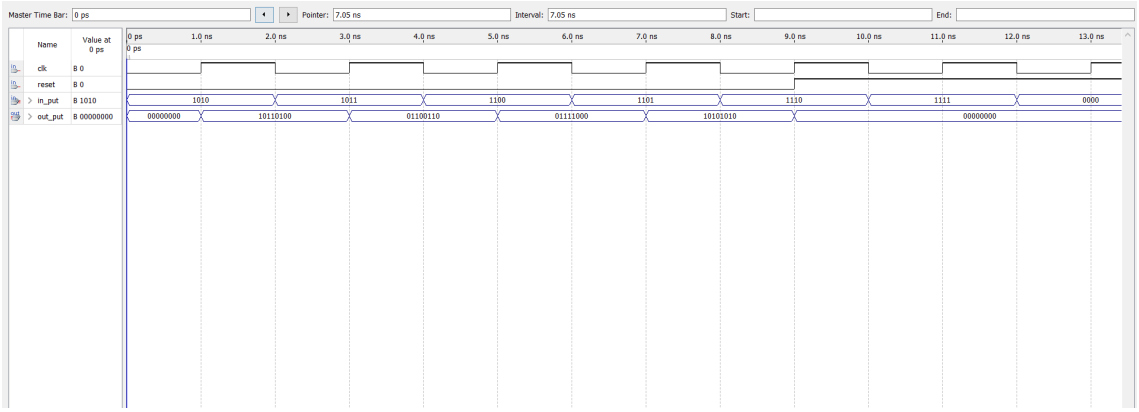
Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C5E144C7	20 / 5,136 (< 1 %)	20 / 5,136 (< 1 %)	16 / 5,136 (< 1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
15 / 95 (16 %)	0	0 / 423,936 (0 %)	0 / 46 (0 %)	0 / 2 (0 %)

4.3 Hamingov kod

Implementirani prošireni Hamingov koder (8, 4) prima poruku dužine četiri bita i na svom izlazu generiše kodnu riječ dužine osam bita. Na slici 34 prikazana je RTL šema proširenog Hamingovog koda (8, 4). Signal in_put predstavlja ulaznu sekvencu informacionih bita, dok signal out_put predstavlja izlaznu kodnu riječ. Odgovarajući informacioni biti dovode se na ulaze XOR kapija kako bi se generisali kontrolni biti. Posljednji kontrolni bit predstavlja bit parnosti za čitavu riječ, pa se na ulaz gornje XOR kapije dovode svi informacioni, kao i svi kontrolni bitovi. D flip-flop osigurava sinhronizaciju izlaznog signala out_put sa kontrolnim signalima clk i $reset$. Signal clk predstavlja taktni signal i izlazne kodirane poruke će se generisati na svakoj uzlaznoj ivici ovog signala. Signal $reset$ služi za postavljanje sistema u poznato stanje. Ukoliko je ovaj signal na visokom nivou, izlaz će biti resetovan. Kako bi se verifikovala tačnost implementiranog kodera, različiti ulazi će biti testirani kroz simulaciju, slika 35. Taktni signal traje 2 ns, a izlazi se generišu na svakoj uzlaznoj ivici ovog signala (tj. na svake dvije ns). Prvi testni podatak predstavlja poruku čije je kodiranje demonstrirano u primjeru 3, poglavlje 2. U petom taktu signal $reset$ prelazi u visoko stanje, pa će od tog momenta izlaz biti resetovan na stanje logičke nule.



Slika 38: RTL šema proširenog Hamingovog kodera (8, 4)



Slika 39: Rezultati simulacije za prošireni Hamingov koder (8, 4)

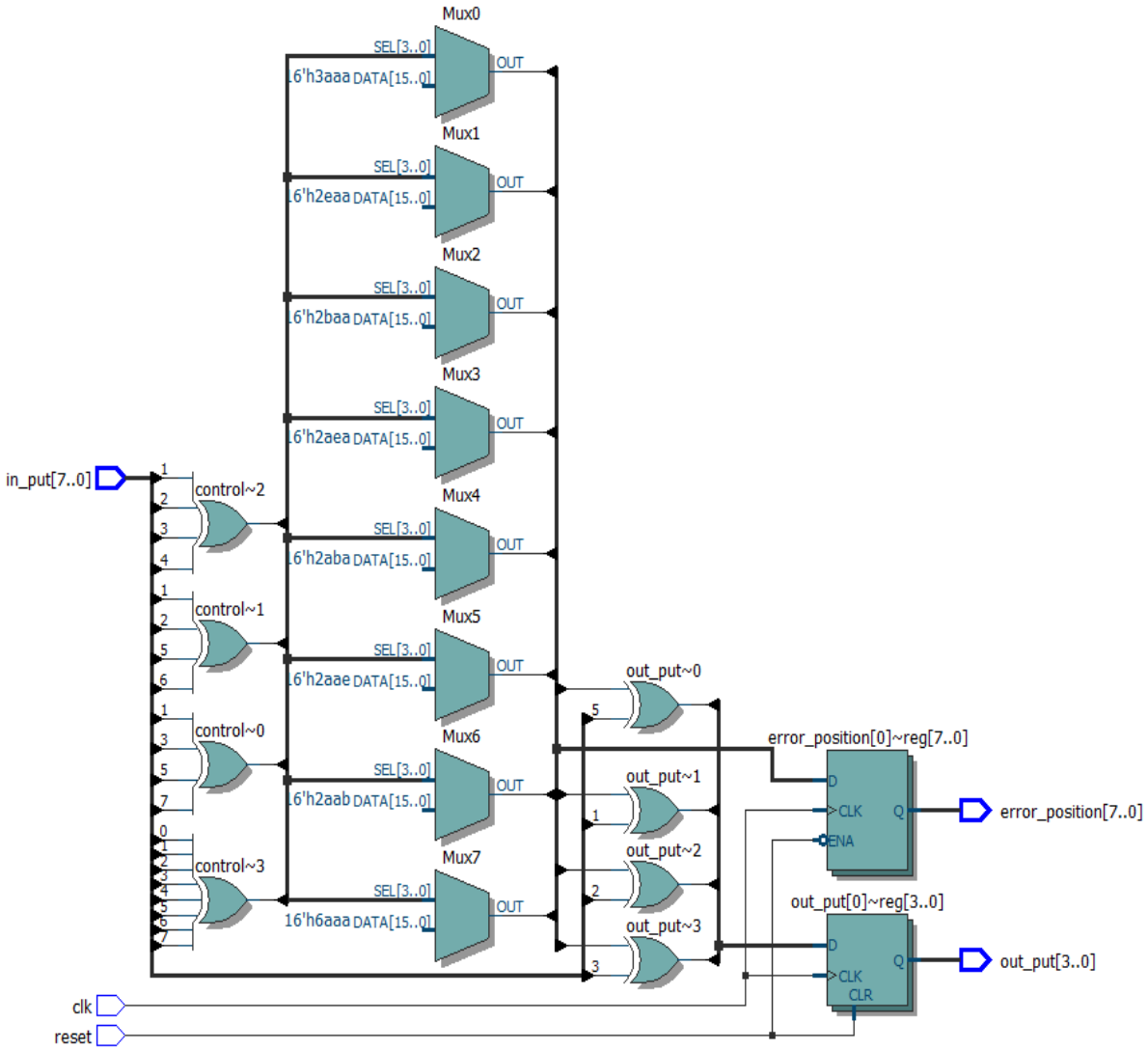
Ukupna termička disipacija snage iznosi 56.29 mW. Dinamička termička disipacija jezgra je 0.00 mW, statička termička disipacija jezgra je 46.10 mW, a I/O termička disipacija snage iznosi 11.19 mW.

Tabela 12: Upotreba resursa čipa EP3C5E144C7 iz familije Cyclone III, koji je korišćen za hardversku implementaciju proširenog Hamingovog kodera (8, 4)

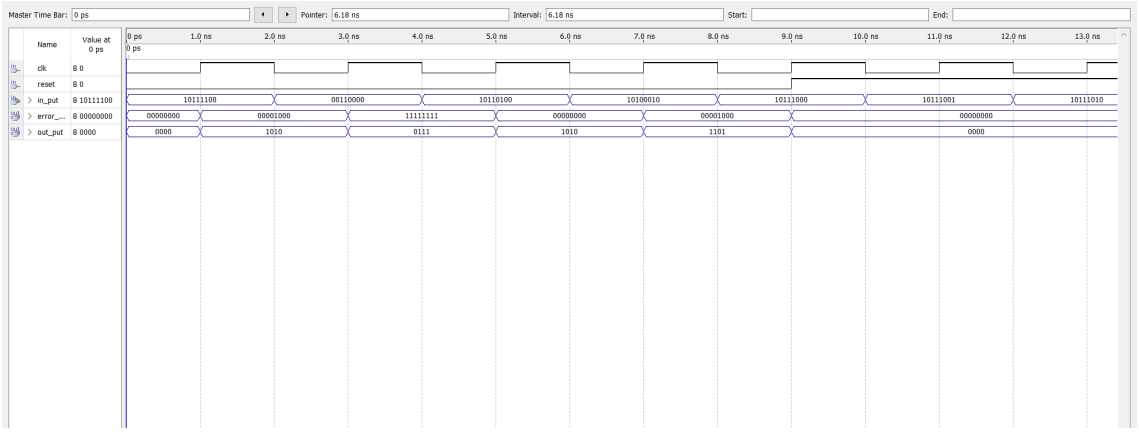
Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C5E144C7	8 / 5,136 (< 1 %)	4 / 5,136 (< 1 %)	8 / 5,136 (< 1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
14 / 95 (15 %)	0	0 / 423,936 (0 %)	0 / 46 (0 %)	0 / 2 (0 %)

Prošireni Hamingov dekoder (8, 4) prima kodiranu poruku dužine osam bita i u stanju je detektovati dvije i ispraviti jednu grešku u poruci. RTL šema implementiranog dekodera prikazana je na slici 36. Signal in_put predstavlja ulaznu kodiranu poruku, dok signal out_put predstavlja originalnu sekvencu informacionih bita nakon detekcije i korekcije potencijalne greške. Signal $error\_position$ je dužine osam bita i on ukazuje na poziciju greške. Ukoliko je svaki bit ovog signala jednak nuli, onda možemo zaključiti da nema greške u kodiranoj poruci. Ako se jedinica nalazi na bilo kojoj poziciji signala $error\_position$, onda zaključujemo da se greška dogodila na toj poziciji u kodiranoj poruci. Konačno, ako signal $error\_position$ ima vrijednost jedinice na svim svojim pozicijama, onda zaključujemo da postoje dvije greške u poruci i da nije moguće uspješno izvršiti proces dekodiranja. Prvi korak jeste provjera svih bita parnosti. Varijabla $control$ je dužine četiri bita i ona pred-

stavlja provjeru parnosti za svaki kontrolni bit. Odgovarajući bitovi ulaznog signala in_put dovode se na ulaze XOR kapija kako bi se generisali bitovi signala $control$. U zavisnosti od vrijednosti signala $control$, multipleksori 2/1 na odgovarajuću poziciju signala $error\_position$ postavljaju nulu ili jedinicu. U dizajnu proširenog Hamongovog dekodera (8, 4) postoji osam multipleksora, jer postoji osam pozicija na kojima može doći do greške, slika 36. Izlazni signal out_put dobija se primjenom XOR operacije nad ulaznim signalom in_put i signalom $error\_position$ koji ima grešku. Dva D flip-flopa osiguravaju sinhronizaciju izlaznih signala sa kontrolnim signalom clk. Jedan čuva vrijednost signala $error\_position$, a drugi vrijednost signala out_put sve do prve uzlazne ivice taktnog signala clk. Ukoliko je signal $reset$ na visokom nivou, svi izlazi će biti resetovani.



Slika 40: RTL šema proširenog Hamingovog dekodera (8, 4)



Slika 41: Rezultati simulacije za prošireni Hamingov dekoder (8, 4)

Na slici 37 prikazani su rezultati simulacije za prošireni Hamingov dekoder (8, 4). Za detekciju i korekciju greške potreban je jedan ciklus taktnog signala *clk*, a izlazi se generišu na svakoj uzlaznoj ivici ovog signala. Prva tri testna podatka predstavljaju poruku dekodiranu u primjeru 3, poglavlje 2. U prvom taktu greška se dogodila na poziciji 5 u kodiranoj poruci, što možemo zaključiti na osnovu signala *error_position*, koji ima vrijednost 1 na ovoj poziciji. U drugom taktu signal *error_position* ukazuje na prisustvo dvije greške u kodiranoj poruci, što znači da pravilno dekodiranje nije moguće. U trećem taktu nije došlo do greške pa signal *error_position* ima vrijednost nule na svakoj svojoj poziciji. U petom taktu signal *reset* prelazi u visoko stanje, pa su od ovog momenta svi izlazi resetovani.

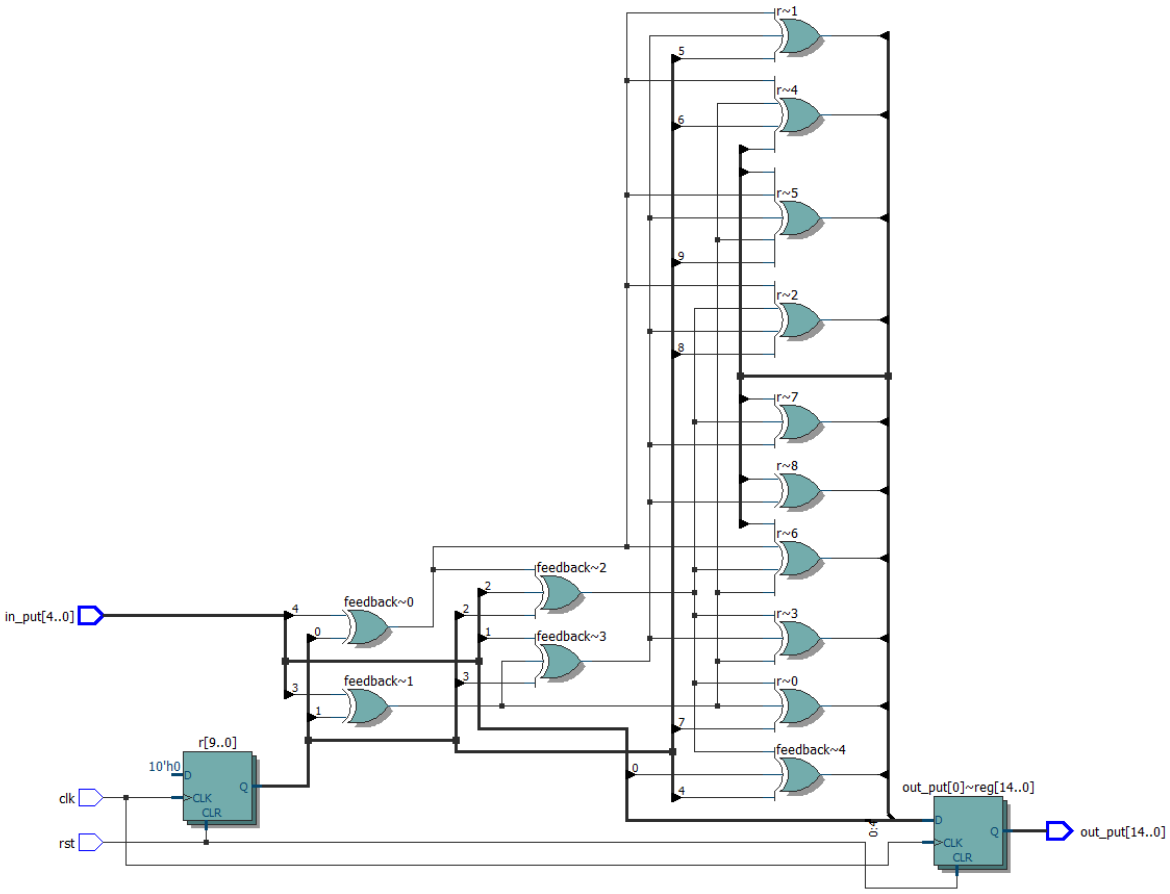
Ukupna termička disipacija snage iznosi 57.04 mW. Dinamička termička disipacija jezgra je 0.00 mW, tj. nema dodatne disipacije usljed promjena u logičkim stanjima. Statička termička disipacija jezgra je 46.10 mW, a I/O termička disipacija snage iznosi 10.94 mW.

Tabela 13: Upotreba resursa čipa EP3C5E144C7 iz familije Cyclone III, koji je korišćen za hardversku implementaciju proširenog Hamingovog dekodera (8, 4)

Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C5E144C7	16 / 5,136 (< 1 %)	16/ 5,136 (< 1 %)	12 / 5,136 (< 1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
22 / 95 (23 %)	0	0 / 423,936 (0 %)	0 / 46 (0 %)	0 / 2 (0 %)

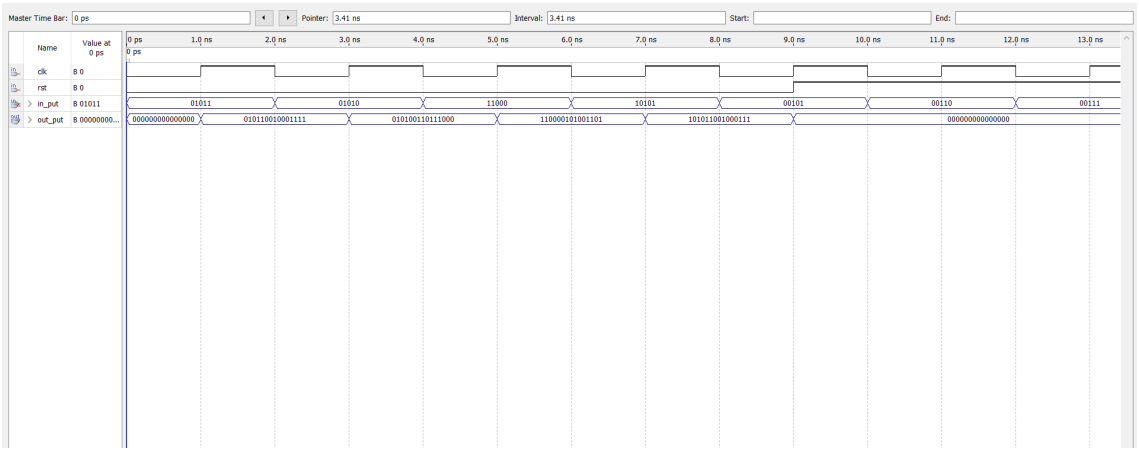
4.4 BCH kod

BCH koder (15 , 5, 3) prima poruku dužine pet bita i na svom izlazu generiše kodiranu poruku dužine peatnaest bita. Kontrolni bitovi se generišu pomijeranjem informacionih bita kroz ćelije linearnog pomjeračkog registra, kao što je prikazano na slici 12. Signal in_put predstavlja ulaznu sekvencu informacionih bita, dok signal out_put predstavlja izlaznu kodiranu poruku. Varijabla r je dužine deset bita i ona predstavlja kontrolne bite tj. sadržaje registra nakon prolaska svih informacionih bita kroz linearni pomjerački registar. Za generisanje svakog pojedinačnog kontrolnog bita potrebna je jedna XOR kapija, slika 38. Dodatne XOR kapije služe za generisanje povratne sprege, čiji se sadržaj čuva u varijabli $feedback$. Ova varijabla je dužine pet bita, jer se rezlutat povratne sprege mijenja kada god se vrši pomjeranje informacinih bitova kroz ćelije pomjeračkog registra. D flip-flopovi osiguravaju da se izlazi generišu na svakoj uzlaznoj ivici taktnog signala clk. Signal rst služi za postavljanje sistema u poznato stanje. Kada je ovaj signal na visokom nivou, proces kodiranja se prekida i izlaz je na nivou logičke nule.



Slika 42: RTL šema BCH kodera (15, 5, 3)

Kako bi se verifikovala tačnost implementiranog kodera, različiti ulazni podaci su testirani kroz simulaciju, slika 39. Prvi ulazni podatak predstavlja poruku čije je kodiranje demonstrirano u primjeru 4, poglavlje 2. Izlazi se generišu na svakoj uzlaznoj ivici taktnog signala. Signal *rst* je postavljen na nivo logičke jedinice u petom taktu, pa će od ovog momenta izlazi biti resetovani, sve dok ovaj signal ponovo ne dođe na nivo logičke nule.



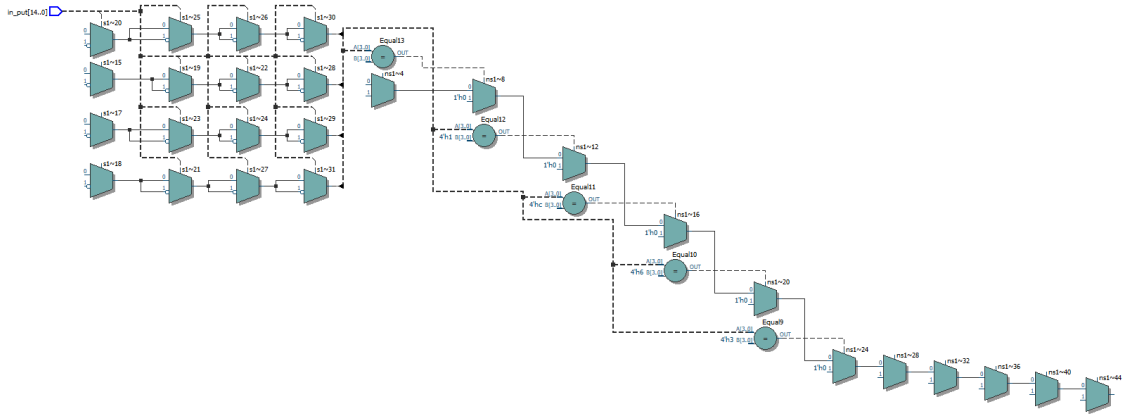
Slika 43: Rezultati simulacije za BCH koder (15, 5, 3)

Ukupna termička disipacija snage iznosi 57.05 mW. Dinamička termička disipacija jezgra je 0.00 mW, statička termička disipacija jezgra je 46.11 mW, a I/O termička disipacija snage iznosi 10.94 mW.

Tabela 14: Upotreba resursa čipa EP3C5E144C7 iz familije Cyclone III, koji je korišćen za hardversku implementaciju BCH kodera (15, 5, 3)

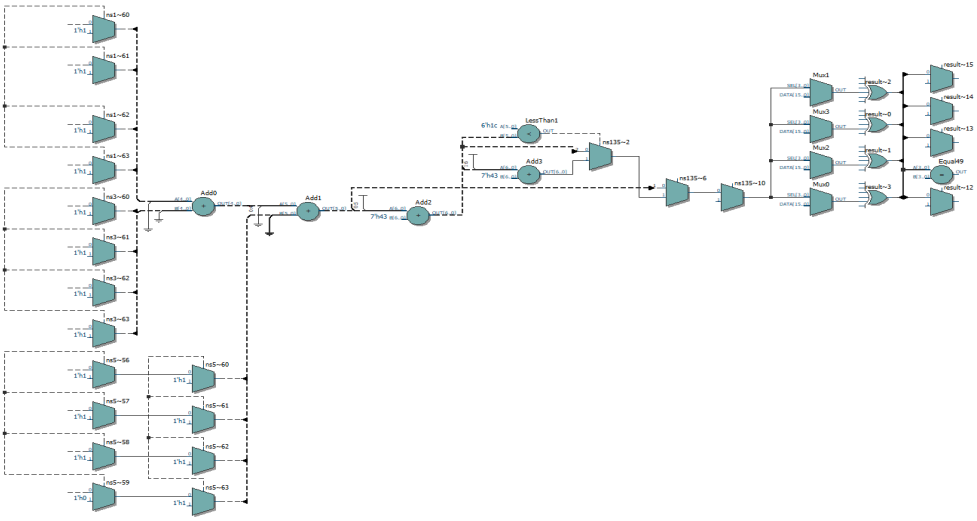
Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C5E144C7	15 / 5,136 (< 1 %)	10/ 5,136 (< 1 %)	15 / 5,136 (< 1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
22 / 183 (12 %)	0	0 / 423,936 (0 %)	0 / 46 (0 %)	0 / 2 (0 %)

BCH dekoder (15, 5, 3) prima kodiranu poruku dužine petnaest bita i u stanju je ispraviti do tri greške u poruci. Signal *in_put* predstavlja ulaznu kodiranu poruku dok signal *out_put* predstavlja kodiranu poruku nakon detekcije i korekcije eventualnih grešaka. Izlazni signal *decoded* predstavlja originalnu sekvencu infromacionih bita.



Slika 44: RTL šema kola za računanje sindroma S_1. Kola za računanje ostalih sindroma realizuju se analogno, pa njihove šeme neće biti prikazane.

Prvi korak je računanje šest sindromskih komponenti na osnovu ulazne poruke in_put. Varijable $s1, s2, ..., s6$ predstavljaju binarne reprezentacije vrijednosti sindroma i dužine su četiri bita, jer se radi o GF(2^4). Varijable $ns1, ns2...ns6$ predstavljaju stepene elemenata Galoovog polja koji odgovaraju vrijednostima sindroma i one mogu imati vrijednost od 0 do 15. Na slici 40 prikazana je RTL šema kola za računanje prvog sindroma. Multipleksori 2/1 provjeravaju da li se na određenoj poziciji u ulaznoj poruci nalazi jedinica ili nula. Nakon toga, dobijena vrijednost sindroma S_1 se poredi sa tabelom elemenata GF(2^4), kako bi se utvrdio stepen primitivnog elementa $ns1$. Ako sindrom $s1$ ima vrijednost 0110, onda će vrijednost varijable $ns1$ biti 5, kako je 0110 binarna reprezentacija elementa α^5 (tabela 2, poglavlje 2). Sljedeći korak je računanje determinante matrice M.

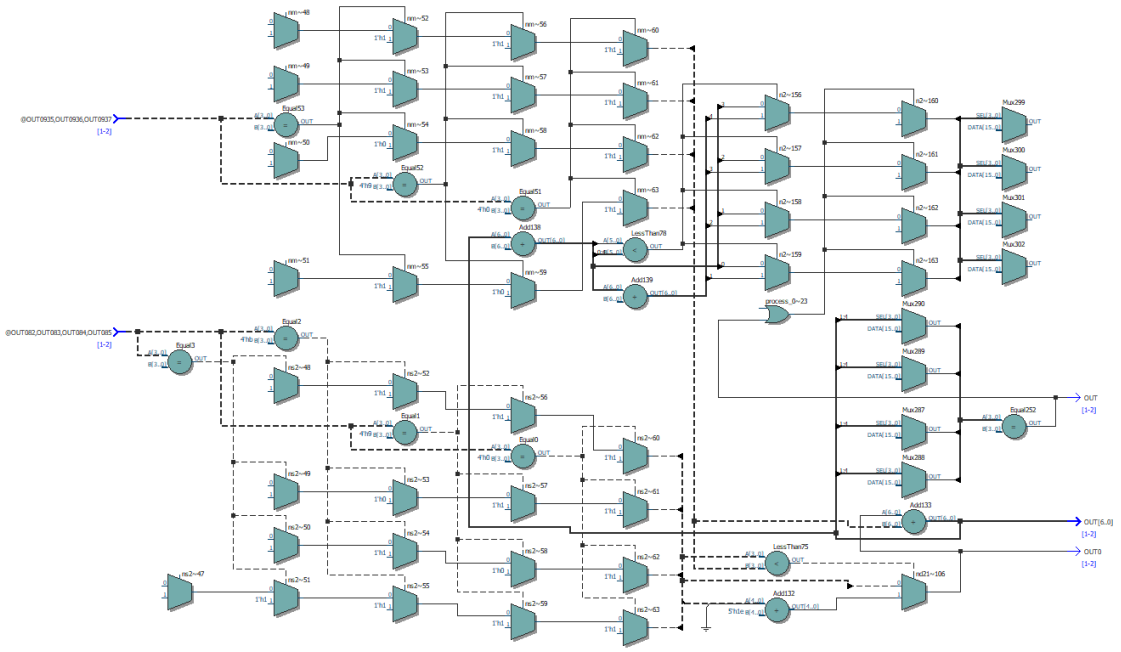


Slika 45: RTL šema kola za računanje jednog elementa determinante det(M)

Determinanta matrice M, za $i = 3$, se računa na sljedeći način:

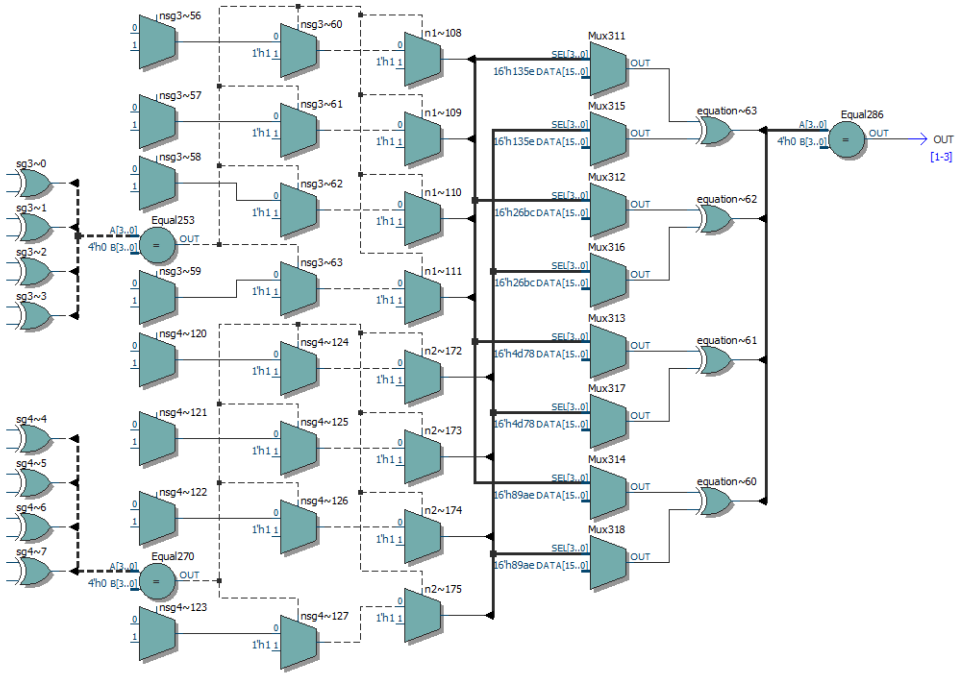
$$\det(M) = s1s3s5 + s1s4s4 + s2s2s5 + s2s4s3 + s3s2s4 + s3s3s3.$$

Vrijednost determinante se smješta u varijablu $result$, dok se stepen elementa GF(2^4), koji odgovara vrijednosti determinante, smješta u varijablu nm. Ukoliko $result$ ima vrijednost 0000, smanjuju se dimenzije matrice M i ponovo se računa determinanta. Kolo za računanje jednog elementa determinante i dodavanje njegove vrijednosti u varijablu $result$ prikazano je na slici 31. Sada je potrebno naći inverznu matricu M^{-1}. Elementi inverzne matrice se smještaju u varijable $d11, d12, d13, d21, d22, d23, d31, d32, d33$. Na slici 42 prikazana je RTL šema kola za računanje jednog elementa invezne matrice, pri čemu se računanje preostalih elementa obavlja na isti način.



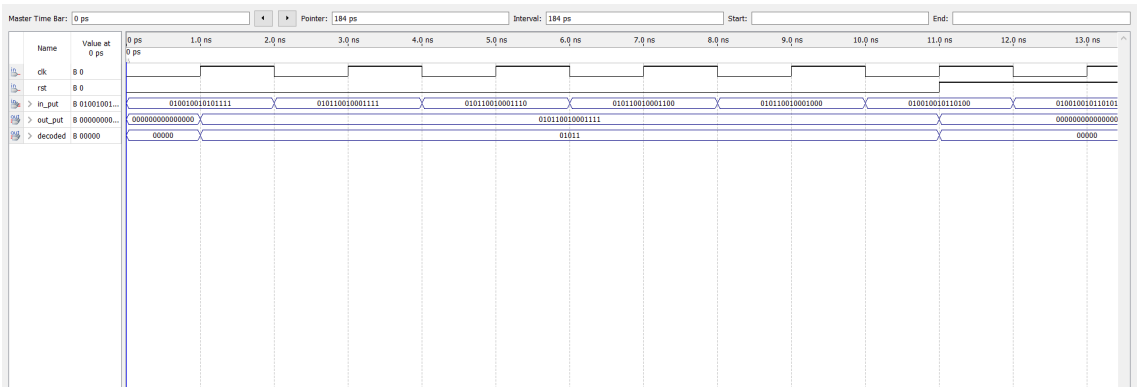
Slika 46: RTL šema kola za računanje jednog elementa inverzne matrice M^{-1}

Množenjem inverzne matrice sa sindromskim vektorom dobijaju se koeficijenti polinoma lokatora greške $\sigma(x)$. Nakon toga, korijeni polinoma $\sigma(x)$ traže se Chienovom pretragom. Chienova pretraga evaluira polinom $\sigma(x)$ sa različitim vrijednostima x, gdje su vrijednosti za x različiti elementi Galoovog polja (2^4). Za svaku vrijednost α^i, gdje je α primitivni element polja, provjerava se da li je $\sigma(\alpha^i) = 0$. Ako je evaluacija jednaka nuli, onda je α^i korijen polinoma $\sigma(x)$. Koeficijenti polinoma lokatora greške smješteni su u varijable $sg1, sg2...sg6$. Varijable $n1, n2$ i $n3$ uzimaju redom vrijednosti iz GF(2^4). Korijen polinoma lokatora greške je ono n za koje varijabla $equation$ ima vrijednost 0000.



Slika 47: RTL šema kola za traženje korijena polinoma lokatora greške Chienovom pretragom

Kako bi se testirala ispravnost implementiranog kodera, više različitih vrijednosti ulaznog signala *in_put* je posmatrano u okviru simulacije, slika 44. Detekcija i korekcija greške obavljaju se u jednom ciklusu taktnog signala *clk*. Prvi testni podatak predstavlja poruku čije je dekodiranje demonstrirano u primjeru 4, poglavlje 2. U drugom taktu nije došlo do greške u poruci. U trećem, četvrtom i petom taktu došlo je do greške na posljednjem, posljednja dva i posljednja tri bita, respektivno. Kako se vrijednost signala *out_put* ne mijenja tokom prvih pet taktova, možemo zaključiti da su sve greške pravilno detektovane i ispravljene. Nakon petog takta signal *rst* prelazi na nivo logičke jedinice, pa su svi izlazi resetovani.



Slika 48: Rezultati simulacije za BCH dekoder (15, 5, 3)

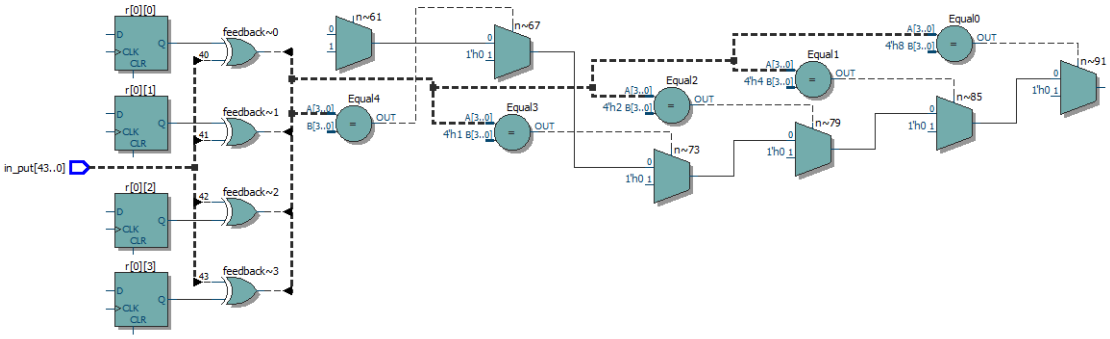
Ukupna termička disipacija snage iznosi 58.62 mW. Dinamička termička disipacija jezgra je 0.00 mW, statička termička disipacija jezgra je 46.28 mW, a I/O termička disipacija snage iznosi 12.34 mW.

Tabela 15: Upotreba resursa čipa EP3C5E144C7 iz familije Cyclone III, koji je korišćen za hardversku implementaciju BCH dekodera (15, 5, 3)

Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C5E144C7	1,938 / 5,136 (38 %)	1,933 / 5,136 (38 %)	20 / 5,136 (< 1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
37 / 183 (20 %)	0	0 / 423,936 (0 %)	0 / 46 (0 %)	0 / 2 (0 %)

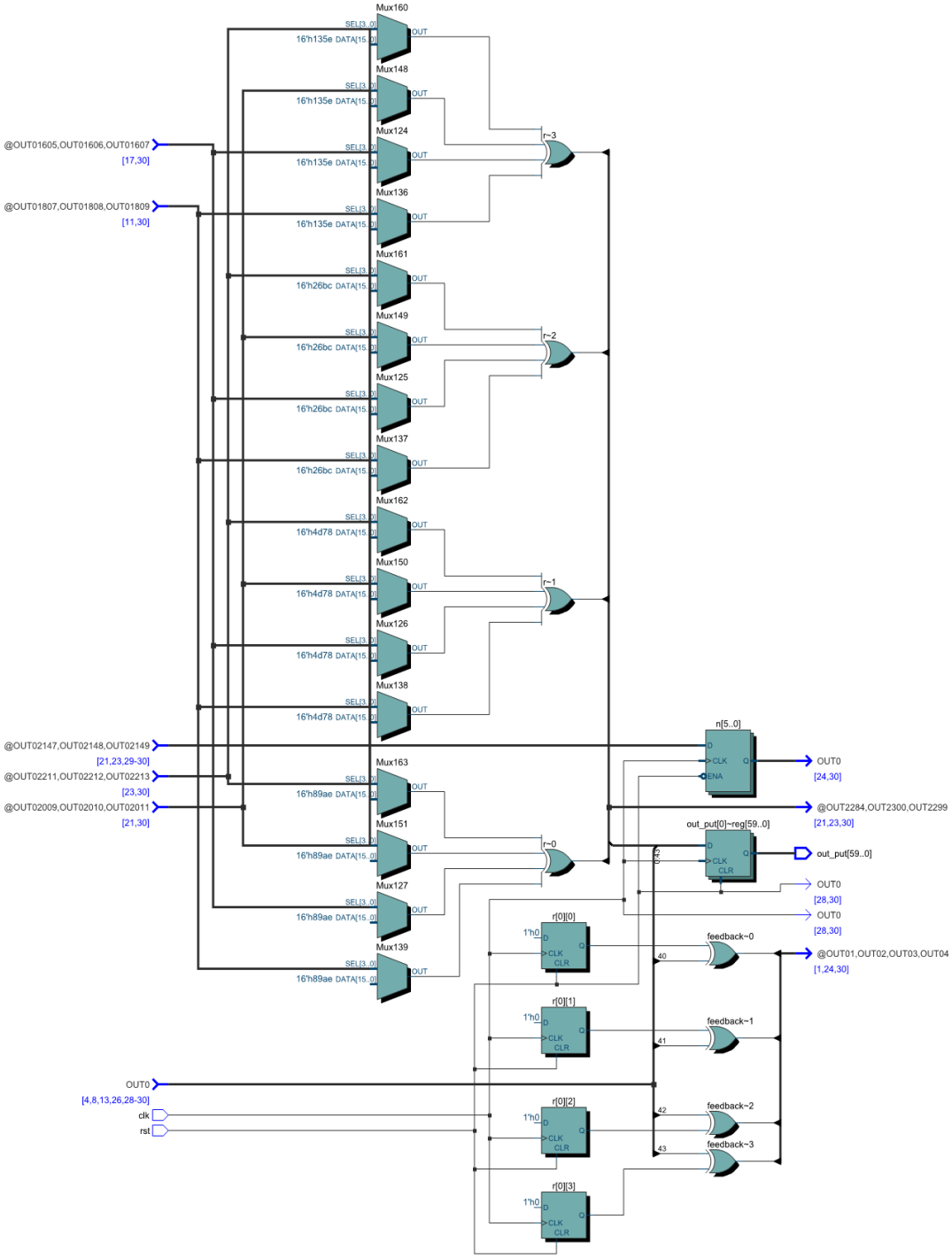
4.5 Reed-Solomon kod

Na ulaz Reed-Solomon (15, 11, 2) kodera dovodi se 11 informacionih simbola, a na izlazu se generiše kodna riječ dužine 15 simbola. Svaki simbol je dužine četiri bita, pa će izlazna kodirana poruka biti duga 60 bita. Ulazni signal in_put predstavlja sekvencu informacionih bita i dužine je 44 bita, dok izlazni signal out_put predstavlja kodiranu poruku dužine 60 bita. Kada informacioni simbol uđe u linearni pomjerački registar prvo se računa rezultat povratne sprege. Ova vrijednost se smješta u varijablu $feedback$, dok se stepen elementa GF(2^4) koji odgovara vrijednosti rezlutata povratne sprege smješta u varijablu n.



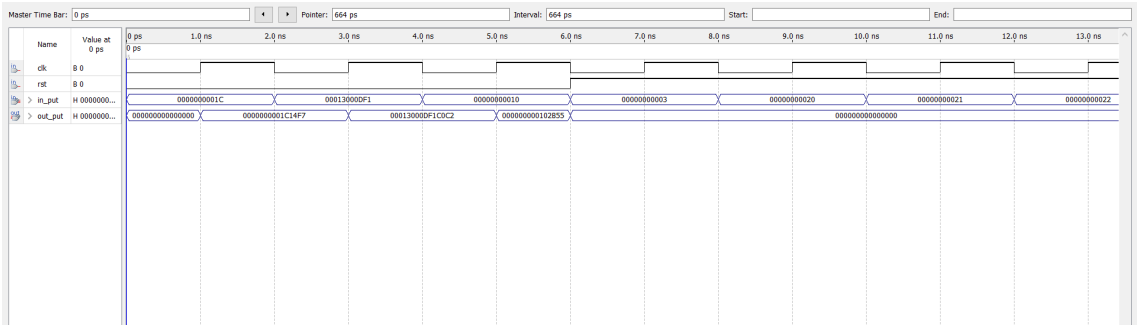
Slika 49: RTL šema kola za računanje rezultata povratne sprege

Sada treba rezultat povratne sprege pomnožiti sa odgovarajućim koeficijentima generatorskog polinoma. Rezultati ovog množenja smješteni su u varijable: nf1,nf2,nf3 i nf4. Na kraju, ove varijable se dovode na ulaz kola prikazanog na slici 46, kako bi se generisala izlazna kodna riječ.



Slika 50: RTL šema kola za generisanje RS kodne riječi

Različite vrijednosti ulaznih signala u RS koder su posmatrane, a rezultati simulacije su prikazani na slici 48. Ulazni i izlazni podaci su prikazani heksadecimalnim zapisom radi bolje preglednosti. Prvi ulazni podatak predstavlja poruku čije je kodiranje demonstrirano u primjeru 5, poglavlje 2. Poruka koja se našla na ulazu kodera u prvom taktu može se zapisati u polinomijalnom obliku kao: $m(x) = x + \alpha^6$, dok se izlazna kodirana poruka može zapisati kao: $c(x) = x^5 + x^4\alpha^6 + x^3 + x^2\alpha^2 + x\alpha^{12} + \alpha^{10}$, što potvrđuje tačnost implementiranog kodera. U drugom taktu na ulaz kodera dolazi poruka: $m(x) = x^7 + x^6\alpha^4 + x^2\alpha^{13} + x\alpha^{12} + 1$, a odgovarajuća kodirana poruka je: $c(x) = x^{11} + x^{10}\alpha^4 + x^6\alpha^{13} + x^5\alpha^{13} + x^4 + x^3\alpha^6 + x\alpha^6 + \alpha$. U trećem taktu na ulaz kodera dolazi poruka $m(x) = x$, a nakon kodiranja na izlazu se dobija poruka: $c(x) = x^5 + x^3\alpha + x^2\alpha^7 + x\alpha^8 + \alpha^8$ Nakon terćeg takta, signal rst prelazi na nivo logičke jedinice, pa su od ovog momenta izlazi resetovani.



Slika 51: Rezultati simulacije za Reed-Solomon (15, 11, 2) koder

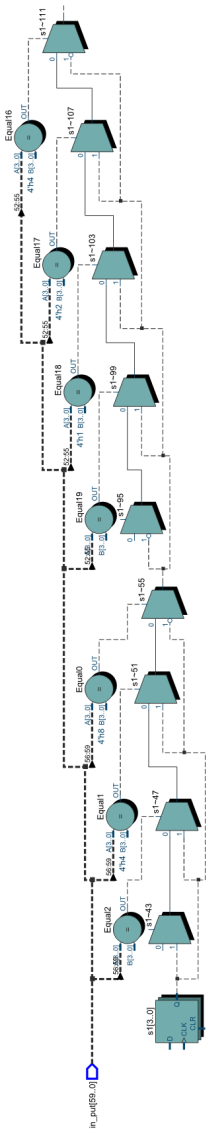
Ukupna termička disipacija snage iznosi 64.95 mW. Dinamička termička disipacija jezgra je 0.00 mW, statička termička disipacija jezgra je 46.14 mW, a I/O termička disipacija snage iznosi 18.81 mW.

Tabela 16: Upotreba resursa čipa EP3C5E144C7 iz familije Cyclone III, koji je korišćen za hardversku implementaciju Reed-Solomon (15, 11, 2) kodera

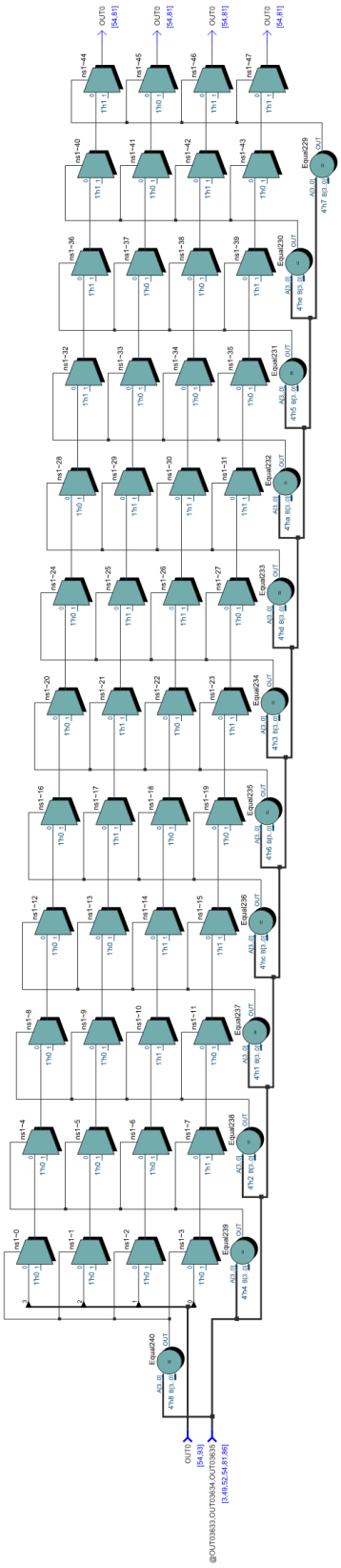
Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C5E144C7	187 / 5,136 (4 %)	143 / 5,136 (3 %)	60 / 5,136 (1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
106 / 183 (58 %)	0	0 / 423,936 (0 %)	0 / 46 (0 %)	0 / 2 (0 %)

Reed-Solomon (15, 11, 2) dekoder prima kodiranu poruku dužine 60 bita (15 simbola) i u stanju je da ispravi dva simbola u kodiranoj poruci, čak i ukoliko su

simboli oštećeni na pozicijama svih bita. Signal in_put predstavlja ulaznu kodiranu poruku, dok signal out_put predstavlja kodiranu poruku nakon ispravljanja eventualnih grešaka. Signal $decoded$ predstavlja originalnu sekvencu informacionih simbola i dužine je 44 bita (11 simbola). Prvi korak je traženje $2t$ sindromskih komponenti. Binarne reprezentacije vrijednosti sindroma smještaju se u varijable $s1$, $s2$, $s3$ i $s4$, dok se stepeni elemenata GF(2^4), koji odgovaraju vrijednostima sindroma, smještaju u varijable $ns1$, $ns2$, $ns3$ i $ns4$. Kolo za računanje prvog sindroma $s1$ prikazano je na slici 48. Sada se ovaj sindrom dovodi na ulaz kola sa slike 49, kako bi se izračunala vrijednost varijable $ns1$.

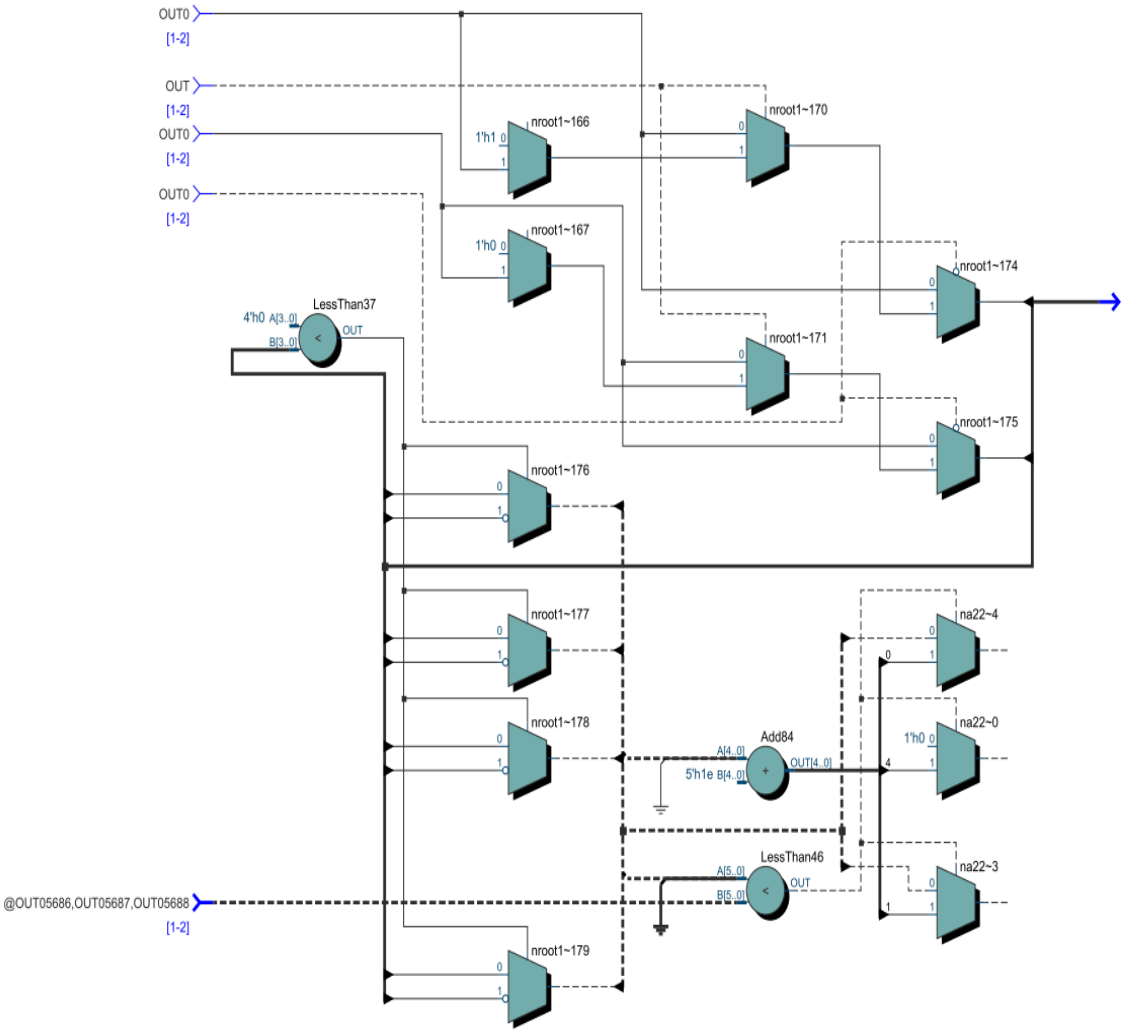


Slika 52: RTL šema kola za računanje prvog sindroma

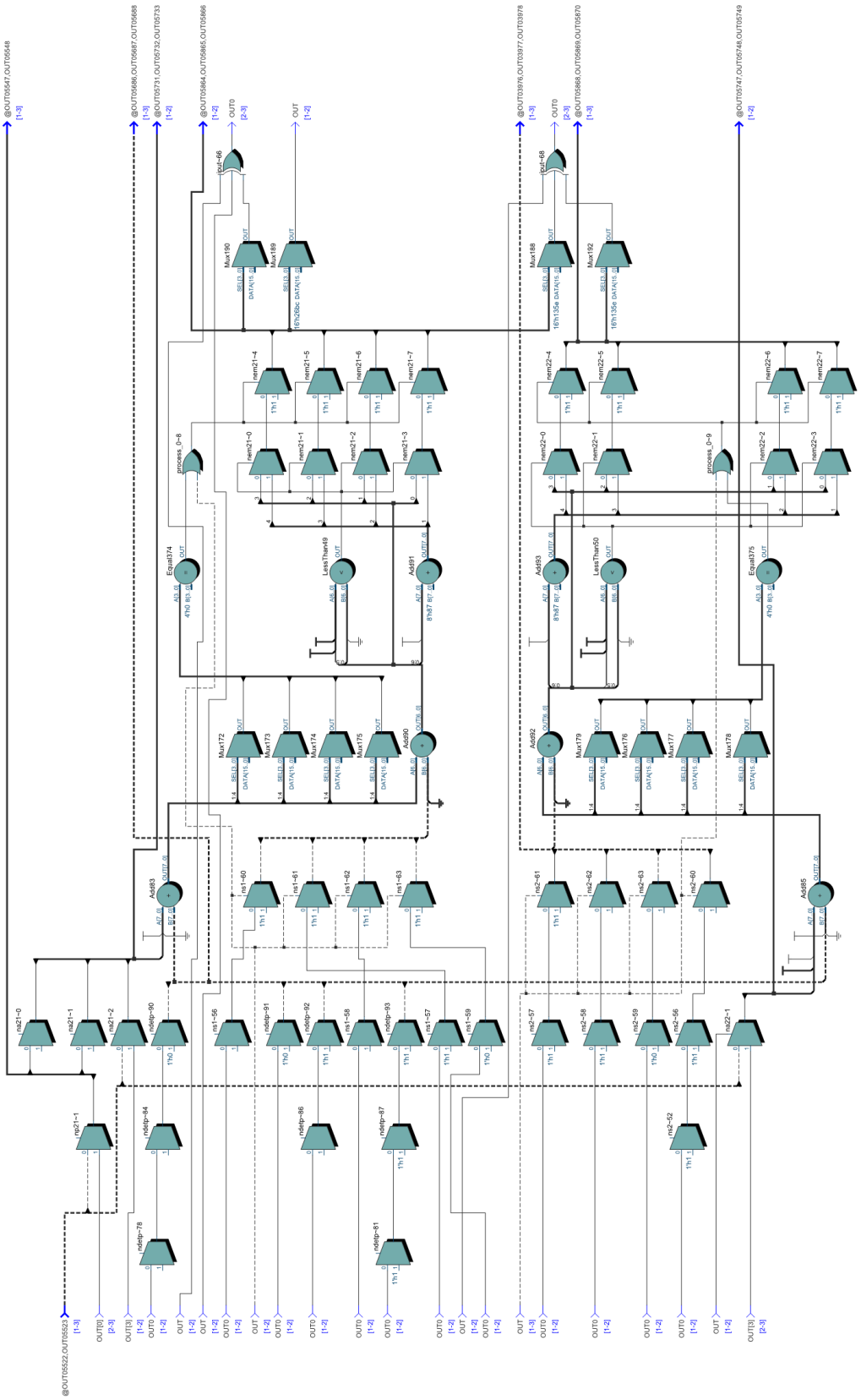


Slika 53: RTL šema kola za računanje stepena onog elementa iz GF(2^4), koji odgovara vrijednosti prvog sindroma

Računanje matrice M i njene daterminante, kao i određivanje lokacija grešaka, obavljaju se na isti način kao što je već prikazano kod BCH dekodera. Lokacije grešaka se smještaju u varijable $nroot1$ i $nroot2$, koje se dalje koriste za konstrukciju matrice P. Elementi matrice P smještaju se u varijable $na11$,$na12$,$na21$ i $na22$ dok se determinanta matrice P smješta u varijablu $ndetp$. Kolo za računanje jednog elementa matrice P prikazano je na slici 50. Sada je potrebno naći invernzu matricu P^{-1} i pomnožiti je sa odgovarajućim sindromom kako bi se dobila vrijednost greške. Vrijednosti grešaka smještaju se u varijable $nem1$ i $nem2$. Konačno, greška se ispravlja primjenom XOR operacije nad simbolom na poziciji $nroot1$ ($nroot2$) u ulaznoj poruci i simbolom koji ima vrijednost $nem1$ ($nem2$). Na slici 51 prikazano je kolo za računanje vrijednosti jedne greške i ispravljanje te greške u originalnoj poruci.

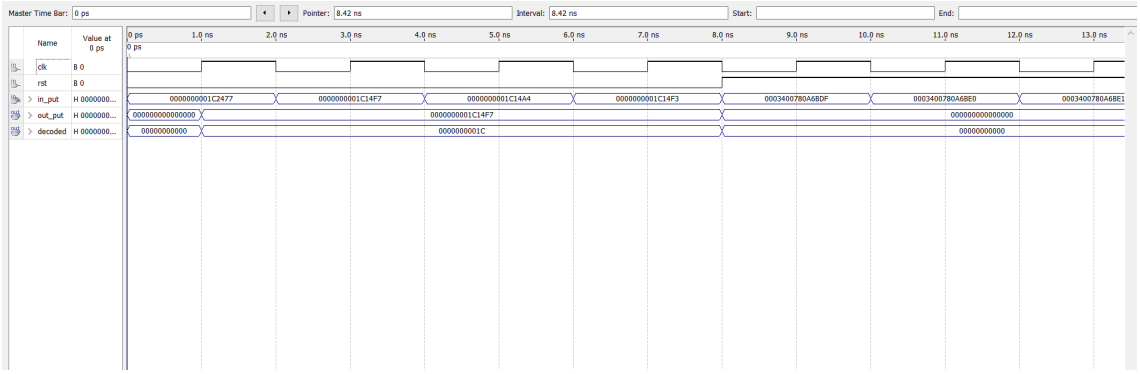


Slika 54: RTL šema kola za računanje jednog elementa matrice P



Slika 55: RTL šema kola za računanje i ispravljanje jedne greške

Kako bi se verifikovala tačnost implementiranog kodera, više različitih vrijednosti ulaznog signala je testirano kroz simulaciju , slika 52. Radi bolje preglednosti ulazni i izlazni podaci su predstavljeni heksadecimalnim zapisom. Za detekciju i korekciju grešaka implementiranim dekoderom potreban je jedan ciklus taktnog signala clk. Prvi testni podatak predstavlja poruku sa greškama na dva simbola, čije je dekodiranje demonstrirano u primjeru 5, poglavlje 2. U drugom taktu nije došlo do greške u poruci. U trećem i četvrtom taktu desile su se greške na zadnja dva i zadnjem simbolu, respektivno. Kako na ulaz dekodera za vrijeme prva četiri takta dolazi ista poruka sa različitim brojem i lokacijama grešaka, a vrijednost signala out_put se ne mijenja za vrijeme prva četiri takta, možemo zaključiti da su sve greške pravilno ispravljene. Nakon četvrtog takta signal rst prelazi na nivo logičke jedinice, pa su svi izlazi resetovani.



Slika 56: Rezultati simulacije za Reed-Solomon (15, 11, 2) dekoder

Ukupna termička disipacija snage iznosi 120.04 mW. Dinamička termička disipacija jezgra je 0.00 mW, statička termička disipacija jezgra je 89.00 mW, a I/O termička disipacija snage iznosi 31.04 mW.

Tabela 17: Upotreba resursa čipa EP3C40F484C6 iz familije Cyclone III, koji je korišćen za hardversku implementaciju Reed-Solomon dekodera (15, 11, 2)

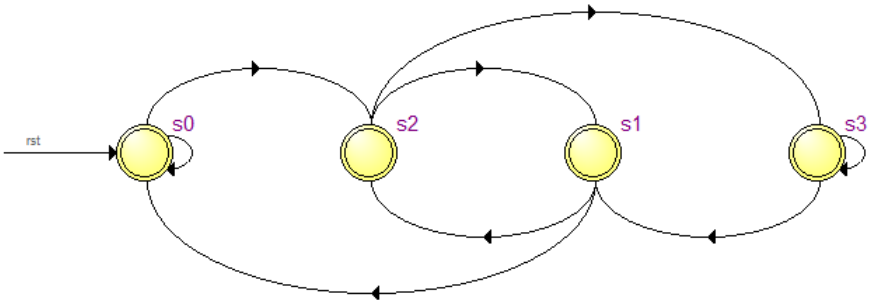
Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C40F484C6	1,515 / 39,600 (4 %)	1,471 / 39,600 (4 %)	104 / 39,600 (< 1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
166 / 332 (50 %)	0	0 / 1,161,216 (0 %)	0 / 252 (0 %)	0 / 4 (0 %)

Hardverska implementacija RS dekodera premašuje kapacitet čipa EP3C5E144C7 iz familije Cyclone III, zbog ograničenja u ukupnom broju pinova. Stoga je dekoder implementiran na čipu EP3C40F484C6 iz iste familije (Tabela 15), koji pruža dovoljan broj pinova za ispunjenje zahtjeva dizajna.

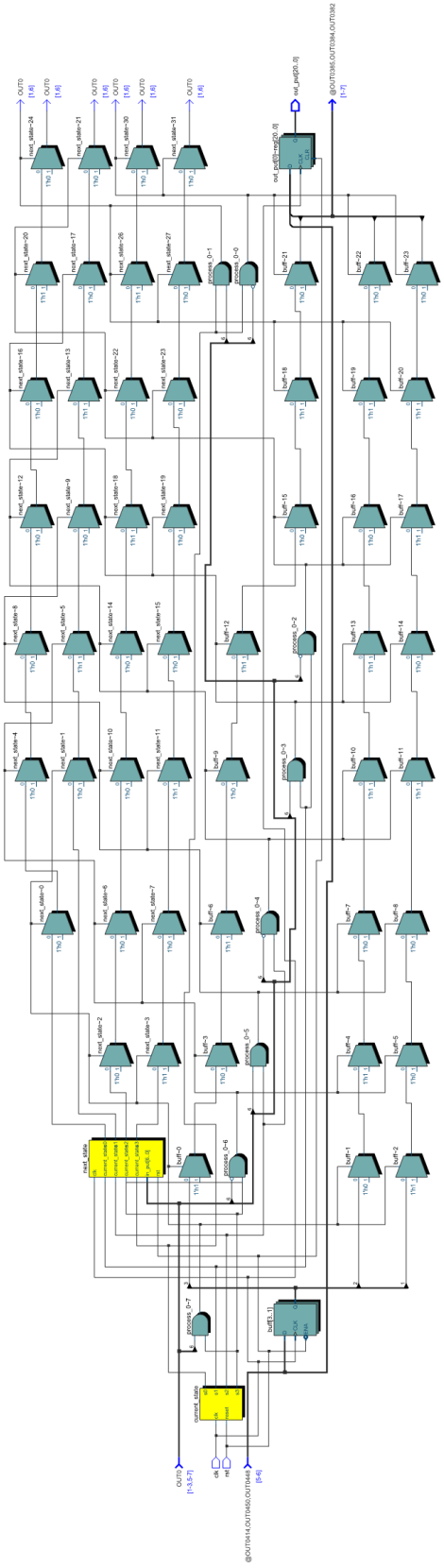
4.6 Konvolucioni kod i Viterbijev algoritam

Sistematski konvolucioni (3, 1, 2) koder realizovan je tako da se za svaki informacioni bit generiše jedan element dužine tri bita na izlazu. Prvi bit svakog izlaznog elementa je informacioni bit, a preostala dva bita su redudantni bitovi. Dakle, implementirani koder ima kodni odnos $R = 1/3$, a ograničenje kodera je $L = 3$. RTL šema implementiranog kodera prikazana je na slici 54. Signal in_put je dužine 7 bita i on predstavlja ulaznu sekvencu informacionih bitova. Signal out_put je dužine 21 bita i on predstavlja izlaznu kodiranu poruku. Signal clk predstavlja taktni signal i izlazne kodirane poruke se generišu na svakoj uzlaznoj ivici ovog signala. Signal rst služi za postavljanje sistema u poznato stanje i kada je on na visokom nivou izlazi su resetovani. Koder je implementiran kao Mealy-eva mašina sa četiri moguća stanja, slika 53. Izlaz u svakom trenutku zavisi od trenutnog stanja i trenutnog ulaza. Trenutno stanje smješta se u varijablu $current\_state$, dok se sljedeće stanje smješta u varijablu $next\_state$. Izlazni element dužine tri bita koji odgovara prelazu iz stanja $current\_state$ u stanje $next\_state$ smješta se u varijablu $buff$.

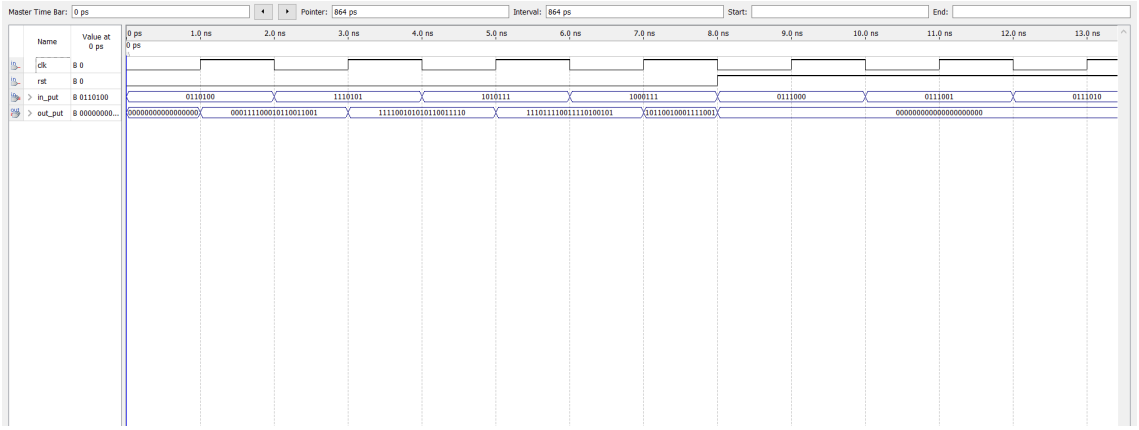
Kako bi se verifikovala tačnost implementiranog kodera, više vrijednosti ulaznog signala in_out je testirano kroz simulaciju , slika 55. Poruka koja se našla na ulazu kodera u prvom taktu predstavlja sekvencu informacionih bitova čije je kodiranje demonstrirano u primjeru 6, poglavlje 2. Izlazna kodirana poruka se generiše prolaskom svakog pojedinačnog informacionog bita kroz pomjerački registar sa slike 18. Ovaj postupak se ponavlja sve dok je singal rst postavljen na nivo logičke nule. Nakon četvrtog takta signal rst prelazi na nivo logičke jedinice, pa su od ovog momenta izlazi resetovani.



Slika 57: Dijagram stanja konvolucionog (3, 1, 2) kodera



Slika 58: RTL šema sistematskog konvolucionog (3, 1, 2) kodera



Slika 59: Rezultati simulacije za sistematski konvolucioni (3, 1, 2) koder

Ukupna termička disipacija snage iznosi 134.99 mW. Dinamička termička disipacija jezgra je 0.00 mW, statička termička disipacija jezgra je 99.11 mW, a I/O termička disipacija snage iznosi 35.88 mW.

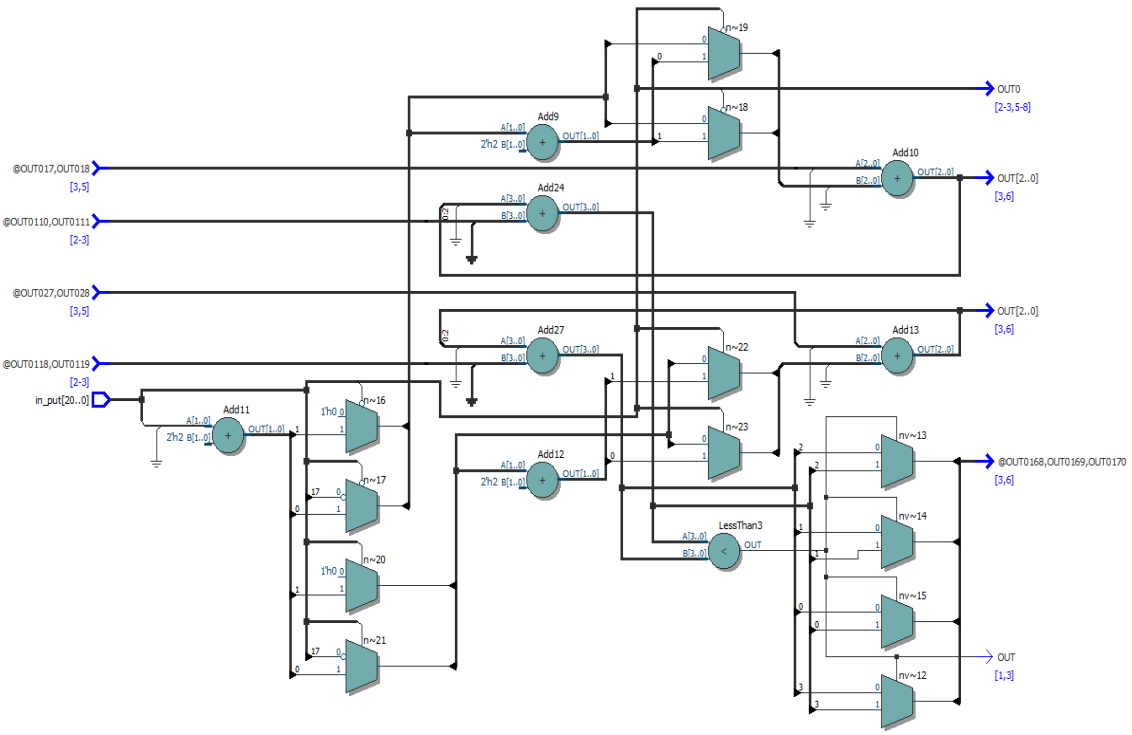
Tabela 18: Upotreba resursa čipa EP3C5E144C7 iz familije Cyclone III, koji je korišćen za hardversku implementaciju sistematskog konvolucinog (3, 1, 2) kodera

Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C5E144C7	21 / 5,136 (< 1 %)	11 / 5,136 (< 1 %)	21/ 5,136 (< 1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
30 / 95 (32 %)	0	0 / 423,936 (0 %)	0 / 46 (0 %)	0 / 2 (0 %)

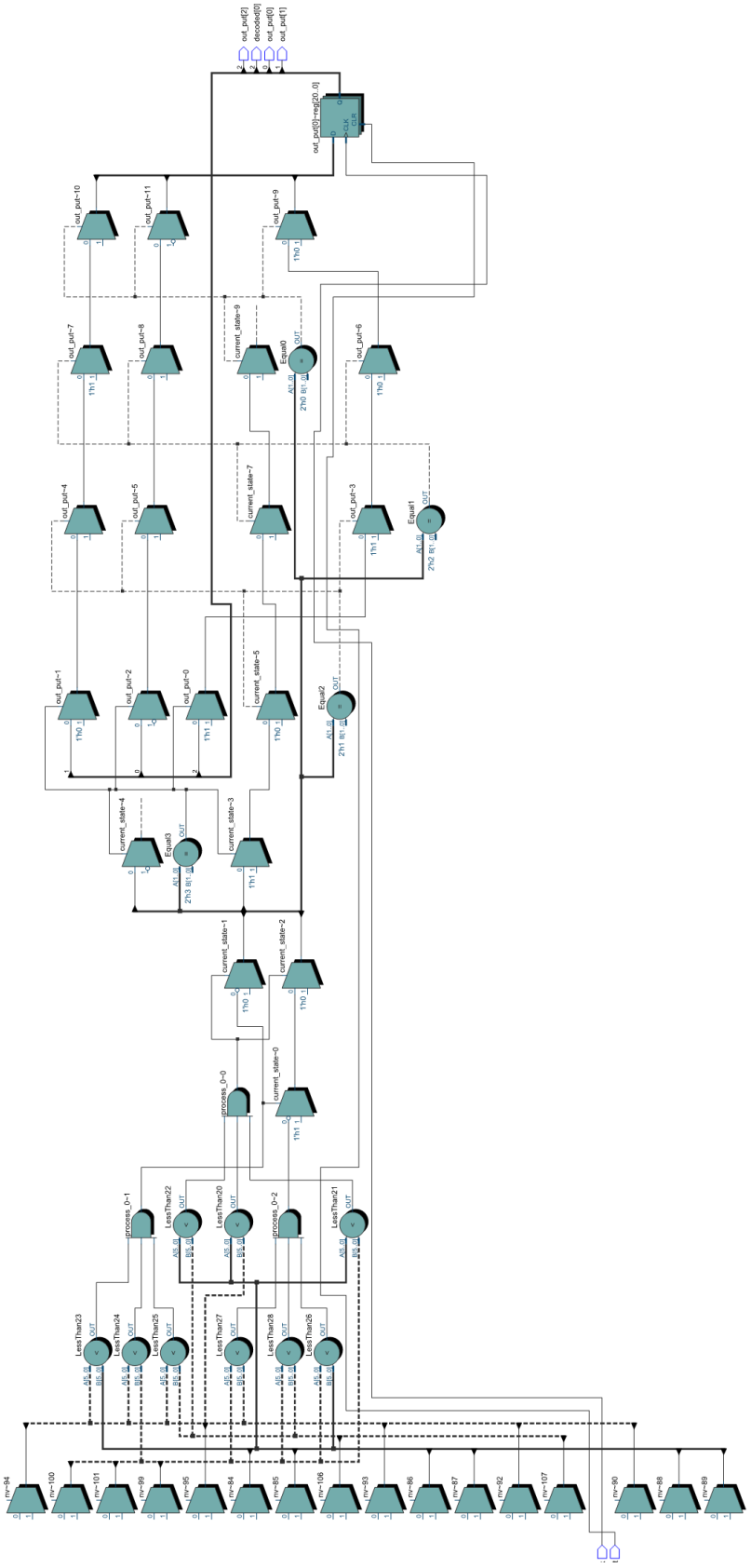
Viterbijev dekoder prima kodiranu poruku dužine 21 bit i na svom izlazu generiše sekvencu informacionih bitova koja se našla na ulazu kodera. Signal in_put predstavlja ulaznu kodiranu poruku, dok signal out_put predstavlja kodiranu poruku nakon detekcije i korekcije eventualnih grešaka. Signal $decoded$ predstavlja sekvencu informacionih bitova koja se našla na ulazu u koder. Prvi korak u Viterbijevom algoritmu je formiranje odgovarajućeg trelis dijagrama. Potrebno je izračunati metriku za svaku granu u dijagramu. Hamingova distanca koju treba sabrati sa metrikom iz prethodne vremenske jedinice smješta se u varijablu n, dok se metrika preživjele grane smješta u dvodimenzionalni niz nv. Nakon računanja metrika za dvije grane koje vode do jednog čvora, potrebno ih je uporediti i u niz nv smjestiti manju vrijednost metrike n. Dakle, ukoliko je $nv(4,3) = 4$ to znači da preživjela

grana koja vodi do trećeg čvora na četvrtom nivou ima vrijednost metrike četiri. RTL šema za računanje jednog elementa niza nv data je na slici 56. Ovaj proces se ponavlja sve do posljednjeg nivoa, nakon čega je formiran trelis dijagram i metrike svih preživjelih grana su smještene u niz nv.

Sada je potrebno pronaći najvjerovatniju putanju preživljavanja u trelis dijagramu. Ovo se postiže prolaskom kroz trelis dijagram, počevši od nivoa $j = 7$, pa sve do nivoa $j = 0$, pri čemu se za prelazak sa jednog nivoa na drugi uvijek koristi preživjela grana, tj. grana čija je metrika smještena u niz nv. Najprije je potrebno pronaći granu sa najmanjom vrijednošću metrike na posljednjem nivou. Ta grana označava početak prolaska kroz trelis dijagram. Ovo se postiže tako što se vrijednosti niza nv za posljednji nivo dovode na ulaz kola sa slike 58. Stanje koje odgovara prelasku sa jednog nivoa na drugi smješta se u varijablu $current\_sate$. Na osnovu promjena vrijednosti ove varijable definišu se vrijednosti izlaznih elemenata, tj. bitovi signala out_put. Vrijednosti signala out_put dovode se na ulaz D flip-flopa, slika 57, kako bi se izlazi generisali na uzlaznoj ivici taktnog signala clk. Svaki treći bit signala out_put je informacioni bit, pa se izdvajanje informacionih bitova i njihovo smještanje u signal $decoded$ obavlja veoma jednostavno. Signal rst služi za postavljanje sistema u poznato stanje i kada je on na visokom nivou svi izlazi su resetovani.

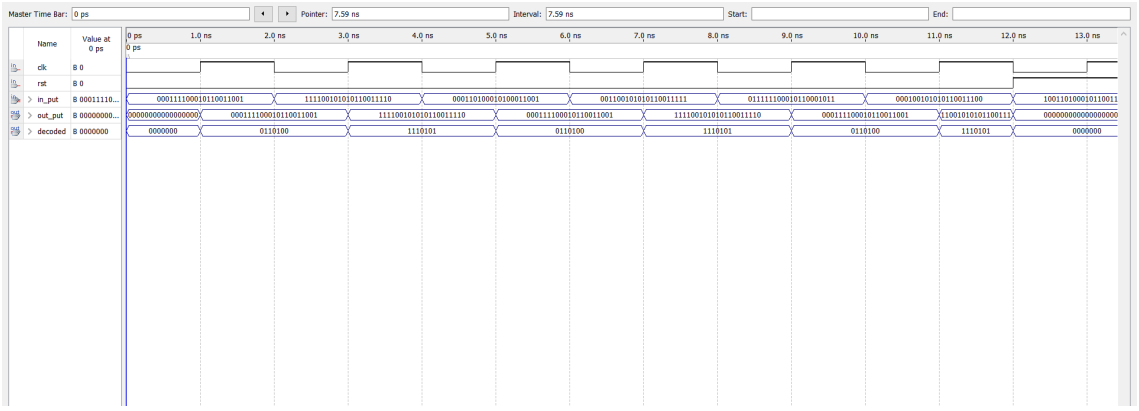


Slika 60: RTL šema kola za računanje vrijednosti metrike jedne preživjele grane



Slika 61: RTL šema kola za traženje najvjerovatnije putanje u trelis dijagramu

Na slici 58 prikazani su rezultati simulacije za implementirani Viterbijev dekoder. Za detekciju i korekciju greške potreban je jedan ciklus taktnog signala clk. U prva dva takta nije došlo do greške u primljenim porukama, što se može zaključiti iz vrijednosti signala out_put koji ima istu vrijednost kao ulazni signal in_put. U trećem taktu dolazi poruka sa dvije greške, čije je dekodiranje demonstrirano u primjeru 6, poglavlje 2. U četrvrtom taktu došlo je do tri greške i to na prva dva i posljednjem bitu. U petom taktu došlo je do četiri greške i to na pozicijama 2, 3, 17 i 20. U šestom taktu došlo je do pet grešaka i to na prva tri i posljednja dva bita. Na osnovu vrijednosti signala out_put zaključujemo da su sve greške pravilno ispravljene. Nakon šestog takta signal rst prelazi na nivo logičke jedinice, pa su od ovog momenta svi izlazi resetovani.



Slika 62: Rezultati simulacije za Viterbijev dekoder

Ukupna termička disipacija snage iznosi 59.80 mW. Dinamička termička disipacija jezgra je 0.00 mW, statička termička disipacija jezgra je 46.14 mW, a I/O termička disipacija snage iznosi 13.65 mW.

Tabela 19: Upotreba resursa čipa EP3C5E144C7 iz familije Cyclone III, koji je korišćen za hardversku implementaciju Viterbijevog dekodera

Chip Family	**Recomended Device**	**Logic Elements**	**Combinational Functions**	**Dedicated Logic Registers**
Cyclone III	EP3C5E144C7	557 / 5,136 (11 %)	557 / 5,136 (11 %)	18 / 5,136 (< 1 %)
Total pins	**Total virtual pins**	**Total memory bits**	**Embedded Multiplier 9-bit elements**	**Total PLLs**
51 / 95 (54 %)	0	0 / 423,936 (0 %)	0 / 46 (0 %)	0 / 2 (0 %)

5 Zaključak

U praktičnim komunikacionim sistemima podaci mogu biti oštećeni šumom tokom prenosa. Kako potražnja za razvojem pouzdanih telekomunikacionih i bežičnih sistema konstantno raste, važno je detektovati i ispraviti greške u informacijama primljenim preko komunikacionih kanala. Zbog ovoga su algoritmi za detekciju i korekciju grešaka od suštinskog značaja u dizajnu komunikacionih sistema za različite primjene.

U okviru drugog poglavlja master rada analaizirani su neki od najčešće upotrebljavanih algoritama za detekciju i korekciju greške: pravougaoni, trougaoni, Hamming-ov, BCH, Reed-Solomon i konvolucioni. Navedena su područija primjene, kao i teorijska ograničenja ovih algoritama. Detaljno su objašnjeni principi kodiranja i dekodiranja podataka pomenutim algoritmima, uz demonstraciju kroz odgovarajuće primjere. Za svaki od opisanih algoritama projektovan je hardver za implementaciju na FPGA tehnologiji. Čip EP3C5E144C7 iz familije Cyclone III korišćen je za implementaciju svih algoritama, osim Reed-Solomon dekodera, čija implementacija premašuje kapacitet pomenutog čipa. Upravo zbog ovoga Reed-Solomon dekoder implementiran je na čipu EP3C40F484C6, koji pruža znatno veći broj pinova i logičkih elemenata. Realizovani koderi i dekoderi testirani su kroz simulaicje, gdje je praćeno njihovo ponašanje za različite vrijednosti ulaznih podataka. U posljednjem poglavlju prikazane su i RTL šeme svih implementiranih algoritama, uz detaljan prikaz resursa koji su korišćeni za hardversku implementaciju svakog od kodera i dekodera.

Implementacija Hammingovog kodera, predložena u poglavlju 4.3, kodira poruku dužine 4 bita i na svom izlazu generiše kodnu riječ dužine 8 bita. Predloženi koder koristi 4 paritetna bita, čime omogućava korekciju jedne greške i detekciju do 2 greške u kodnoj riječi. Ukupan proces kodiranja se odvija za 1 ns, što predstavlja značajno ubrzanje u odnosu na kodere predložene u radovima [2] i [16], kod kojih proces kodiranja traje 1 μs. Slično tome, proces dekodiranja u predloženom riješenju, uključujući detekciju i korekciju grešaka, zahtijeva samo 1 ciklus taktnog signala, trajanja 1 ns, dok u radovima [2] i [16] dekodiranje traje 1 μs. Ovo ukazuje na značajnu prednost predloženog riješenja u smislu brzine generisanja kodne riječi i brzine detekcije i koreckije grešaka.

Implementacija BCH koda, predstavljena u poglavlju 4.4 koristi, paralelnu arhitekturu, što predstavlja značajnu prednost u odnosu na serijske arhiteture opisane u radovima [7], [8] i [12]. Predloženi BCH koder prima poruku dužine 5 bita i generiše kodnu riječ dužine 15 bita, dok implementirani dekoder omogućava detekciju i korekciju do 3 greške u kodiranoj poruci. U radu [7], BCH koder obrađuje poruku

dužine 7 bita i generiše kodnu riječ iste dužine od 15 bita, ali dekoder ispravlja najviše 2 greške u kodiranoj poruci, a serijska arhitektura zahtijeva 15 ciklusa taktnog signala za kodiranje i korekciju grešaka. Glavna prednost predloženog rješenja se dakle ogleda u vremenu potrebnom za detekciju i korekciju greške, kako predloženi dekoder obavlja ovaj proces za 1 ns, dok dekoderi predloženi u radovima [7], [8] i [12] obavljaju ovaj proces za 100 ns, 1.09 μs i 10 μs, respektivno. Ova vremenska efikasnost postignuta je po cijenu povećane složenosti dizajna i veće potrošnje FPGA resursa.

Implementacija Reed-Solomon koda predstavljena u poglavlju 4.5 prima poruku dužine 11 simbola od 4 bita i generiše kodnu riječ dužine 15 simbola, dok implementacija iz rada [9] obrađuje poruku dužine 8 simbola i generiše kodnu riječ dužine 12 simbola. Oba dekodera su u stanju da detektuju greške na 2 simbola, ali predložena paralelna arhitektura omogućava realizaciju svih operacija u jednom ciklusu taktnog signala trajanja 1 ns, za razliku od serijske arhitekture iz rada [9], koja zahtijeva 12 ciklusa taktnog signala. Iako predložena riješenja postižu značajno unapređenje u pogledu brzine, njihova složenost je povećana zbog korišćenja većeg broja logičkih elemenata.

Kovolucioni koder implementiran u poglavlju 4.6 predstavlja sistematski koder sa kodnim odnosom 1/3. Ključna karakteristika ovog kodera je da ulazna poruka ostaje sadržana u kodiranoj poruci, što omogućava lakšu rekonstrukciju originalnih podataka. Predložena arhitektura omogućava generisanje kodne riječi u jednom ciklusu taktnog signala trajanja 1 ns, dok je u radu [10] za isti proces potrebno 1,5 ns, čime je postignuto unapređenje u brzini kodiranja. Poredeći Viterbi dekodere, implementacija predložena u ovom radu koristi 557 logičkih elemenata, dok implementacije predložene u radovima [10] i [41] koriste značajno manji broj logičkih elemenata, što ukazuje na veću složenost predloženog riješenja. U pogledu performansi, proces detekcije i korekcije grešaka u predloženom Viterbi dekoderu realizuje se u jednom ciklusu taktnog signala trajanja 1 ns, dok je u radovima [10], i [41] za isti proces potrebno 200 ns i 550 ns, respektivno.

Oslanjajući se na rezlutate ovog rada, dalji pravci u istraživanju bi se mogli odnositi na dodatno redukovanje potrebnih resursa za hardversku realizaciju algoritama. Pored toga, u budućem radu bi mogla biti razmatrana mogućnost paralelizacije algoritama, radi ubrzanja procesa obrade podataka, odnosno skraćenja vremena izvršavanja. Dalje istraživanje bi se, takođe, odnosilo na analizu i optimizaciju latencije, odnosno kašnjenja pojednih komponenti u projektovanim sistemima, a u cilju poboljšanja sveukupnih perfromansi sistema.

6 Literatura

[1] Dominguez Y Sainza et al. *Locator decoding for BCH codes*. PhD thesis, Faculty of Science and Engineering, 2001.

[2] Divya Mokara, Sushmi Naidu, and Akash Kumar Gupta. Design and implementation of hamming code using vhdl & dsch. *International Journal of Latest Engineering Research and Applications*, 2:33–40, 2017.

[3] Jiri Adamek. *Foundations of coding: Theory and applications of error-correcting codes with an introduction to cryptography and information theory*. John Wiley & Sons, 2011.

[4] Kenny Chung Chung Wai. Fpga implementation of reed solomon codec for 40gbps forward error correction in optical networks. Thesis, Rochester Institute of Technology, 2006.

[5] Igor Djurović. *Teorija informacija i kodova*. Univerzitet Crne Gore, Elektrotehnički fakultet, Podgorica, 2023.

[6] Vijayakumara M, Byregowda K, Pradeep S, Shashwat Singh Raghav, Nataraja M, and Sheshappa N. A vlsi implementation of hamming code algorithm using fpga architecture. *International Journal of VLSI & Signal Processing*, 7:29–35, 08 2020.

[7] S Rohith and S Pavithra. Fpga implementation of (15, 7) bch encoder and decoder for text message. *International Journal of Research in Engineering and Technology*, 2(9):209–214, 2013.

[8] Hardik Sutaria and Deepti Khurge. Fpga based bch decoder. *International Journal for Scientific Research & Development*, 1(3):665, 2013.

[9] P Parvathi and P Rajendra Prasad. Fpga based design and implementation of reed-solomon encoder & decoder for error detection and correction. In *2015 Conference on Power, Control, Communication and Computational Technologies for Sustainable Growth (PCCCTSG)*, pages 261–266. IEEE, 2015.

[10] S Ranjitha, K Divya Preethi, and K Megha. An efficient fpga implementation of convolutional encoder and viterbi decoder for dsp applications. *International Journal of Engineering Research & Technology (IJERT Vol. 4 Issue 10*, 4(10):387–390, 2015.

[11] J Patel Pratik and Tarunkumar C Lad. Analysis and perform an convolutional encoder. *International Journal for Scientific Research & Development*, 1(10):2256–2259, 2013.

[12] Samir Jasam Mohammed and Hayder Fadhil Abdulsada. Fpga implementation of 3 bits bch error correcting codes. *International Journal of Computer Applications*, 71(7):35, 2013.

[13] Laurenţiu Mihai Ionescu, Constantin Anton, Ion Tutănescu, Alin Mazăre, and Gheorghe Şerban. Hardware implementation of bch error-correcting codes on a fpga. *International Journal of Intelligent Computing Research (IJICR)*, 1(3):148–153, 2010.

[14] J. I. Hall. *Notes on Coding Theory*. Department of Mathematics, Michigan State University, East Lansing, MI, USA, 2010. September 9, 2010.

[15] Saneep Kaur. *VHDL Implementation of REED-SOLOMON Codes*. PhD thesis, Citeseer, 2007.

[16] Debalina Roy Choudhury and Krishanu Podder. Design of hamming code encoding and decoding circuit using transmission gate logic. *International Research Journal of Engineering and Technology*, 2(7):1165–1169, 2015.

[17] Priyanka Dayal and Rajeev Kumar Patial. Fpga implementation of reed-solomon encoder and decoder for wireless network 802. 16. *International Journal of Computer Applications*, 68(16), 2013.

[18] Ranjita Singh. Vhdl implementation of reed-solomon encoder-decoder for wimax network. *International Journal of Analytical, Experimental and Finite Element Analysis*, 3(1):126–132, 2014.

[19] Veeru Sb. Paper: Vhdl implementation of convolutional encoder and viterbi decoder. 01 2014.

[20] Kanchana Katta. Design of convolutional encoder and viterbi decoder using matlab. *International Journal for Reasearch in Emerging Science and Technology*, 1(7):10–15, 2014.

[21] Romi Banerjee, Saptarshi Naskar, and Samar Sen Sarma. A closer look into the rectangular codes: Where the parity check predominates. *Journal of Global Research in Computer Science*, 3(3):17–23, 2012.

[22] Jaya Kumari Verma and Shipra Varshney. Performance assessment of hamming code. In *Proc. of the International Conference on Science and Engineering (ICSE 2011).*, 2011.

[23] Himanshu Sharma and Amit Kumar. Hamming code for error detection and corection using vhdl. *International Journal Of Engineering Research & Management Technology*, 1, 2014.

[24] Ravi Hosamani and Ashwini S Karne. Design and implementation of hamming code on fpga using verilog. *International Journal of Engineering and Advanced Technology (IJEAT)*, 4(2):180–184, 2014.

[25] AE Makhloufi, SE Adib, and Naoufal Raissouni. Highly efficient security level implementation in radiation-tolerance fpga using a combination of aes algorithm and hamming code: Lst-sw case. *Int. J. Electr. Electron. Eng. Telecommun*, 12(4):1–12, 2022.

[26] Amit Kumar Panda, Shahbaz Sarik, and Abhishek Awasthi. Fpga implementation of encoder for (15, k) binary bch code using vhdl and performance comparison for multiple error correction control. In *2012 International Conference on Communication Systems and Network Technologies*, pages 780–784. IEEE, 2012.

[27] VP Mahadevaswamy, SL Sunitha, and BN Shobha. Implementation of fault tolerant method using bch code on fpga. *International Journal of Soft Computing and Engineering (IJSCE)*, 2(4):2231–2307, 2012.

[28] Meera Srinivasan and Dilip V Sarwate. Malfunction in the peterson-gorenstein-zierler decoder. *IEEE transactions on information theory*, 40(5):1649–1653, 1994.

[29] Jürgen Freudenberger, Daniel Nicolas Bailon, and Malek Safieh. Reduced complexity hard-and soft-input bch decoding with applications in concatenated codes. *IET circuits, devices & systems*, 15(3):284–296, 2021.

[30] Sheng-Feng Wang, Huai-Yi Hsu, and An-Yeu Wu. A very low-cost multi-mode reed-solomon decoder based on peterson-gorenstein-zierler algorithm. In *2001 IEEE Workshop on Signal Processing Systems. SiPS 2001. Design and Implementation (Cat. No. 01TH8578)*, pages 37–48. IEEE, 2001.

[31] Diplaxmi Chaudhari, Mayura Bhujade, and Pranali Dhumal. Vhdl design and fpga implementation of reed solomon encoder and decoder for rs (7, 3). *Inter-*

national Journal of Science, Engineering and Technology Research (IJSETR), 3(3):563, 2014.

[32] Vishal Chandale, Pallavi Gavali, Payal Giri, and Anjali Shrivastav. Fpga based error detection and correction system using reed-solomon code. *International Research Journal of Engineering and Technology*, 3(5):1446–1450, 2016.

[33] Gabriel Marchesan Almeida, Eduardo Augusto Bezerra, Luis Vitorio Cargnini, Rubem Dutra Ribeiro Fagundes, and Daniel Gomes Mesquita. A reed-solomon algorithm for fpga area optimization in space applications. In *Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007)*, pages 243–249. IEEE, 2007.

[34] Aqib Al Azad and Md Imam Shahed. A compact and fast fpga based implementation of encoding and decoding algorithm using reed solomon codes. *International Journal of Future Computer and Communication*, 3(1):31–35, 2014.

[35] Valentina Bianchi, Marco Bassoli, and Ilaria De Munari. Comparison of fpga and microcontroller implementations of an innovative method for error magnitude evaluation in reed–solomon codes. *Electronics*, 9(1):89, 2020.

[36] Mustapha Elharoussi, Asmaa Hamyani, and Mostafa Belkasmi. Vhdl design and fpga implementation of a parallel reed-solomon (15, k, d) encoder/decoder. *International Journal of Advanced Computer Science and Applications*, 4(1), 2013.

[37] KV Mohan. Vhdl implementation of reed solomon coding. *International Journal of Advances in Engineering Research*, 7(2):45–54, 2014.

[38] Stephen B Wicker and Vijay K Bhargava. An introduction to reed-solomon codes. *Reed-Solomon codes and their applications*, pages 1–16, 1994.

[39] Amandeep Singh and Mandeep Kaur. Design and implementation of reed solomon encoder on fpga. *International Journal of Information and Communication Engineering*, 7(9):1248–1250, 2013.

[40] J Tulasi, T Venkata Lakshmi, and M Kamaraju. Fpga implementation of convolutional encoder and hard decision viterbi decoder. *International Journal of Computer & Communication Technology (IJCCT)*, 3(4):2231–0371, 2012.

[41] Yan Sun and Zhizhong Ding. Fpga design and implementation of a convolutional encoder and a viterbi decoder based on 802.11 a for ofdm. *Wireless Engineering and Technology*, 3(3):125–131, 2012.

[42] Sajjad Ahmed Ghauri, Hasan Humayun, Muhammad Ehsan ul Haq, and Farhan Sohail. Implementation of convolutional codes on fpga. In *2012 International Conference for Internet Technology and Secured Transactions*, pages 175–178. IEEE, 2012.

[43] Komal Wayal, Kalpana Gore, Smita Waikule, and SC Wagaj. Convolution encoding and viterbi decoding based on fpga using vhdl. *International Journal of Advanced Technology in Engineering and Science*, 3(01):622–626, 2015.

[44] G David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[45] KS Arunlal and SA Hariprasad. An efficient viterbi decoder. *International Journal of Computer Science, Engineering and Applications*, 2(1):95, 2012.

Izjava o autorstvu

Potpisani-a Aleksej Vukčević
Broj indeksa/upisa 10/22

Izjavljujem

da je master rad pod naslovom:
Projektovanje i analiza hardvera za realizaciju odabranih algoritama za detekciju i korekciju grešaka

- rezultat sopstvenog istraživačkog rada,
- da predloženi master rad ni u cjelini ni u djelovima nije bio predložen za dobijanje bilo koje diplome prema studijskim programima drugih ustanova visokog obrazovanja,
- da su rezultati korektno navedeni, i
- da nijesam povrijedio/la autorska i druga prava intelektualne svojine koja pripadaju trećim licima.

Potpis magistranda

U Podgorici, dana 05.06.2025. godine

Aleksej Vukčević

Izjava o istovjetnosti štampane i elektronske verzije master rada

Ime i prezime autora Aleksej Vukčević
Broj indeksa/upisa 10/22
Studijski program Računari
Naslov rada Projektovanje i analiza hardvera za realizaciju odabranih algoritama za detekciju i korekciju greške
Mentor Prof. dr. Nevena Rašović

Potpisani/a Aleksej Vukčević

Izjavljujem da je štampana verzija mog master rada istovjetna elektronskoj verziji koju
sam predao/la za objavljivanje u Digitalni arhiv Univerziteta Crne Gore.

Istovremeno izjavljujem da dozvoljavam objavljivanje mojih ličnih podataka u vezi sa dobijanjem akademskog naziva master nauka, odnosno zvanja master umjetnosti, kao što su ime i prezime, godina i mjesto rođenja, naslov master rada i datum odbrane rada.

Potpis magistranda

U Podgorici, dana 05.06.2025 godine

Aleksej Vukčević

IZJAVA O KORIŠĆENJU

Ovlašćujem Univerzitetsku biblioteku da u Digitalnom arhivu Univerziteta Crne Gore pohrani moj master rad pod nazivom: Projektovanje i analiza hardvera za realizaciju zadatnih algoritama za detekciju i korekciju greške

koji je moje autorsko djelo.

Master rad sa svim prilozima predao/la sam u elektronskom formatu pogodnom za trajno arhiviranje.

Moj master rad pohranjen u Digitalnom arhivu Univerziteta Crne Gore mogu da koriste svi koji poštuju odredbe sadržane u odabranom tipu licence Kreativne zajednice (Creative Commons) za koju sam se odlučio/la.

1. Autorstvo
2. Autorstvo – nekomercijalno
3. Autorstvo – nekomercijalno – bez prerade
4. Autorstvo – nekomercijalno – dijeliti pod istim uslovima
5. Autorstvo – bez prerade
6. Autorstvo – dijeliti pod istim uslovima

(Molimo da zaokružite samo jednu od šest ponuđenih licenci, kratak opis licenci dat je na poleđini lista).

Potpis magistranda

U Podgorici, dana 05.06.2025. godine

Aleksej Vukičević

1. Autorstvo - Dozvoljavate umnožavanje, distribuciju i javno saopštavanje djela, i prerade, ako se navede ime autora na način određen od strane autora ili davaoca licence, čak i u komercijalne svrhe. Ovo je najslobodnija od svih licenci.

2. Autorstvo - nekomercijalno. Dozvoljavate umnožavanje, distribuciju i javno saopštavanje djela, i prerade, ako se navede ime autora na način određen od strane autora ili davaoca licence. Ova licenca ne dozvoljava komercijalnu upotrebu djela.

3. Autorstvo - nekomercijalno - bez prerade. Dozvoljavate umnožavanje, distribuciju i javno saopštavanje djela, bez promjena, preoblikovanja ili upotrebe djela u svom djelu, ako se navede ime autora na način određen od strane autora ili davaoca licence. Ova licenca ne dozvoljava komercijalnu upotrebu djela. U odnosu na sve ostale licence, ovom licencom se ograničava najveći obim prava korišćenja djela.

4. Autorstvo - nekomercijalno - dijeliti pod istim uslovima. Dozvoljavate umnožavanje, distribuciju i javno saopštavanje djela, i prerade, ako se navede ime autora na način određen od strane autora ili davaoca licence i ako se prerada distribuira pod istom ili sličnom licencom. Ova licenca ne dozvoljava komercijalnu upotrebu djela i prerade.

5. Autorstvo - bez prerade. Dozvoljavate umnožavanje, distribuciju i javno saopštavanje djela, bez promjena, preoblikovanja ili upotrebe djela u svom djelu, ako se navede ime autora na način određen od strane autora ili davaoca licence. Ova licenca dozvoljava komercijalnu upotrebu djela.

Autorstvo - dijeliti pod istim uslovima. Dozvoljavate umnožavanje, distribuciju i javno saopštavanje djela, i prerade, ako se navede ime autora na način određen od strane autora ili davaoca licence i ako se prerada distribuira pod istom ili sličnom licencom. Ova licenca dozvoljava komercijalnu upotrebu djela i prerada. Slična je softverskim licencama, odnosno licencama otvorenog koda