UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Generator Hammingove (7,4) kode

1. SEMINARSKA NALOGA PRI PREDMETU
OPTIČNE IN NANOTEHNOLOGIJE

AVTORJI:

Luka Andrejak, 63050054

Jani Ježˇ, 63050051

Marko Turšič, 63050121

Simon Struna, 63050111

Ljubljana, 2010

Kazalo

1 Uvod

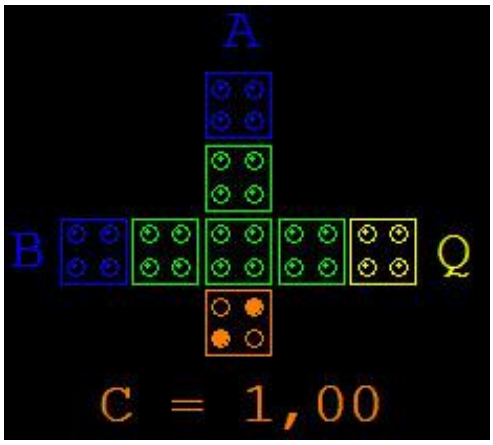
Naloga seminarske naloge je realizacija generatorja Hammingovega (7,4) koda in logike za detekcijo ter korekcijo enobitne napake. Celotna rešitev mora biti realizirana s pomočjo kvantnih celičnih avtomatov (QCA).

Problema smo se lotili s programom QCA Designer. To je orodje za izdelavo in simulacijo QCA, katerega osnovni gradniki so kvantne celice. S povezovanjem teh celic v kompleksnejše strukture pa lahko gradimo razne logične funkcije.

Naša rešitev je sestavljena iz dveh delov. Prvi del deluje kot oddajnik v katerem smo realizirali generiranje Hammingovega koda glede na vhodni podatek. V drugem delu pa je realizirana detekcija in korekcija napak in deluje kot sprejemnik odporen na enobitne napake.

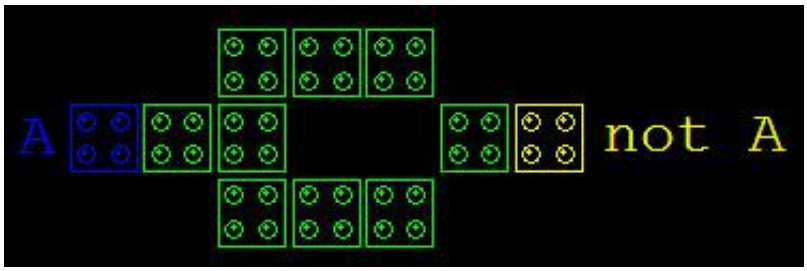
2 Uporabljene QCA strukture

Osnovna logična vrata v QCA so majoritetna vrata. Majoritetna vrata lahko vršijo funkciji AND in OR. Na spodnji sliki (slika 1) so prikazana majoritetna vrata s tremi vhodi in izhodom Q. Logično funkcijo vrat krmilimo z vhodom C. V kolikor je ta postavljen na 1 imajo vrata logično funkcijo OR, sicer pa AND.

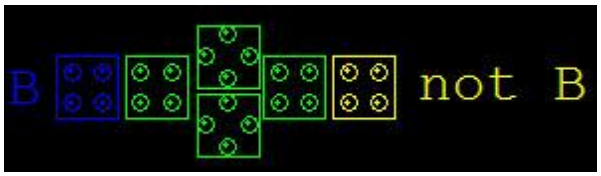


Slika 1: Majoritetna vrata z logično funkcijo OR

Negacijo lahko dosežemo na več načinov. Mi smo uporabili dva načina za negiranje:

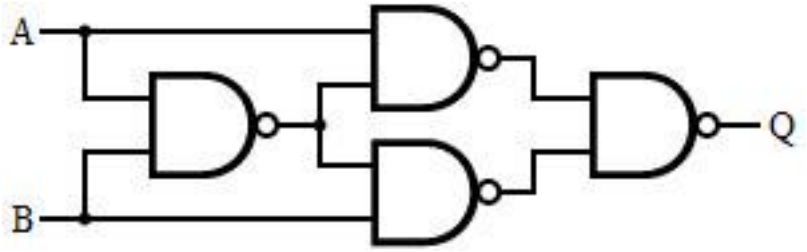


Slika 2: Negator

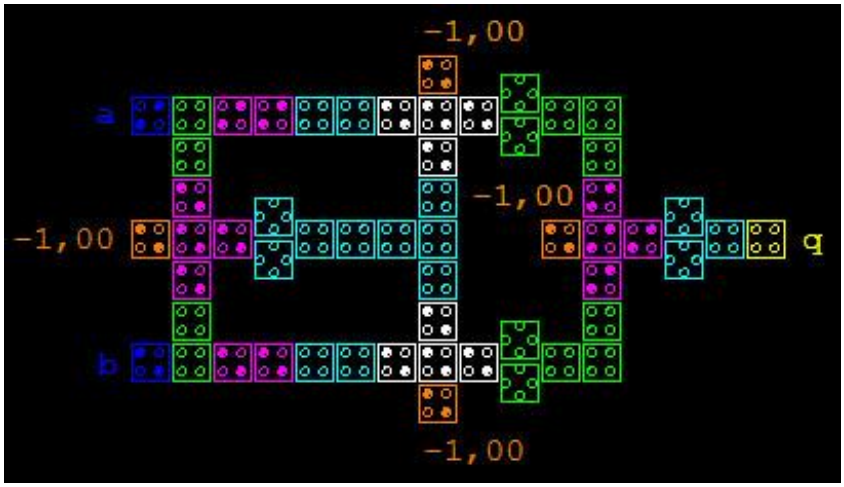


Slika 3: Negator z uporabo invertirajoče linije

Logičnih vrat s funkcijo XOR v QCA ni, zato smo morali XOR vrata zgraditi s pravilno kombinacijo AND in OR vrat. Stabilna XOR vrata smo realizirali s štirimi NAND vrati.



Slika 4: Logična shema XOR vrat zgrajenih z NAND vrati



Slika 5: XOR vrata v QCA Designerju zgrajena iz 4 NAND vrat

3 Hammingov kod

Hammingov kod je metoda, ki pri digitalnih prenosih podatkov omogoča zaznavanje in popravljanje napake. Deluje tako, da željen 4-biten podatek razširimo z dodatnimi paritetnimi biti, s pomočjo katerih lahko sprejemna naprava zazna in odpravi napake.

Z uporabo Hammingovega koda lahko zaznamo katerokoli dvobitno napako, popravimo pa samo enobitne napake. Z drugimi besedami bi lahko rekli, da je lahko Hammingova razdalja med poslanim in prejetim podatkom največ ena, da lahko popravimo morebitne napake.

Ključ Hammingovega koda so paritetni biti s katerimi razširimo originalen podatek. Ti biti so urejeni tako, da različni napačni biti generirajo različne rezultate. Tako lahko prepoznamo in popravimo napačne bite. V 7-bitnem podatku lahko pride do sedmih različnih enobitnih napak, zato potrebujemo 3 paritetne bite.

	d1	d2	d3	d4
p1	Ne	Da	Da	Da
p2	Da	Ne	Da	Da
p3	Da	Da	Ne	Da

Tabela 1: Pokrivanje podatkovnih bitov

Iz tabele 1 vidimo, da so kombinacije paritetnih bitov, ki pokrivajo podatkovne različne, kar je ključnega pomena za popravljanje točno določene napake.

Hammingov kod lahko izračunamo s pomočjo dveh matrik: generacijske matrike G in matrike za preverjanje paritete H.

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \qquad H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Če je vhodni podatek $p = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \end{bmatrix}$, potem izračunamo Hammingov tako, da pomnožimo $x = G \times p^T$. Če med prenosom ni prišlo do napak je sprejet podatek r enak oddanemu x in je rezultat $z = H \times r = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$, kjer je z sindrom napake in enolično identificira napačno prejeti bit.

Če tabelo 1 preberemo po stolpcih, hitro vidimo povezavo med kombinacijami sindromskega vektorja in napakami določenega bita.

Na primer: če je $z = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix}^T$ gre za napako prvega bita, itd.

4 Implementacija

Pri implementaciji smo imeli kar nekaj težav. Ena večjih je bilo križanje linij, zato smo se odločili za večnivojsko rešitev. Na koncu se je izkazalo, da je taka rešitev stabilnejša in na nek način preglednejša ter lažja za realizacijo.

Ker smo implementirali generiranje Hammingovega koda in odkrivanje ter popravljanje napak smo nalogo razdelili na oddajni in sprejemni del.

4.1 Oddajnik

Naloga oddajnika je generirati Hammingov kod na podlagi vhodnega (4-bitnega) podatka. To naredimo tako da generacijsko Hammingovo matriko pomnožimo s transponiranim vhodnim vektorjem. Vzemimo, da je $p = \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix}$ in

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$, kar nam da rezultat $x = G \times p^T = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$

Če vektor x zapišemo po komponentah, vidimo razporeditev podatkovnih in paritetnih bitov

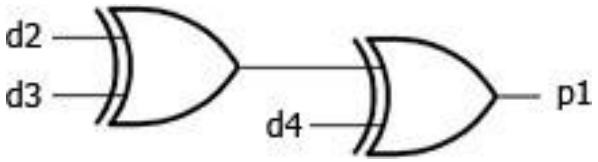
$$x = \begin{bmatrix} d_1 & d_2 & d_3 & d_4 & p_1 & p_2 & p_3 \end{bmatrix}^T$$

Paritetne bite smo glede na generatorsko matriko G izračunali na nasledni način:

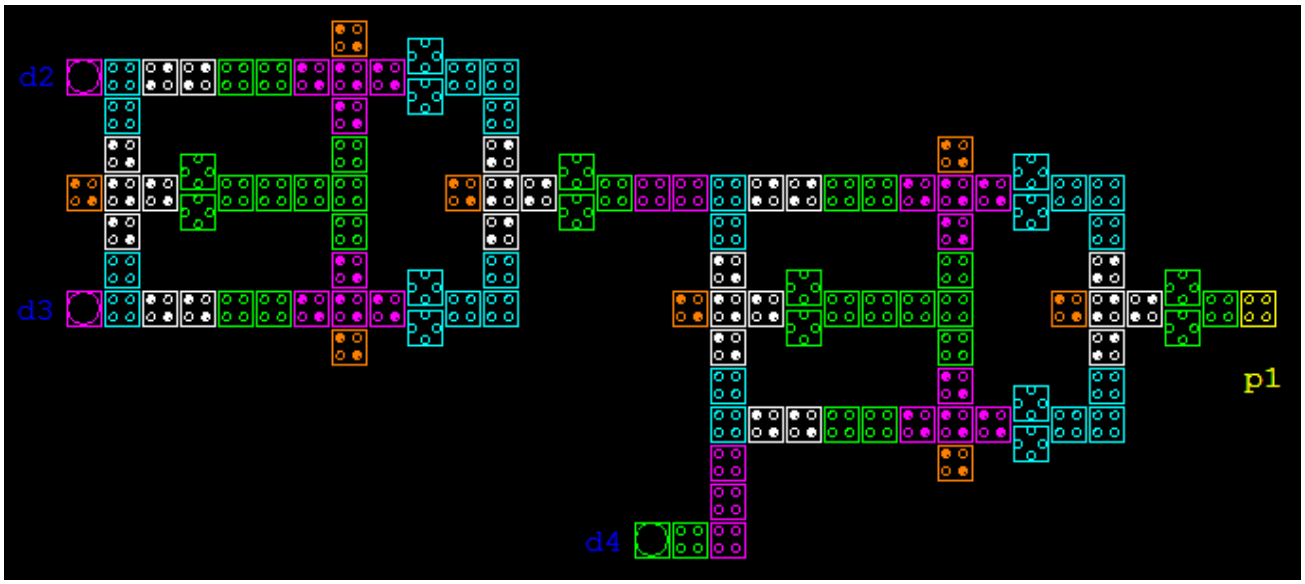
$$p_1 = d_2 \oplus d_3 \oplus d_4$$

$$p_2 = d_1 \oplus d_3 \oplus d_4$$

$$p_3 = d_1 \oplus d_2 \oplus d_4$$



Slika 6: Logična shema generiranja paritetnega bita p1



Slika 7: Generiranje paritetnega bita p1 v QCA Designerju

4.2 Sprejemnik

Naloga sprejemnika je odkrivanje in popravljanje napak. Ker vemo pomen bitov na vhodu ne potrebujemo posebne logike za dekodiranje.

Vhod v sprejemnik je 7-bitni vektor $i = [i_1 \quad i_2 \quad i_3 \quad i_4 \quad i_1 \quad i_2 \quad i_3]$. Vemo, da prvi štirje biti (i_1 ... i_4) predstavljajo podatek, zadnji trije pa pariteto.

Napako detektiramo tako, da izračunamo sindromski vektor. Matriko za preverjanje paritete H pomnožimo z vhodnim vektorjem i. Če vzamemo za primer $i = [1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1]$ in je matrika

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}, \qquad z = H \times i^T = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

vidimo da napake ni bilo, ker je rezultat 0. Če pa bi prišlo do napake na kateremu od bitov, bi dobili eno od kombinacij sindromskega vektorja, ki so tabelirane v spodnji tabeli.

Sindromski vektor	i1	i2	i3	i4	i5	i6	i7
z1	0	1	1	1	1	0	0
z2	1	0	1	1	0	1	0
z3	1	1	0	1	0	0	1

Tabela 2: Kombinacije vhodnega vektorja za detekcijo napake na določenem bitu vhodnega vektorja

Komponente sindromskega vektorja izračunamo na naslednji način:

$$z_1 = i_2 \oplus i_3 \oplus i_4 \oplus i_5$$

$$z_2 = i_1 \oplus i_3 \oplus i_4 \oplus i_6$$

$$z_3 = i_1 \oplus i_2 \oplus i_4 \oplus i_7$$

Če je katerakoli od komponent 1 vemo, da je prišlo do napake, zato lahko napako detektiramo s preprosto logično funkcijo OR

$$napaka = z_1 \vee z_2 \vee z_3$$

V tem primeru odkrijemo mesto napake tako, da komponente sindromskega vektorja konjugiramo po tabeli 2.

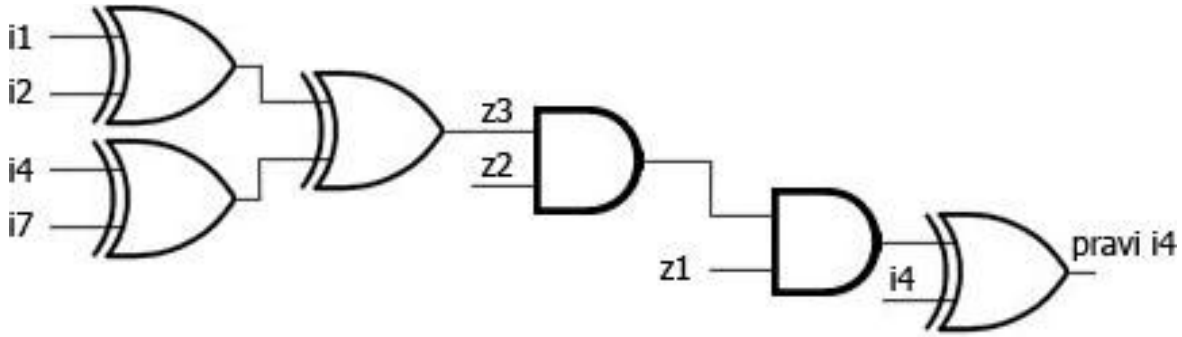
$$napaka\ na\ 1.bitu = \overline{z_1} \wedge z_2 \wedge z_3$$

$$napaka\ na\ 2.bitu = z_1 \wedge \overline{z_2} \wedge z_3$$

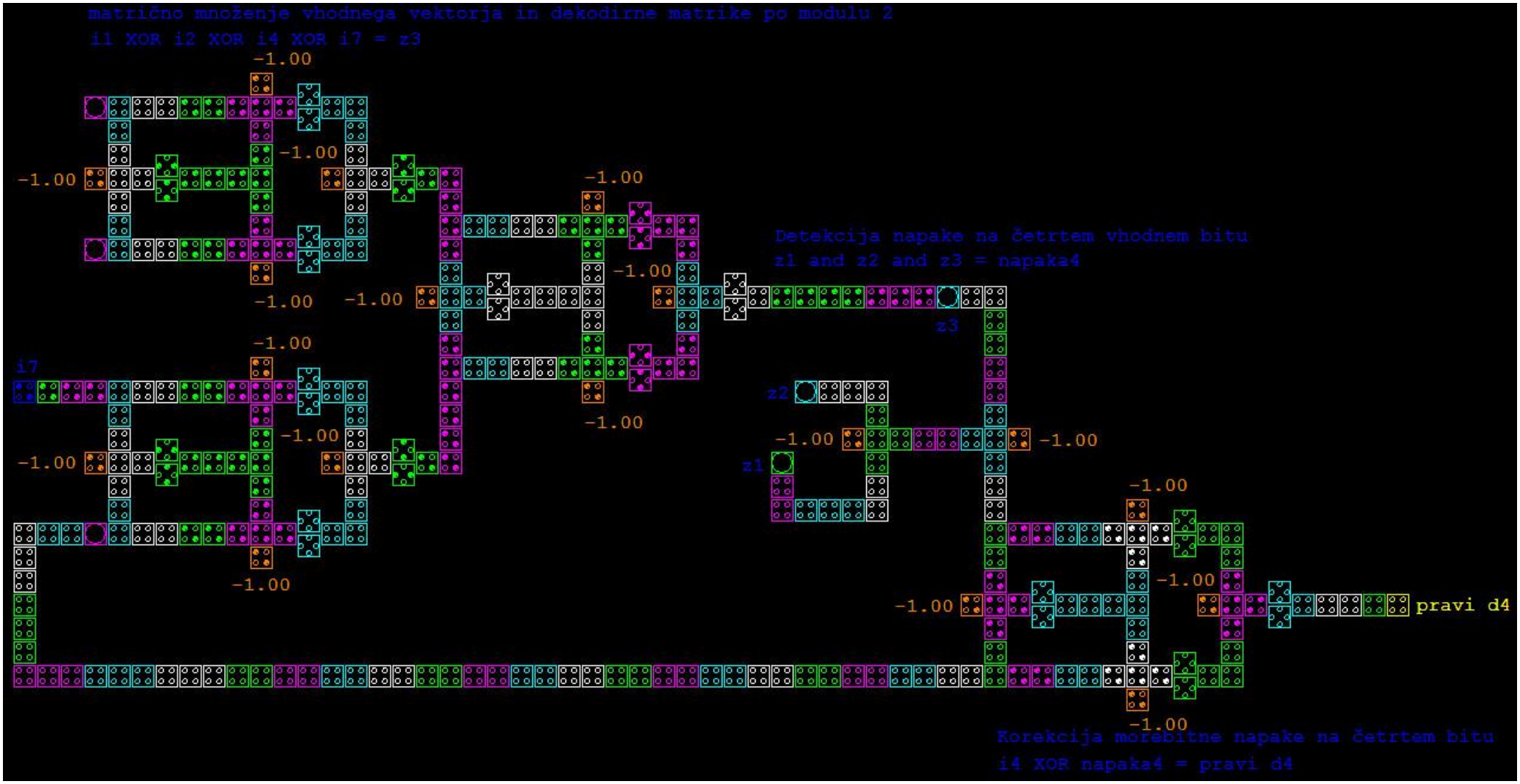
$$napaka\ na\ 3.bitu = z_1 \wedge z_2 \wedge \overline{z_3}$$

$$napaka\ na\ 4.bitu = z_1 \wedge z_2 \wedge z_3$$

Napako popravimo tako, da ustreznemu bitu obrnemo vrednost.



Slika 8: Logična shema odkrivanja in popravljanja napake na četrtem bitu



Slika 9: Odkrivanje in popravljanje napake na četrtem bitu v QCA Designerju

5 Rezultati

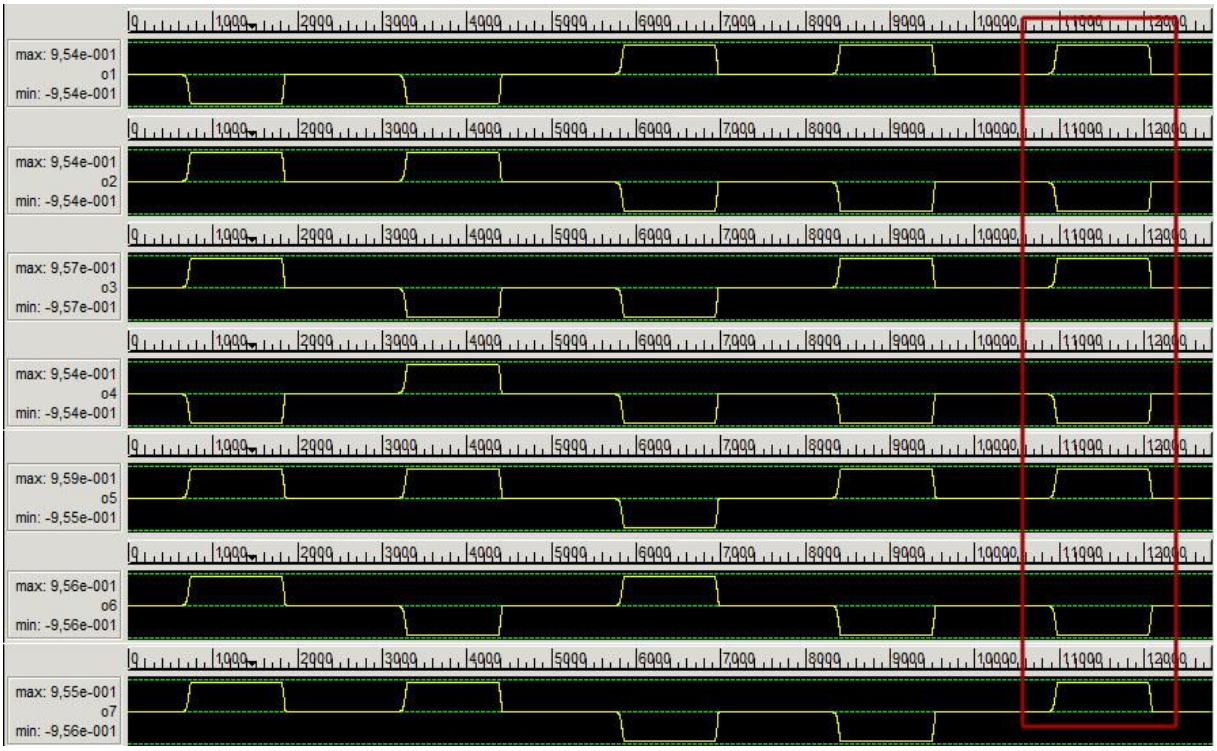
Pri oddajniku in sprejemniku se pojavijo zakasnitve. Glavni razlog je XOR operacija, ki vrne rezultat z zakasnitvijo 1,75 urine periode. Ker so XOR vrata le dvovhodna smo morali za več vhodov vrata zaporedno vezati, kar pomeni zamik za 3,5 urine periode (pri oddajniku je ta zakasnitev vidna na sliki 7, pri sprejemniku pa prvi del slike 9). Pri sprejemniku pa se pojavi še dodatna zakasnitev zaradi odkrivanja napak, kar doda zakasnitev za 1,75 urine periode, ter popravljanja napake, za kar skrbi še en XOR (slika 9). Svoj delež k zakasnitvi doda tudi večplastnost, ker morajo biti prehodne linije med plastemi na svojem clocku, da je delovanje vezja pravilno.

5.1 Rezultati oddajnika

Rezultat je zakasnjen za štiri urine periode. Komponente izhodnega vektorja so označene z o_i.

Primer 1

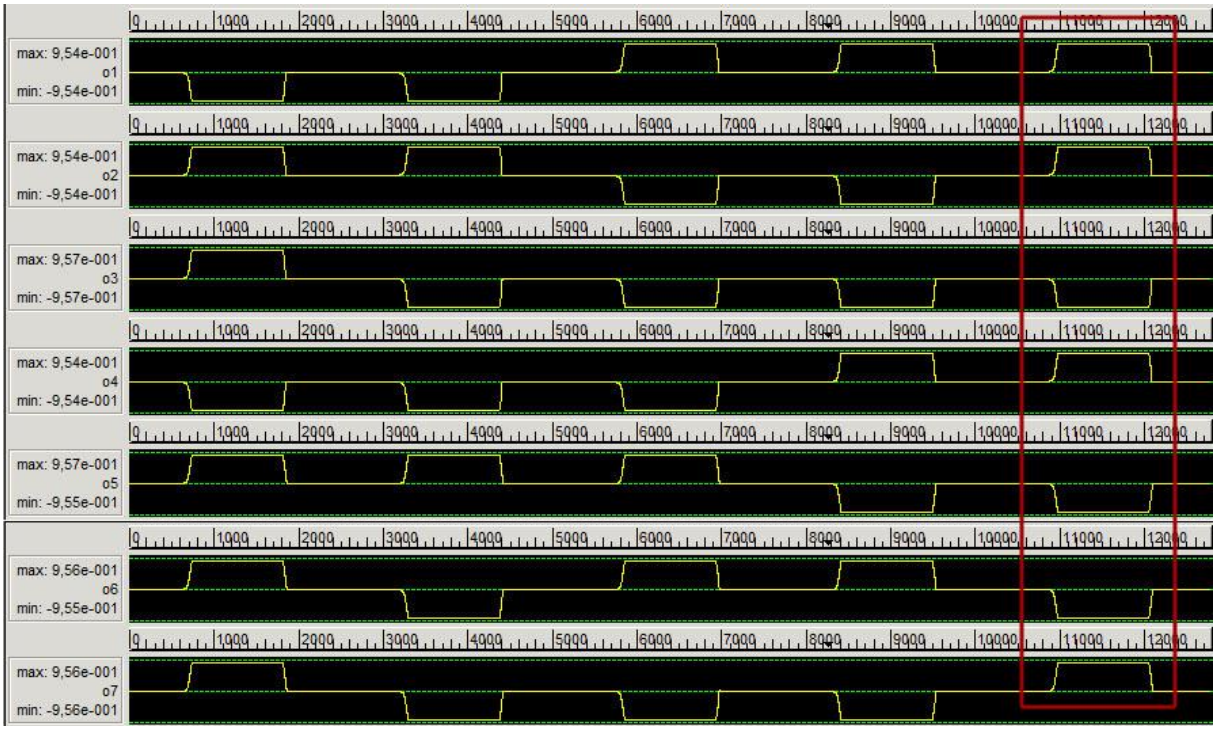
Vhodni vektor 1010. Na izhodu dobimo vektor 1010101



Slika 10: Generiranje Hammingovega koda iz vhodnega vektorja 1010

Primer 2

Vhodni vektor 1101. Na izhodu dobimo vektor 1101001



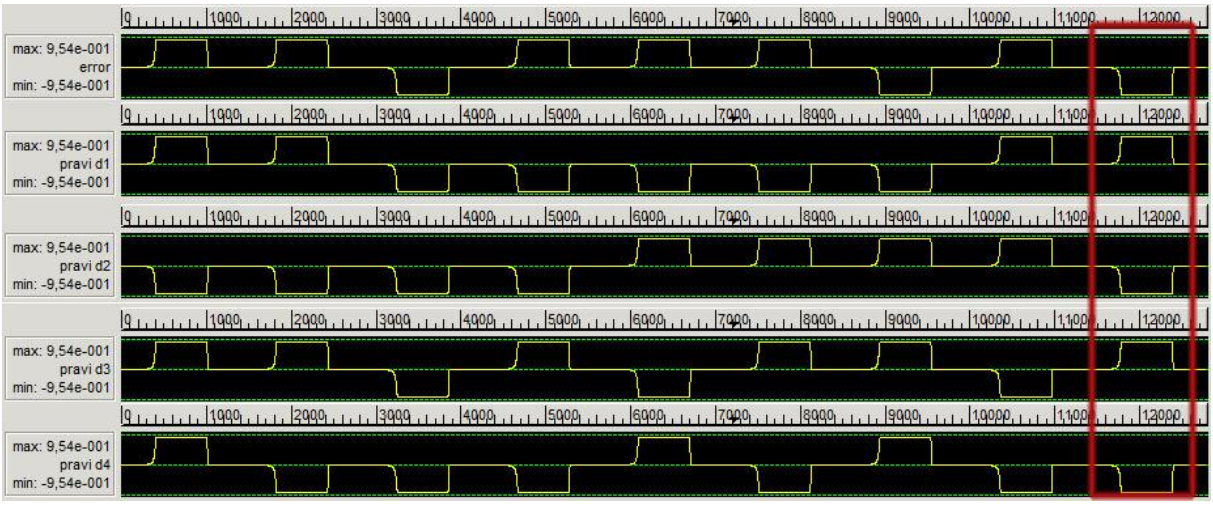
Slika 11: Generiranje Hammingovega koda iz vhodnega vektorja 1101

5.2 Rezultati sprejemnika

Rezultat je zakasnjen osem urinih period, zato so prave vrednosti v deveti urini periodi. Prva vrstica tabele prikazuje signal error, ki nam pove, če je pri prenosu prišlo do napake, ostale štiri vrstice pa so vrednosti podatkovnih signalov na izhodu.

Primer 1

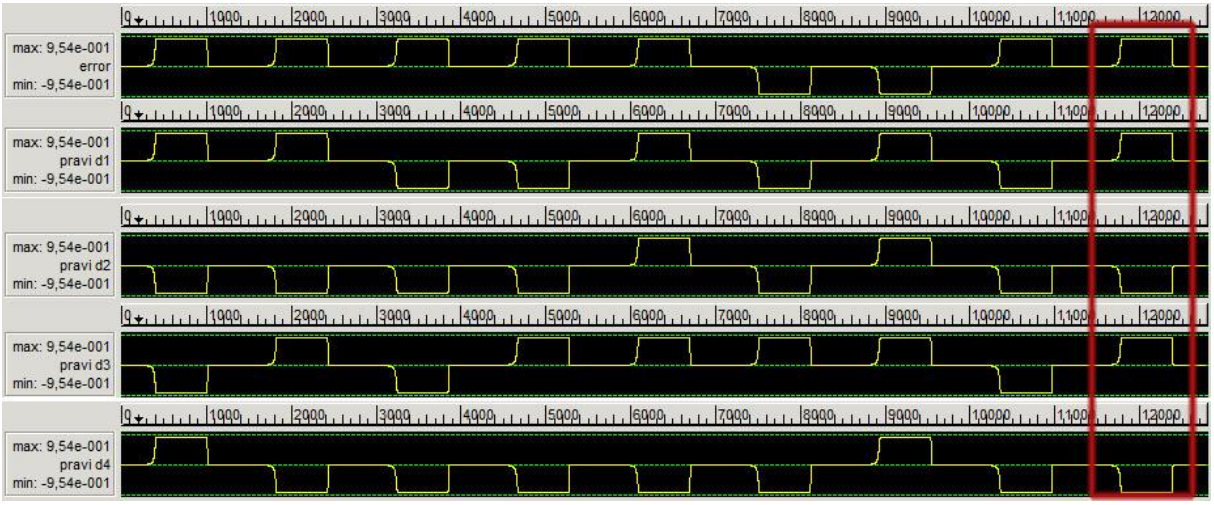
Vhodni vektor 1010101. Podatki so se prenesli pravilno. To lahko vidimo v prvi vrstici, saj je vrednost error signala negativna, to pa pomeni, da do napake ni prišlo. Na izhodu je pravilen podatek.



Slika 12: Pravilen izhod brez napake

Primer 2

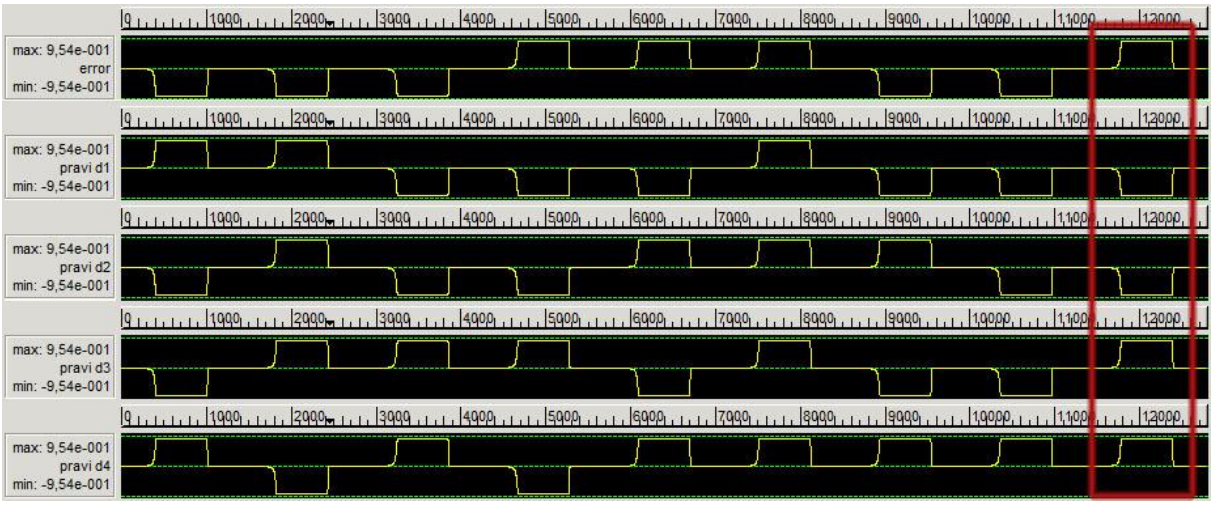
Vhodni vektor 101**1**101. Pokvarjen četrti bit. Signal error postavljen, kar pomeni, da je prišlo do napake pri prenosu. Ker pa je bila napaka le enobitna je izhod pravilen.



Slika 13: Pravilen izhod pri enobitni napaki

Primer 3

Podali smo vektor **0**011**1**101, kar pomeni, da smo prejšnjemu primeru dodali še eno napako. Napaka je bila zaznana, to se vidi iz error signala, vendar so se podatki prenesli napačno, saj logika popravlja le enobitne napake, dvobitne pa le detektira.



Slika 14: Napačen izhod pri dvobitni napaki

6 Zaključek

Pri realizaciji seminarske naloge smo naleteli na kar nekaj težav. Najprej smo imeli težavo z XOR vrati, saj smo s prvimi rešitvami dobivali napačne in zelo nestabilne rezultate pri zaporedni vezavi. To smo rešili z realizacijo XOR vrat s štirimi NAND vrati, dodatno pa še z realizacijo logike v več plasteh (layer-jih). Tako je postala celotna struktura bolj stabilna in se obnaša po pričakovanjih. Hkrati pa smo si z večplastnostjo olajšali delo pri križanju linij, saj je enoplastna rešitev prav tako zelo nestabilna. Večplastnost smo nadalje izkoristili tudi zato, da smo tvorbo vsakega paritetnega bita izvršili na svoji plasti. S tem je sam design tudi preglednejši in bolj modularen.

Veliko časa smo izgubili tudi s samim programom QCA designer, saj se je velikokrat sesuval oziroma je prihajalo do nepričakovanih rezultatov.

Sam design bi lahko še izboljšali s tem, da bi zreducirali število časovnih zamikov v logiki. Druga možnost optimizacije je tudi zmanjšanje števila celic, ki smo jo uporabili. Oddajni del je sestavljen iz 520 celic in ima zamik štirih urinih period. Sprejemni del pa je kompleksnejši in je sestavljen iz 1170 celic. Izhod je glede na vhod zamaknjen za 8 urinih period.

7 Viri in literature

i. http://www.t-kougei.ac.jp/research/pdf/vol31-1-03.pdf

ii. http://michael.dipperstein.com/hamming/index.html

iii. http://zone.ni.com/devzone/cda/tut/p/id/6480

iv. http://tams-www.informatik.uni-hamburg.de/applets/hades/webdemos/10-gates/50-hamming/hamming.html