Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Optične tehnologije in nanotehnologije
Seminarska naloga

# Realizacija Hammingovega (7, 4) kodiranja s kvantnimi celičnimi avtomati

Avtorji:
Samo Kralj, Maja Somrak, Dušan Vučko, Jan Hanzel

Ljubljana, november 2010

# Kazalo

# 1. Uvod

Zagotavljanje pravilnosti prenešenih informacij je v informacijskih kanalih izredno pomembno, saj le z detekcijo napak lahko zagotovimo, da bo prejemnik res prejel tisto, kar smo mu poslali. V najosnovnejši obliki se prenešenim informacijam dodajajo paritetni biti.

Paritetni bit je bit, ki se doda zaporedju bitov (bloku) z namenom ugotavljanja napake v njem pri npr. prenosu prek kanala. V primeru sode paritete vsak blok bitov skupaj s paritetnim bitom vsebuje sodo število enic. Če blok vsebuje liho število enic, bo paritetni bit nastavljen na 1, če sodo število, pa na 0. Tako bo skupno število enic vedno sodo. Če nato kateri od bitov zavzame napačno vrednost, bo iz števila enic razvidno, da je prišlo do napake.

Tak pristop je neustrezen v primeru, ko je napačen več kot en bit. Poleg tega lahko napako zgolj detektira, ne more pa je odpraviti. To pomankljivost odpravlja Hammingov kod.

Hammingov kod je linearen bločni kod, ki se uporablja pri komunikacijah preko izgubnih kanalov. Pogost je Hammingov kod (7, 4). Njegova posebnost je v večjem številu paritetnih bitov, s pomočjo katerih lahko pri dekodiranju popravimo eno in zaznamo do dve napaki.

Hammingov kod (7, 4) je sestavljen iz štirih podatkovnih ($x_1$-$x_4$) ter treh paritetnih ($p_1$-$p_3$) bitov, ki jih določajo enačbe:

- $p_1 = (x_1 + x_2 + x_4) \bmod 2$
- $p_2 = (x_1 + x_3 + x_4) \bmod 2$
- $p_3 = (x_2 + x_3 + x_4) \bmod 2$

| Pozicija bita | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Bit | **$p_1$** | **$p_2$** | $x_1$ | **$p_3$** | $x_2$ | $x_3$ | $x_4$ |

Paritetni biti se nahajajo na pozicijah bitov, ki so potence števila 2. Ostale pozicije bitov so uporabljene za podatkovne bite, ki jih kodiramo.

Vsak paritetni bit računa sodo pariteto določenih podatkovnih bitov. Katere bite bo posamezni paritetni bit preverjal (pokrival), je določeno z njegovo pozicijo. Paritetni bit na poziciji *n* najprej preskoči *n*-1 bitov, jih nato pokrije *n*, *n* preskoči, spet pokrije *n* itn. Paritetni biti pri Hammingovem (7, 4) kodu pokrivajo naslednje podatkovne bite:

- $p_1$ - $x_1$, $x_2$, $x_4$
- $p_2$ - $x_1$, $x_3$, $x_4$
- $p_3$ - $x_2$, $x_3$, $x_4$

Za primer navedimo kodiranje bitov 1011 – rezultat bo oblike:

| Pozicija bita | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Bit | **$p_1$** | **$p_2$** | 1 | **$p_3$** | 0 | 1 | 1 |

Pri določanju posameznega paritetnega bita upoštevamo, katere podatkovne bite le-ta pokriva:

- $p_1 = 0$, ker biti 3, 5, 7 (1, 0, 1) vsebujejo sodo število enic (vsota po modulu 2 je 0),
- $p_2 = 1$, ker biti 3, 6, 7 (1, 1, 1) vsebujejo liho število enic (vsota po modulu 2 je 1),
- $p_3 = 0$, ker biti 5, 6, 7 (0, 1, 1) vsebujejo sodo število enic (vsota po modulu 2 je 0),

Rezultat kodiranja bo torej sedmerica bitov 0110011.

Matematično predstavljeno podatkovne bite (vektor x) kodiramo z množenjem z generatorsko matriko G:

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$y^T = G \cdot x^T$$

Dekodiranje prejetega vektorja y poteka z matriko H:

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$E^T = H \cdot y^T$$

Velja:

- $E = 0$ - prenos brez napak,
- $E \neq 0$ - napaka v prejetem vektorju y.

V nalogi je opisana izvedba Hammingovega (7, 4) koda s kvantnimi celičnimi avtomati (QCA), vključno s korekcijo 1-bitne napake.

# 2. Delovanje in logične sheme

## 2.1. Kodirnik

Hamming(7,4) kodirnik kot vhod prejme 4 podatkovne bite $x_1$, $x_2$, $x_3$, $x_4$, kot izhod pa vrne v 7 bitov zakodirane omenjene 4 bite, razširjene s tremi paritetnimi biti $p_1$, $p_2$, $p_3$ (Slika 1).

$$[x_1, x_2, x_3, x_4] \xrightarrow[kodirnik]{} [p_1, p_2, x_1, p_3, x_2, x_3, x_4] = [r_1, r_2, r_3, r_4, r_5, r_6, r_7]$$

*Slika 1: delovanje kodirnika*

Pri tem velja:

- $r_1 = p_1 = (x_1 + x_2 + x_4) \bmod 2$
- $r_2 = p_2 = (x_1 + x_3 + x_4) \bmod 2$
- $r_4 = p_3 = (x_2 + x_3 + x_4) \bmod 2$

Z drugimi besedami:

- $r_1 = p_1 = 1$ natanko takrat, ko biti $x_1$, $x_2$, $x_4$ vsebujejo liho število enic,
- $r_2 = p_2 = 1$ natanko takrat, ko biti $x_1$, $x_3$, $x_4$ vsebujejo liho število enic,
- $r_4 = p_3 = 1$ natanko takrat, ko biti $x_2$, $x_3$, $x_4$ vsebujejo liho število enic.

V primeru sodega števila enic v naboru posamezne trojice bitov je paritetni bit enak 0.

Velja tudi:

- $r_3 = x_1$
- $r_5 = x_2$
- $r_6 = x_3$
- $r_7 = x_4$

Na podlagi teh ugotovitev lahko določimo pravilnostne tabele za določitev posameznega paritetnega bita (izhodni biti $r_1$, $r_2$, $r_4$), na podlagi tabel pa logično shemo kodirnika.

| $x_1$ | $x_2$ | $x_4$ | $r_1$ |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **1** |
| 0 | 1 | 0 | **1** |
| 0 | 1 | 1 | **0** |
| 1 | 0 | 0 | **1** |
| 1 | 0 | 1 | **0** |
| 1 | 1 | 0 | **0** |
| 1 | 1 | 1 | **1** |

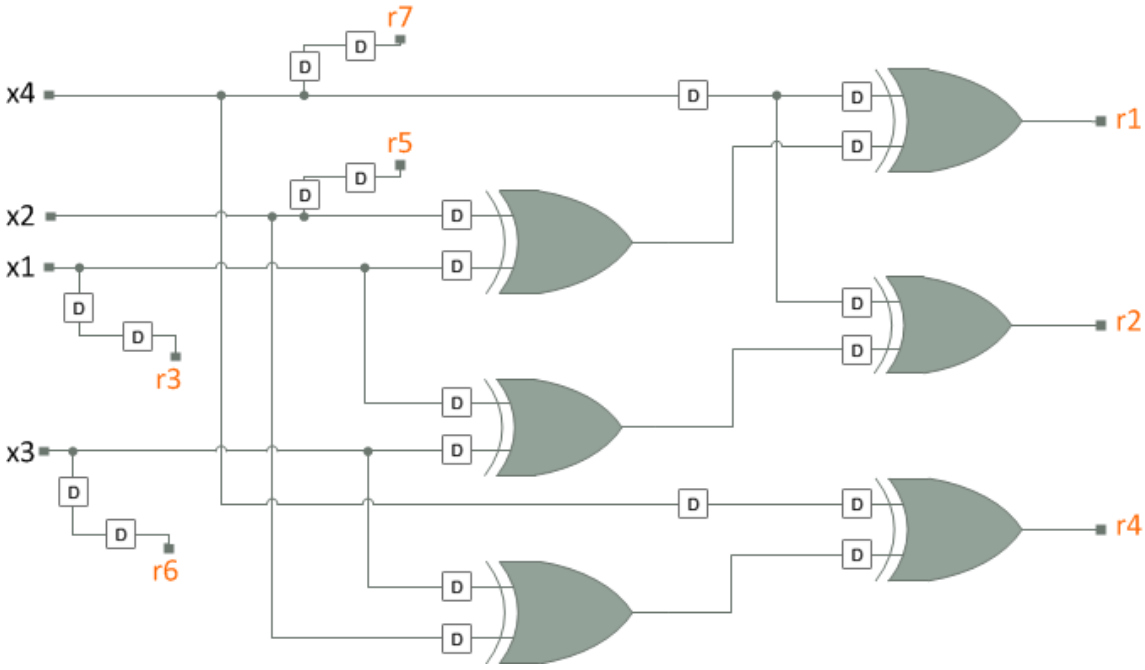
| $x_1$ | $x_3$ | $x_4$ | $r_2$ |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **1** |
| 0 | 1 | 0 | **1** |
| 0 | 1 | 1 | **0** |
| 1 | 0 | 0 | **1** |
| 1 | 0 | 1 | **0** |
| 1 | 1 | 0 | **0** |
| 1 | 1 | 1 | **1** |

| $x_2$ | $x_3$ | $x_4$ | $r_4$ |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **1** |
| 0 | 1 | 0 | **1** |
| 0 | 1 | 1 | **0** |
| 1 | 0 | 0 | **1** |
| 1 | 0 | 1 | **0** |
| 1 | 1 | 0 | **0** |
| 1 | 1 | 1 | **1** |

Velja:

- $r_1 = (x_1 \text{ xor } x_2) \text{ xor } x_4$
- $r_2 = (x_1 \text{ xor } x_3) \text{ xor } x_4$
- $r_4 = (x_2 \text{ xor } x_3) \text{ xor } x_4$

Iz zgornjih izrazov sledi logična shema kodirnika (Slika 2). Vsak D pomeni zamik signala na dani liniji za eno urino periodo (4 faze) pri QCA realizaciji.




*Slika 2: logična shema kodirnika*

## 2.2. Dekodirnik – detekcija napake

Prvi del dekodirnika služi detekciji napake – na podlagi vhodnih 7 bitov $r_1$ - $r_7$, ki predstavljajo kodiran blok podatkov, ugotovi, ali je eden izmed sprejetih bitov napačen in kateri bit to je. Če je bit napačen, ga je potrebno popraviti – invertirati.

Pri zasnovi logike za detekcijo napake izhajamo iz vednosti, da mora pri pravilno zakodirani četverici bitov posamezen paritetni bit skupaj s podatkovnimi biti, ki jih pokriva, vsebovati sodo število enic. To velja, ko je vsota po modulu 2 paritetnega bita in podatkovnih bitov, ki jih ta pokriva, enaka nič:

- $(\mathbf{r_1} + r_3 + r_5 + r_7) \bmod 2 = 0$,
- $(\mathbf{r_2} + r_3 + r_6 + r_7) \bmod 2 = 0$,
- $(\mathbf{r_4} + r_5 + r_6 + r_7) \bmod 2 = 0$.

Pri tem je

- $r_1 = p_1$ (pokriva bite $r_3$, $r_5$, $r_7$);
- $r_2 = p_2$ (pokriva bite $r_3$, $r_6$, $r_7$);
- $r_4 = p_3$ (pokriva bite $r_5$, $r_6$, $r_7$).

Naj bo:

- $(\mathbf{r_1} + r_3 + r_5 + r_7) \bmod 2 = e_1$,
- $(\mathbf{r_2} + r_3 + r_6 + r_7) \bmod 2 = e_2$,
- $(\mathbf{r_4} + r_5 + r_6 + r_7) \bmod 2 = e_3$.

Kot izhod detekcije napake želimo dobiti vektor napake $E = [e_1, e_2, e_3]$, ki enolično določa, ali vhodna zakodirana beseda vsebuje napačen bit in kateri bit to je.

Napake ni – sprejeta beseda je pravilna – če velja E = [0, 0, 0]. V nasprotnem primeru desetiška vrednost vektorja E določa, kateri bit je potrebno popraviti, pri čemer so uteži za $e_1$ 1, za $e_2$ 2 in za $e_3$ 4. Npr. E = [1, 0, 0] pomeni, da je napačen bit $r_1$, E = [1, 1, 1] pa, da je napačen bit $r_7$ (ker je 1 + 2 + 4 = 7).

Velja torej:

- $e_1$ = 1 natanko takrat, ko biti $r_1$, $r_3$, $r_5$, $r_7$ vsebujejo liho število enic,
- $e_2$ = 1 natanko takrat, ko biti $r_2$, $r_3$, $r_6$, $r_7$ vsebujejo liho število enic,
- $e_3$ = 1 natanko takrat, ko biti $r_4$, $r_5$, $r_6$, $r_7$ vsebujejo liho število enic.

Zapišemo pravilnostne tabele za E:

| $r_1$ | $r_3$ | $r_5$ | $r_7$ | $e_1$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **0** |
| 0 | 0 | 0 | 1 | **1** |
| 0 | 0 | 1 | 0 | **1** |
| 0 | 0 | 1 | 1 | **0** |
| 0 | 1 | 0 | 0 | **1** |
| 0 | 1 | 0 | 1 | **0** |
| 0 | 1 | 1 | 0 | **0** |
| 0 | 1 | 1 | 1 | **1** |
| 1 | 0 | 0 | 0 | **1** |
| 1 | 0 | 0 | 1 | **0** |
| 1 | 0 | 1 | 0 | **0** |
| 1 | 0 | 1 | 1 | **1** |
| 1 | 1 | 0 | 0 | **0** |
| 1 | 1 | 0 | 1 | **1** |
| 1 | 1 | 1 | 0 | **1** |
| 1 | 1 | 1 | 1 | **0** |

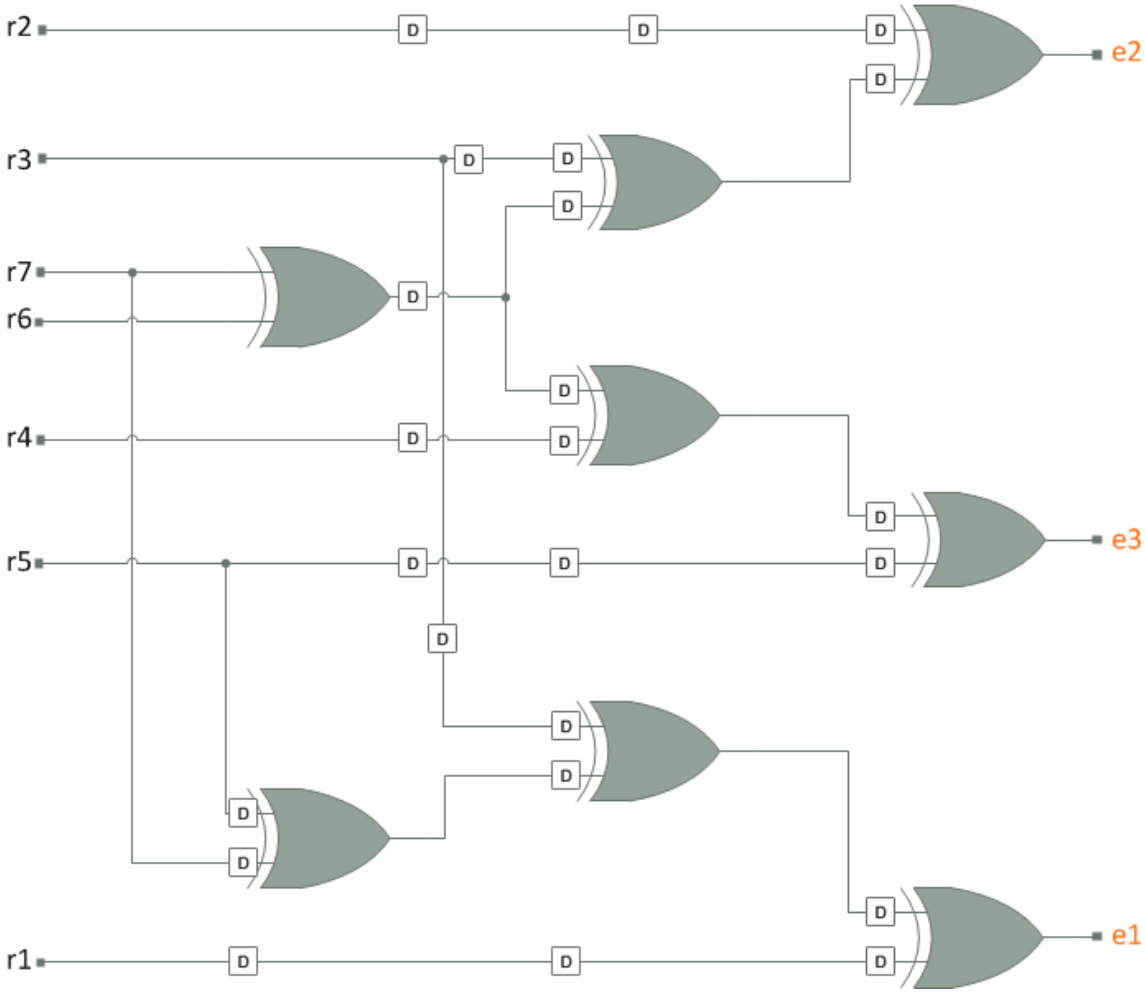
| $r_2$ | $r_3$ | $r_6$ | $r_7$ | $e_2$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **0** |
| 0 | 0 | 0 | 1 | **1** |
| 0 | 0 | 1 | 0 | **1** |
| 0 | 0 | 1 | 1 | **0** |
| 0 | 1 | 0 | 0 | **1** |
| 0 | 1 | 0 | 1 | **0** |
| 0 | 1 | 1 | 0 | **0** |
| 0 | 1 | 1 | 1 | **1** |
| 1 | 0 | 0 | 0 | **1** |
| 1 | 0 | 0 | 1 | **0** |
| 1 | 0 | 1 | 0 | **0** |
| 1 | 0 | 1 | 1 | **1** |
| 1 | 1 | 0 | 0 | **0** |
| 1 | 1 | 0 | 1 | **1** |
| 1 | 1 | 1 | 0 | **1** |
| 1 | 1 | 1 | 1 | **0** |

| $r_4$ | $r_5$ | $r_6$ | $r_7$ | $e_3$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **0** |
| 0 | 0 | 0 | 1 | **1** |
| 0 | 0 | 1 | 0 | **1** |
| 0 | 0 | 1 | 1 | **0** |
| 0 | 1 | 0 | 0 | **1** |
| 0 | 1 | 0 | 1 | **0** |
| 0 | 1 | 1 | 0 | **0** |
| 0 | 1 | 1 | 1 | **1** |
| 1 | 0 | 0 | 0 | **1** |
| 1 | 0 | 0 | 1 | **0** |
| 1 | 0 | 1 | 0 | **0** |
| 1 | 0 | 1 | 1 | **1** |
| 1 | 1 | 0 | 0 | **0** |
| 1 | 1 | 0 | 1 | **1** |
| 1 | 1 | 1 | 0 | **1** |
| 1 | 1 | 1 | 1 | **0** |

Velja:

- $e_1 = ((r_5 \text{ xor } r_7) \text{ xor } r_3) \text{ xor } r_1$,
- $e_2 = ((r_6 \text{ xor } r_7) \text{ xor } r_3) \text{ xor } r_2$,
- $e_3 = ((r_6 \text{ xor } r_7) \text{ xor } r_4) \text{ xor } r_5$.

Iz zgornjih izrazov sledi logična shema za detekcijo napak (Slika 3). Vsak D pomeni zamik signala na dani liniji za eno urino periodo (4 faze) pri QCA realizaciji.




*Slika 3: logična shema za detekcijo napak*

## 2.3. Dekodirnik – popravljanje napake

Drugi del dekodirnika služi popravljanju 1-bitne napake na osnovi vektorja $E = [e_1, e_2, e_3]$; napako po potrebi popravi in na izhod vrne podatkovne bite $x_1$, $x_2$, $x_3$, $x_4$. Če napake ni, zgolj pripelje vrednost posameznega podatkovnega bita na izhod. Popravljanje bita pomeni invertiranje njegove vrednosti.

Ker nas zanimajo zgolj podatkovni biti, ki so zakodirani v bitih $r_3$, $r_5$, $r_6$, $r_7$, paritetnih bitov ($r_1$, $r_2$, $r_4$) ne popravljamo, zato tudi ne nastopajo kot vhod v dekodirnik. Rezultat te odločitve je manjše in bolj optimalno vezje.

$x_1$ (zakodiran v $r_3$) popravimo, ko je $E = [1, 1, 0]$; v vseh ostalih primerih je $x_1 = r_3$.
$x_2$ (zakodiran v $r_5$) popravimo, ko je $E = [1, 0, 1]$; v vseh ostalih primerih je $x_2 = r_5$.
$x_3$ (zakodiran v $r_6$) popravimo, ko je $E = [0, 1, 1]$; v vseh ostalih primerih je $x_3 = r_6$.
$x_4$ (zakodiran v $r_7$) popravimo, ko je $E = [1, 1, 1]$; v vseh ostalih primerih je $x_4 = r_7$.

Pravilnostne tabele:

| $e_1$ | $e_2$ | $e_3$ | $r_3$ | $x_1$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **0** |
| 0 | 0 | 0 | 1 | **1** |
| 0 | 0 | 1 | 0 | **0** |
| 0 | 0 | 1 | 1 | **1** |
| 0 | 1 | 0 | 0 | **0** |
| 0 | 1 | 0 | 1 | **1** |
| 0 | 1 | 1 | 0 | **0** |
| 0 | 1 | 1 | 1 | **1** |
| 1 | 0 | 0 | 0 | **0** |
| 1 | 0 | 0 | 1 | **1** |
| 1 | 0 | 1 | 0 | **0** |
| 1 | 0 | 1 | 1 | **1** |
| 1 | 1 | 0 | 0 | **1** |
| 1 | 1 | 0 | 1 | **0** |
| 1 | 1 | 1 | 0 | **0** |
| 1 | 1 | 1 | 1 | **1** |

| $e_1$ | $e_2$ | $e_3$ | $r_6$ | $x_3$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **0** |
| 0 | 0 | 0 | 1 | **1** |
| 0 | 0 | 1 | 0 | **0** |
| 0 | 0 | 1 | 1 | **1** |
| 0 | 1 | 0 | 0 | **0** |
| 0 | 1 | 0 | 1 | **1** |
| 0 | 1 | 1 | 0 | **1** |
| 0 | 1 | 1 | 1 | **0** |
| 1 | 0 | 0 | 0 | **0** |
| 1 | 0 | 0 | 1 | **1** |
| 1 | 0 | 1 | 0 | **0** |
| 1 | 0 | 1 | 1 | **1** |
| 1 | 1 | 0 | 0 | **0** |
| 1 | 1 | 0 | 1 | **1** |
| 1 | 1 | 1 | 0 | **0** |
| 1 | 1 | 1 | 1 | **1** |

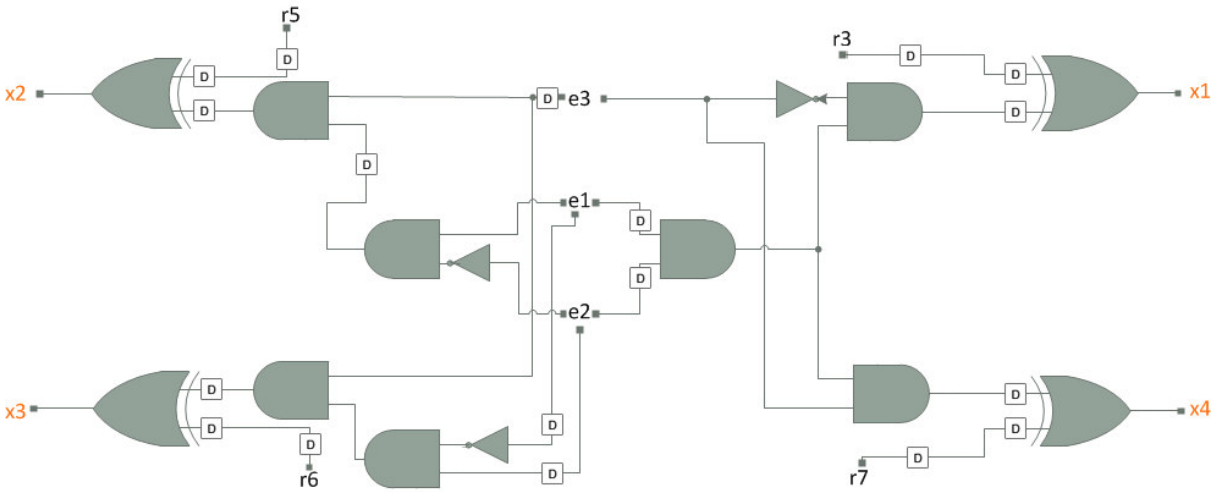
| $e_1$ | $e_2$ | $e_3$ | $r_5$ | $x_2$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **0** |
| 0 | 0 | 0 | 1 | **1** |
| 0 | 0 | 1 | 0 | **0** |
| 0 | 0 | 1 | 1 | **1** |
| 0 | 1 | 0 | 0 | **0** |
| 0 | 1 | 0 | 1 | **1** |
| 0 | 1 | 1 | 0 | **0** |
| 0 | 1 | 1 | 1 | **1** |
| 1 | 0 | 0 | 0 | **0** |
| 1 | 0 | 0 | 1 | **1** |
| 1 | 0 | 1 | 0 | **1** |
| 1 | 0 | 1 | 1 | **0** |
| 1 | 1 | 0 | 0 | **0** |
| 1 | 1 | 0 | 1 | **1** |
| 1 | 1 | 1 | 0 | **0** |
| 1 | 1 | 1 | 1 | **1** |

| $e_1$ | $e_2$ | $e_3$ | $r_7$ | $x_4$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | **0** |
| 0 | 0 | 0 | 1 | **1** |
| 0 | 0 | 1 | 0 | **0** |
| 0 | 0 | 1 | 1 | **1** |
| 0 | 1 | 0 | 0 | **0** |
| 0 | 1 | 0 | 1 | **1** |
| 0 | 1 | 1 | 0 | **0** |
| 0 | 1 | 1 | 1 | **1** |
| 1 | 0 | 0 | 0 | **0** |
| 1 | 0 | 0 | 1 | **1** |
| 1 | 0 | 1 | 0 | **0** |
| 1 | 0 | 1 | 1 | **1** |
| 1 | 1 | 0 | 0 | **0** |
| 1 | 1 | 0 | 1 | **1** |
| 1 | 1 | 1 | 0 | **1** |
| 1 | 1 | 1 | 1 | **0** |

Iz tabel sledijo končni izhodi iz dekodirnika:

- $x_1 = r_3$ xor (not $e_3$ and ($e_1$ and $e_2$))
- $x_2 = r_5$ xor ($e_3$ and ($e_1$ and not $e_2$))
- $x_3 = r_6$ xor ($e_3$ and ($e_2$ and not $e_1$))
- $x_4 = r_7$ xor ($e_3$ and ($e_1$ and $e_2$))

Iz zgornjih izrazov sledi logična shema za popravljanje napake (Slika 4). Vsak D pomeni zamik signala na dani liniji za eno urino periodo (4 faze) pri QCA realizaciji.




*Slika 4: logična shema za popravljanje napake*

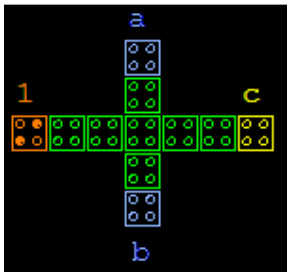
# 3. Realizacija s QCA

## 3.1. Osnovni gradniki in principi načrtovanja

Za realizacijo je bil uporabljen program QCADesigner. Pri QCA načrtovanju izhajamo iz logičnih shem iz prejšnjega poglavja. Osnovne QCA strukture, ki jih uporabljamo za izvedbo, ustrezajo naslednjim logičnim funkcijam:

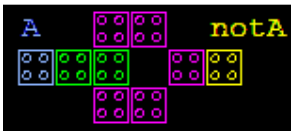
- XOR,
- AND,
- OR,
- negacija.

QCA realizacija AND in OR funkcije je dosežena z uporabo običajnih majoritetnih vrat, ki pa z namenom povečanja stabilnosti strukture vsebujejo v posamezno smer več kot zgolj eno celico. Primer takih majoritetnih vrat (OR) je na Sliki 5.




*Slika 5: OR vrata*

Za realizacijo negacije namesto dveh navpično zamaknjenih QCA linij uporabimo pristop, prikazan na Sliki 6. Taka struktura je zaradi večje simetrije bolj stabilna.

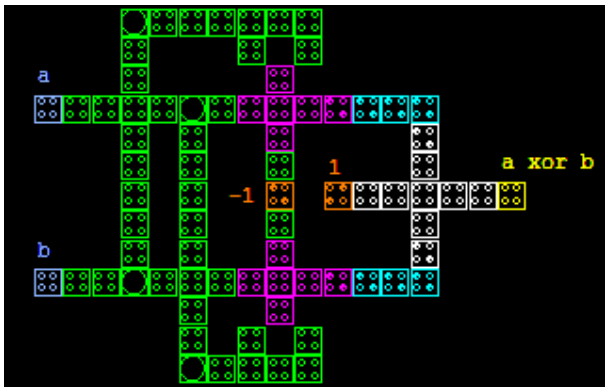



*Slika 6: negacija*

Pri realizaciji XOR za osnovo vzamemo enakost:

A xor B = (A and not B) or (not A and B)

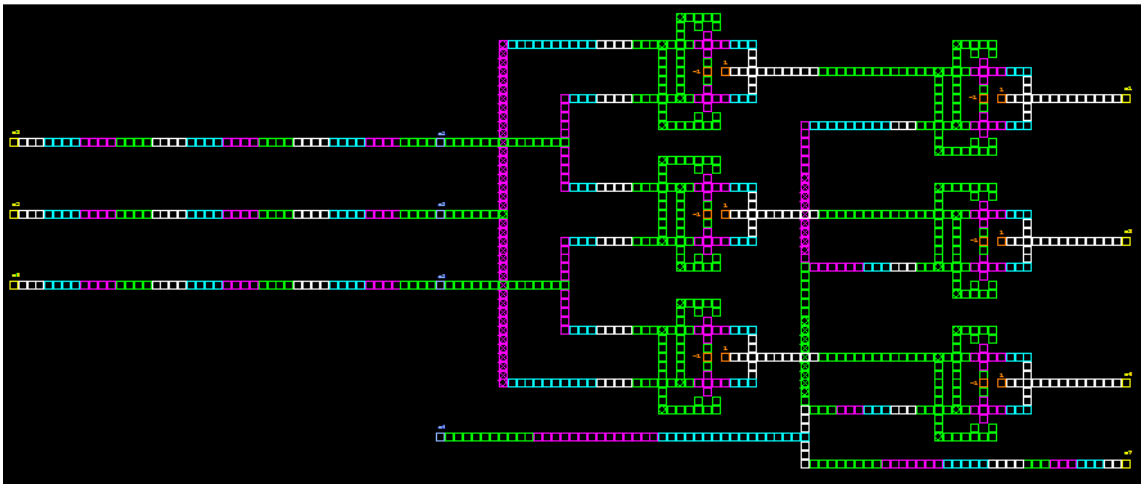
Na ta način lahko XOR realiziramo z uporabo QCA majoritetnih vrat in negacije (Slika 7).



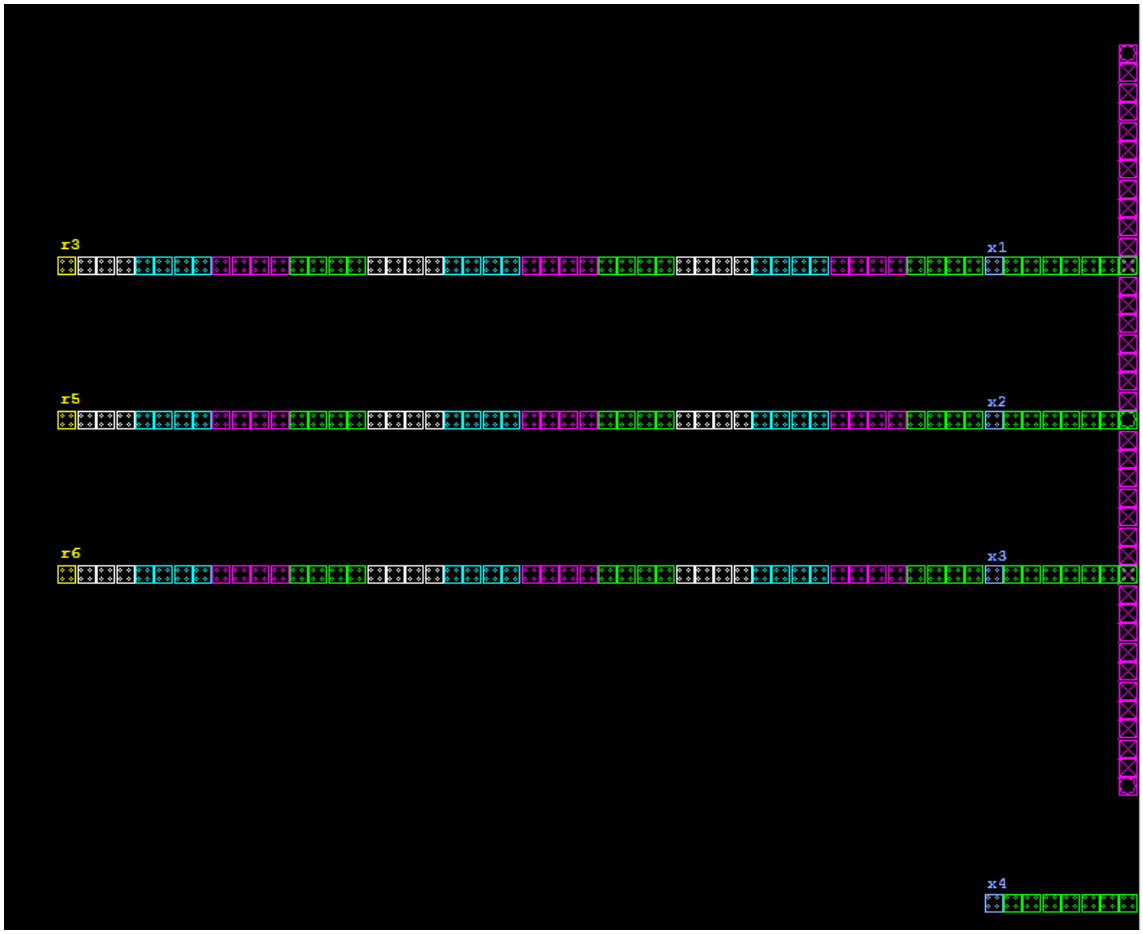

*Slika 7: QCA realizacija XOR vrat*

## 3.2. Kodirnik

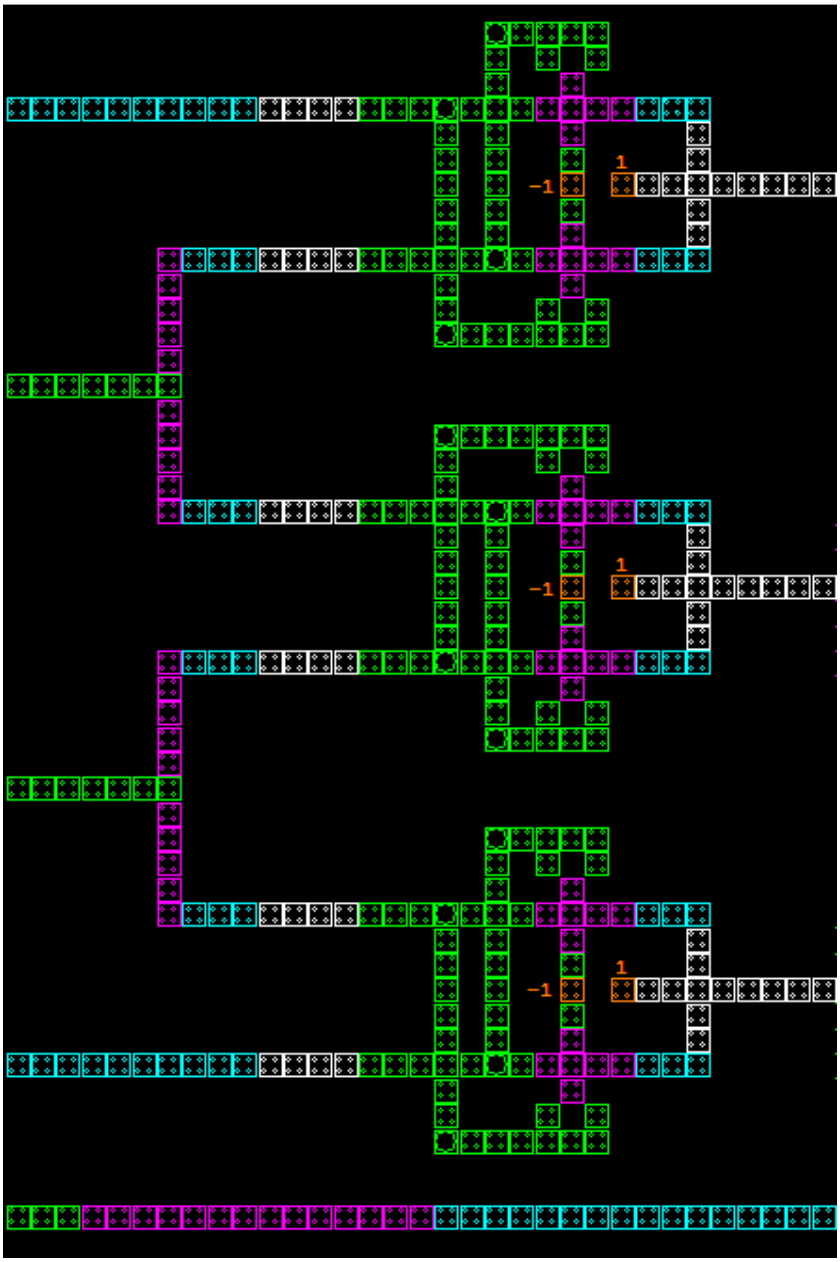
QCA realizacija kodirnika je prikazana na Sliki 8. Obsega 897 celic. Zaradi velikosti strukture je levi del sheme bolj pregledno prikazan na Sliki 9, srednji del na Sliki 10 in desni del na Sliki 11.



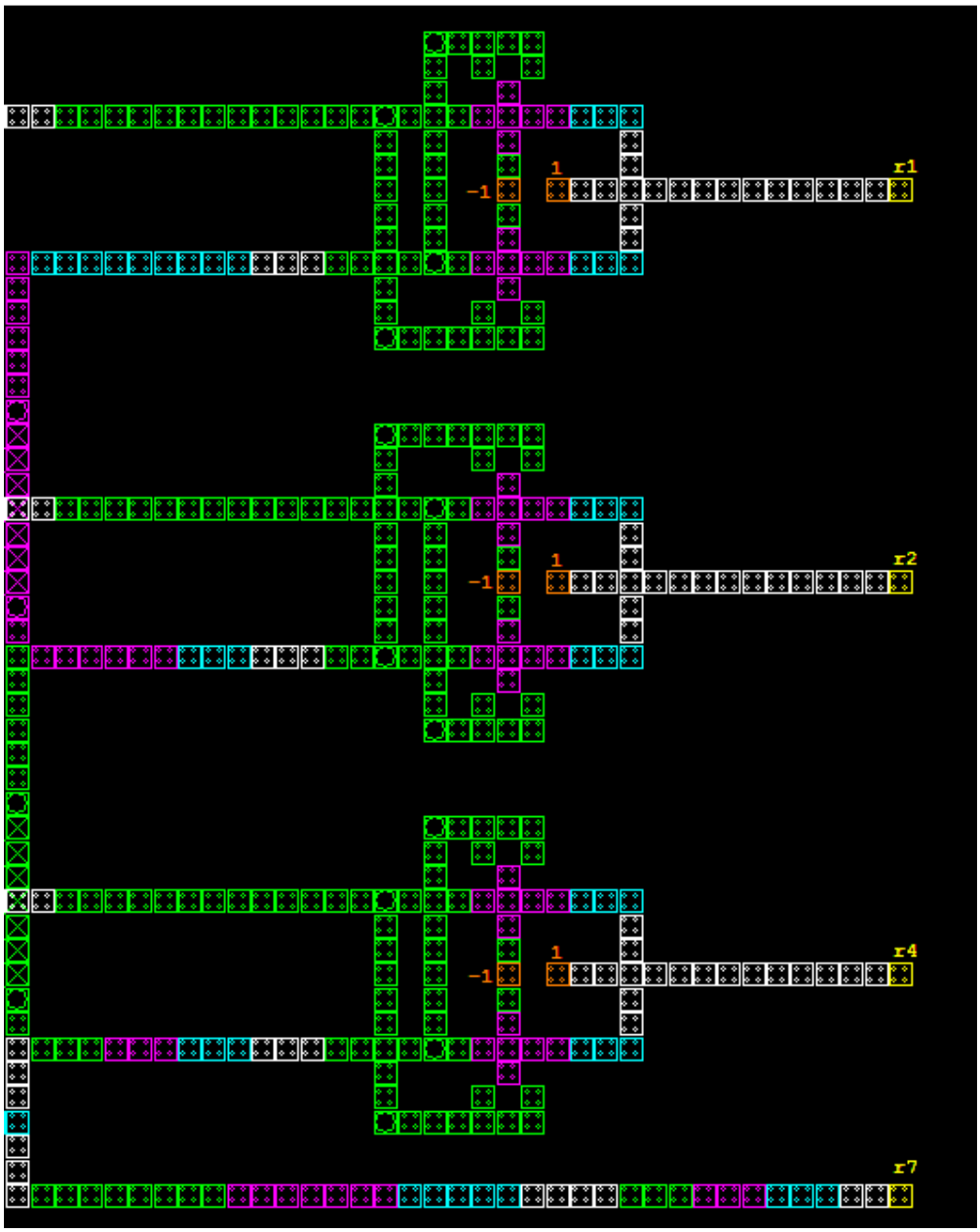
*Slika 8: QCA realizacija kodirnika – celotna struktura*




*Slika 9: QCA realizacija kodirnika – levi del strukture*



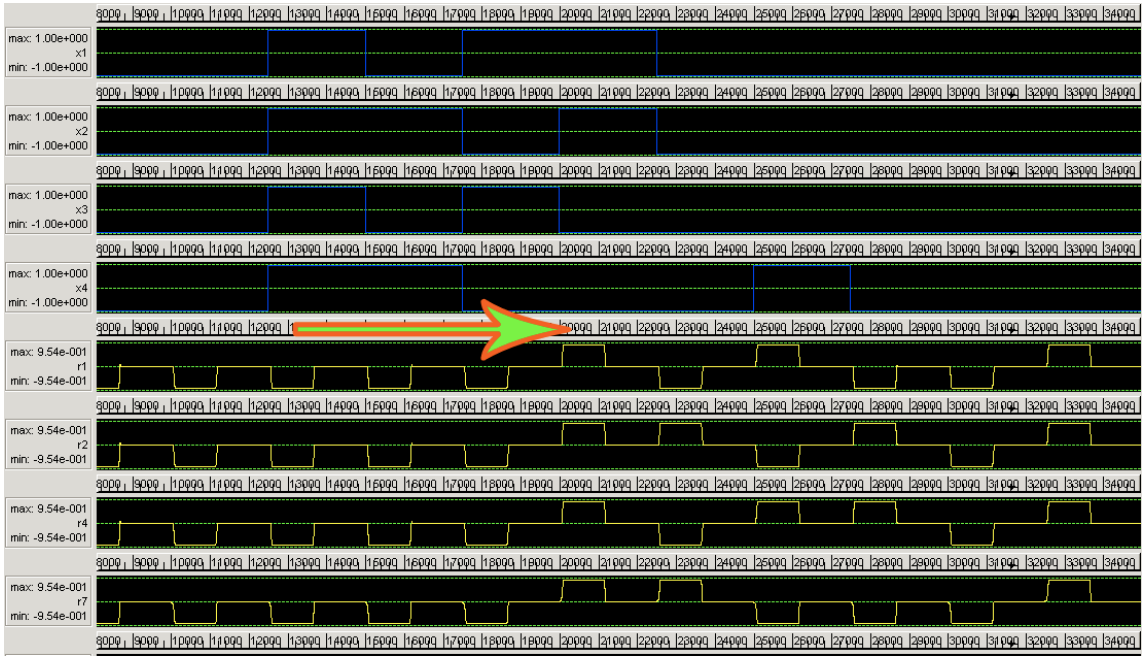

*Slika 10: QCA realizacija kodirnika – srednji del strukture*




*Slika 11: QCA realizacija kodirnika – desni del strukture*

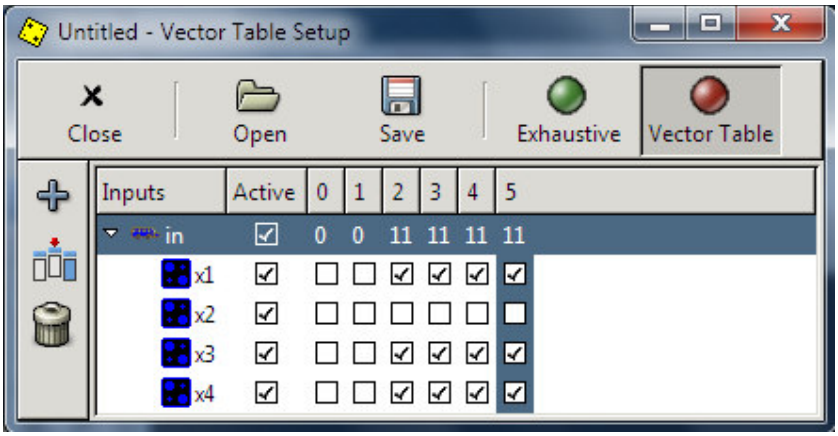
Izhodi kodirnika so na obeh straneh strukture (levo in desno od vhodov), ker se na ta način izognemo križanju linij. Posledično se poveča stabilnost vezja zaradi večje simetričnosti oz. zmanjšanja medsebojnega vpliva različnih povezav.

Rezultat simulacije je na Sliki 12. Izhodi pri podanih vhodih ustrezajo pravilnostnim tabelam iz poglavja 2.1. Zaradi preglednosti niso prikazane vse kombinacije. Izhodne vrednosti so za 3 urine periode zamaknjene glede na vhode, kar je na sliki označeno s puščico. Vhodi se na levih treh XOR vratih namreč pojavijo po 1 urini periodi, XOR pa porabi še dodatno urino periodo, da vrne izhod. Vhodi na tri XOR vrata v desnem delu zato prispejo po 2 urinih periodah, končni izhodi pa nastopijo po še eni dodatni, torej treh periodah. Ker želimo vse izhode dobiti po istem številu urinih period, za 3 periode zamaknemo še izhode v levem delu vezja ($r_3$, $r_5$, $r_6$).

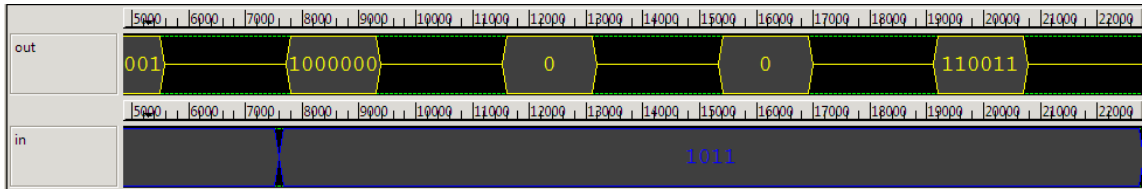



*Slika 12: rezultati simulacije kodirnika*

Primer delovanja na konkretnem primeru je kodiranje bitov 1011. Vhodni vektor prikazuje Slika 13. Slika 14 prikazuje rezultat simulacije, kjer je razvidno, da kot izhod dobimo bite 0110011 (ker je prvi bit 0, ga simulator ne izpiše), torej je bilo kodiranje uspešno oz. v skladu z enačbami iz poglavja 2.1.




*Slika 13: vhodni vektor za kodiranje 1011*

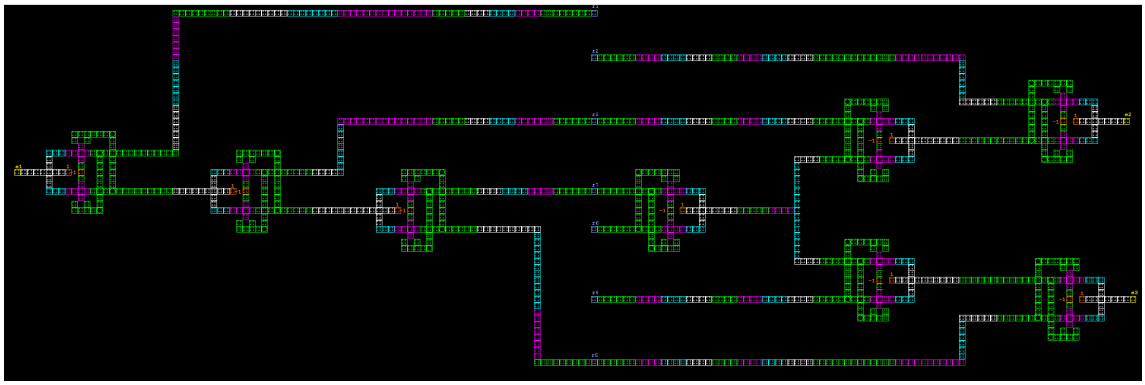



*Slika 14: rezultati simulacije za kodiranje 1011*

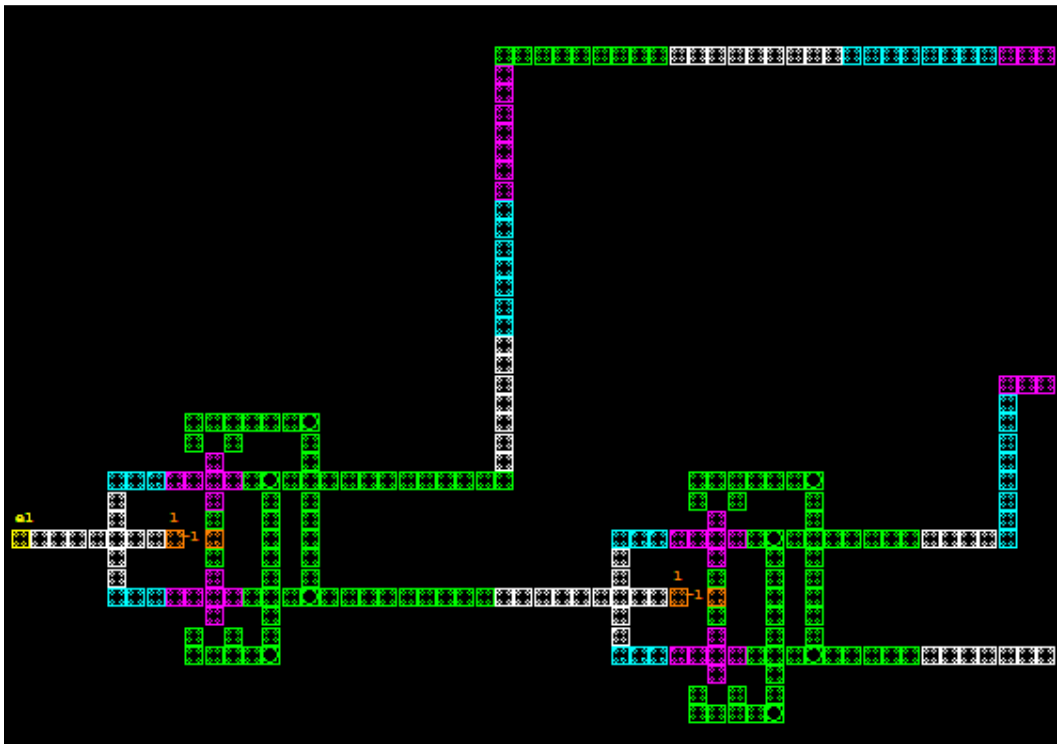
## 3.3. Dekodirnik - detekcija napake

Tudi pri QCA strukturi za detekcijo napake izhode postavimo na obe strani vezja, saj se – tako kot pri kodirniku – izognemo potrebi po križanju linij.

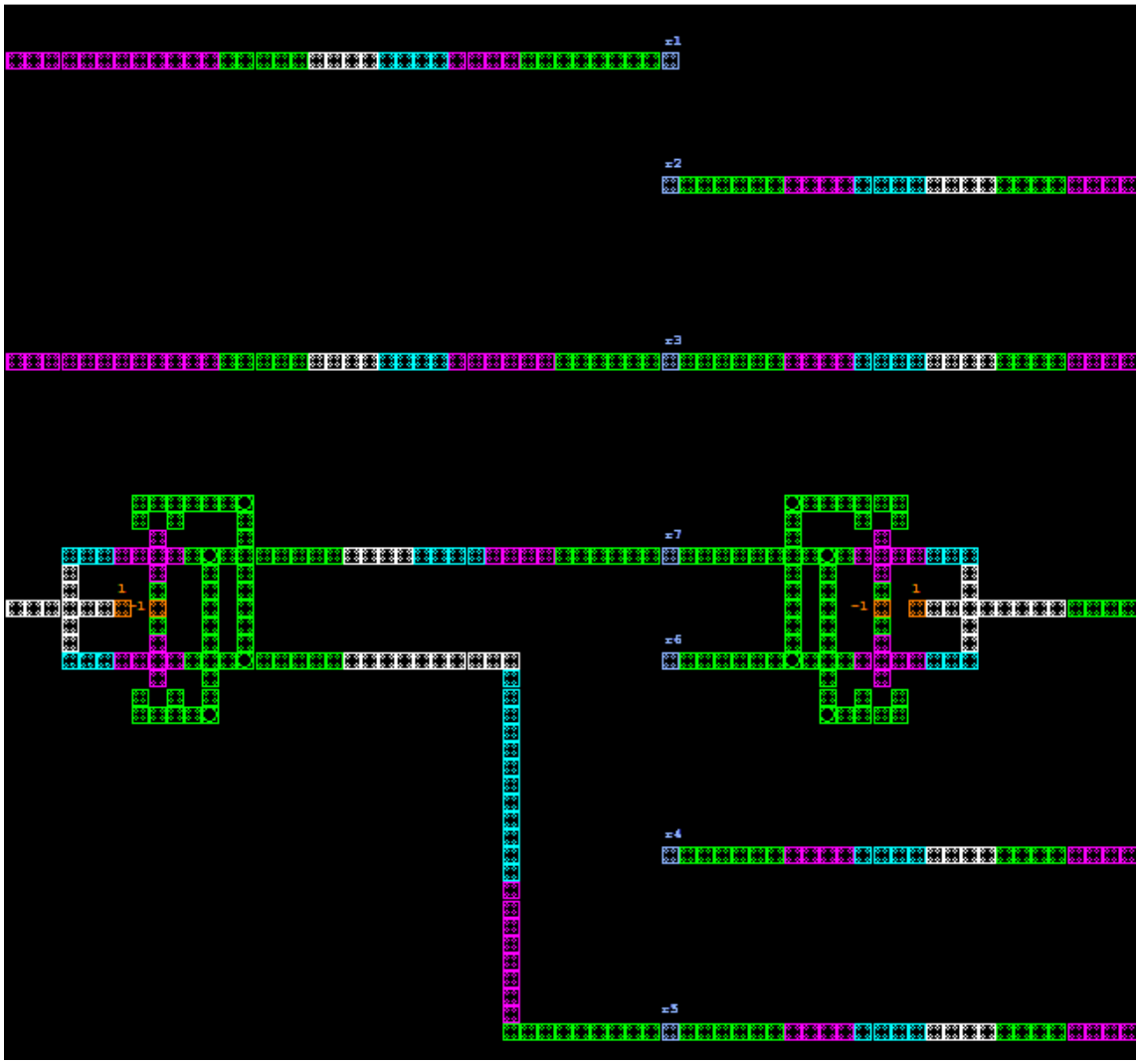
QCA realizacija detekcije napake je prikazana na Sliki 15. Obsega 1212 celic. Zaradi velikosti strukture je levi del sheme bolj pregledno prikazan na Sliki 16, srednji del na Sliki 17 in desni del na Sliki 18.



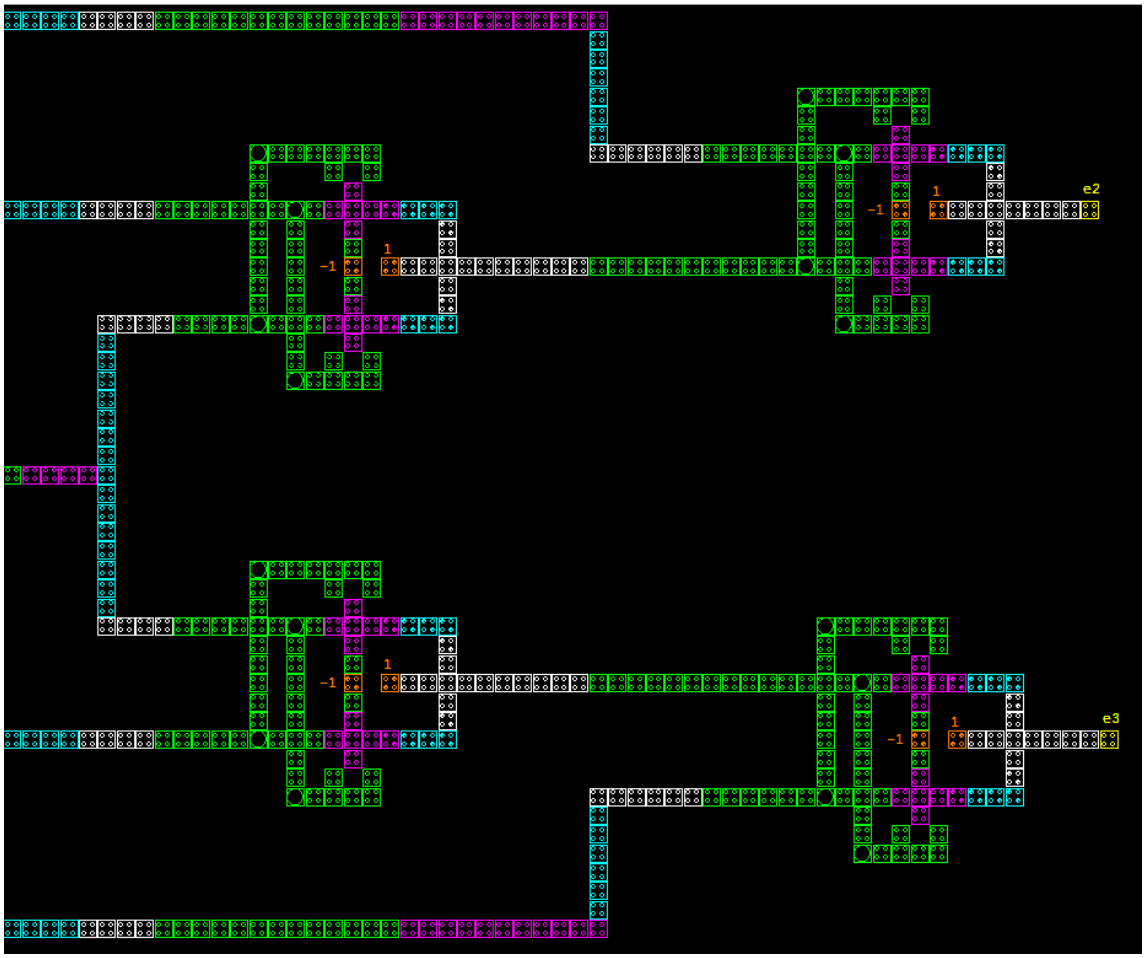
*Slika 15: QCA realizacija detekcije napake – celotna struktura*




*Slika 16: QCA realizacija detekcije napake – levi del*



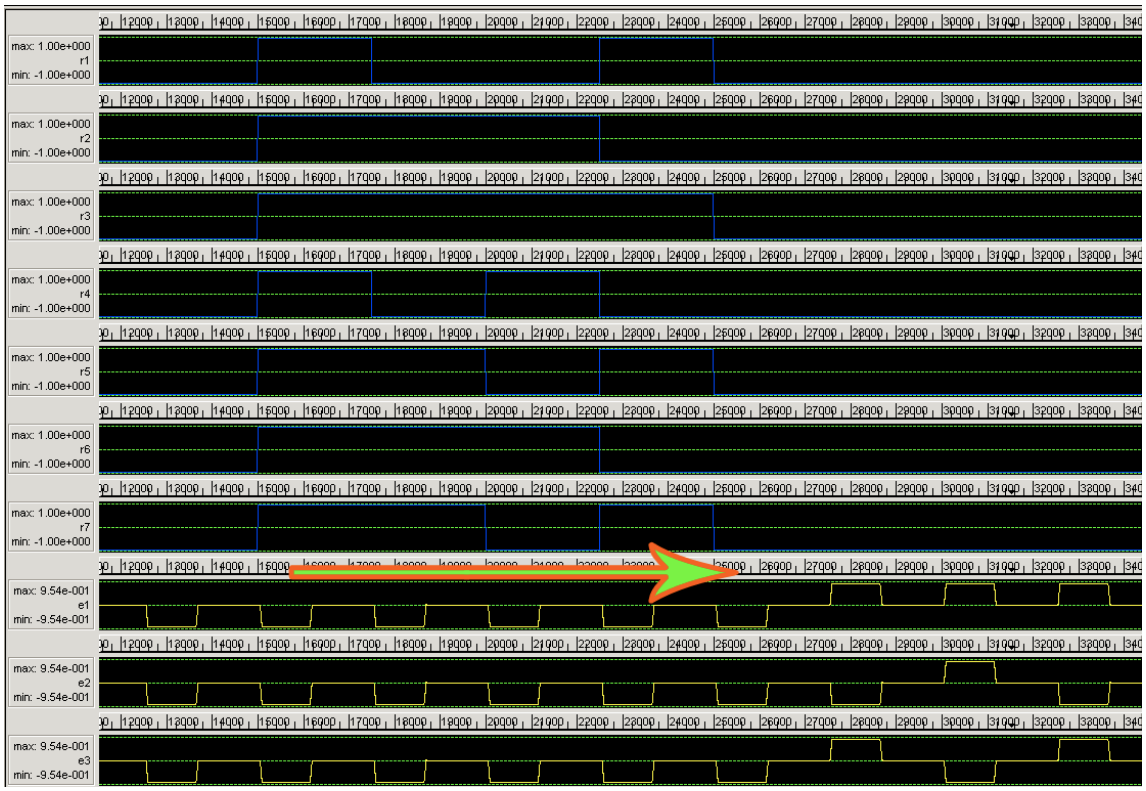

*Slika 17: QCA realizacija detekcije napake – srednji del*



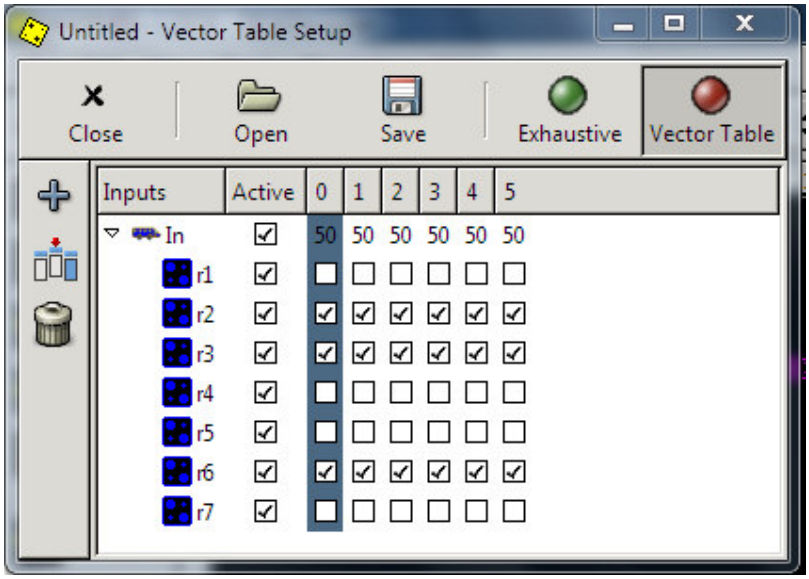

*Slika 18: QCA realizacija detekcije napake – desni del*

Rezultat simulacije je na Sliki 19. Izhodi pri podanih vhodih ustrezajo pravilnostnim tabelam iz poglavja 2.2. Zaradi preglednosti niso prikazane vse kombinacije. Izhodne vrednosti so za 4 urine periode zamaknjene glede na vhode, kar je na sliki označeno s puščico. Ker signal od vhoda do izhoda potuje čez 3 XOR vrata – vsaka imajo zakasnitev 1 urino periodo – to znaša 3 urine periode zakasnitve. Dodatno urino periodo prispevajo neravne linije oz. razvejitev, ki jih je potrebno za dosego stabilnosti delovanja ustrezno sinhronizirati s faznimi zamiki.

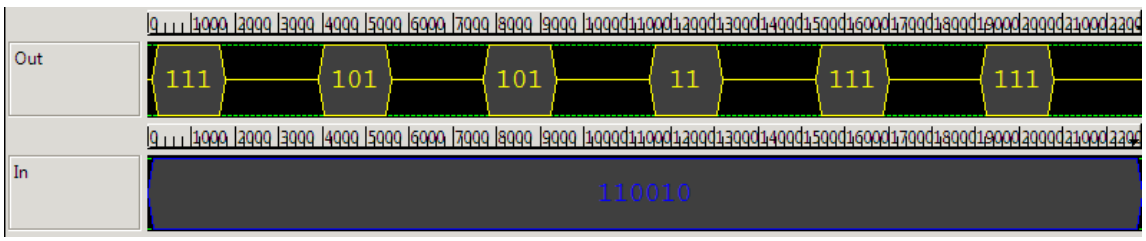
Primer delovanja ugotavljanja napake je primer, ko se pri sedmerici 0110011 pojavi napaka na zadnjem, sedmem bitu: 0110010. Tak vhodni vektor prikazuje Slika 20. Slika 21 prikazuje rezultat simulacije, kjer je razvidno, da kot izhod dobimo E = [111], kar pomeni pravilno ugotovitev, da je napaka na sedmem bitu.




*Slika 19: rezultati simulacije detekcije napake*




*Slika 20: vhodni vektor za ugotavljanje napake na 0110010*

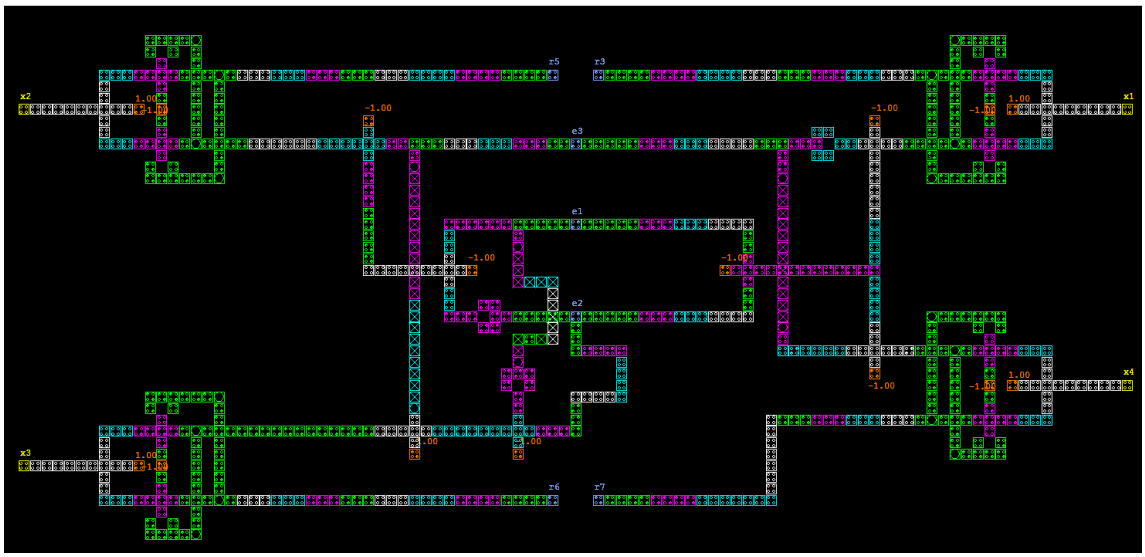



*Slika 21: rezultati simulacije za ugotavljanje napake na 0110010*

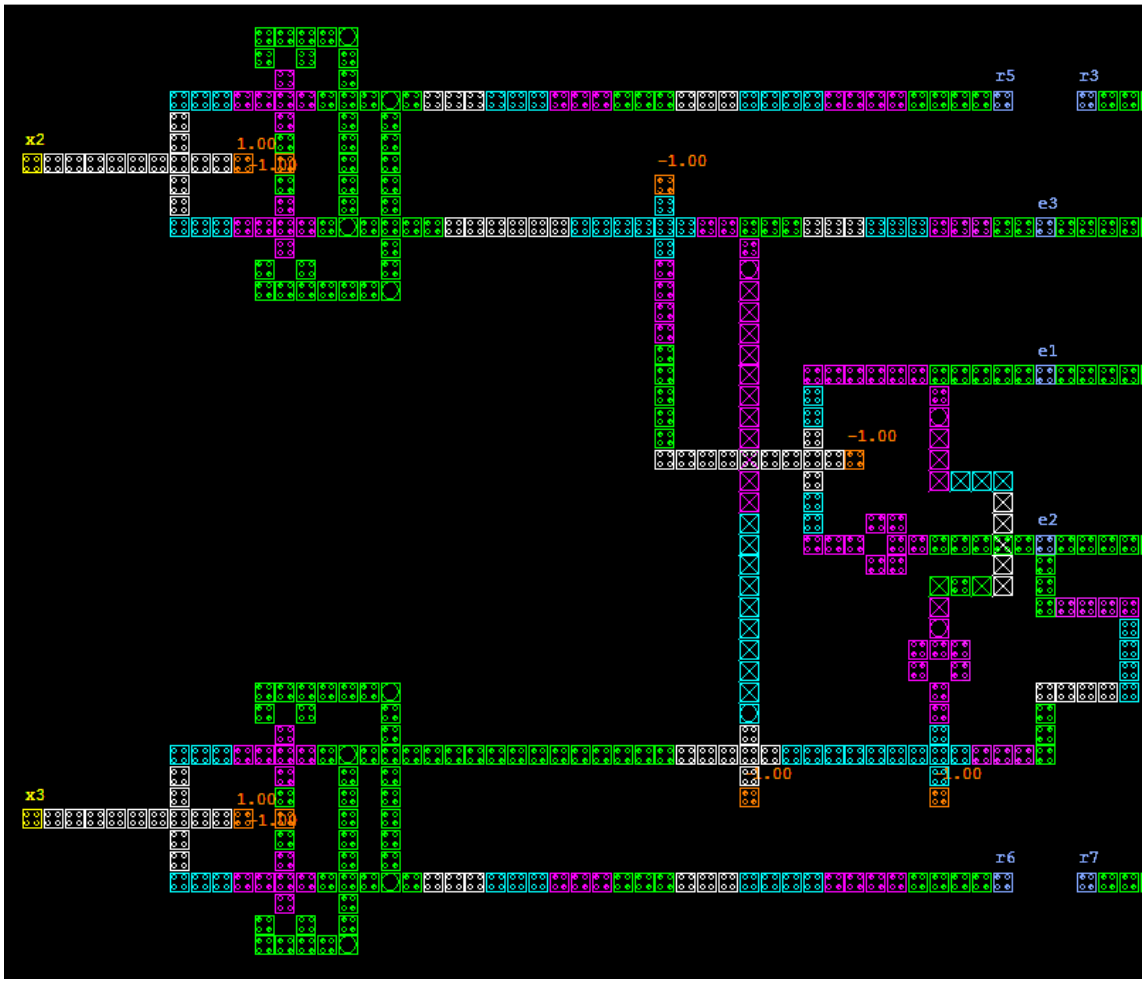
## 3.4. Dekodirnik - popravljanje napake

Realizacija popravljanja napake se izkaže za bolj zapleteno. V strukturi namreč nastopa večje število osnovnih gradnikov in križanja linij. Težavo predstavlja tudi pravilna sinhronizacija in stabilnost zaradi različnih dolžin linij v isti urini fazi.

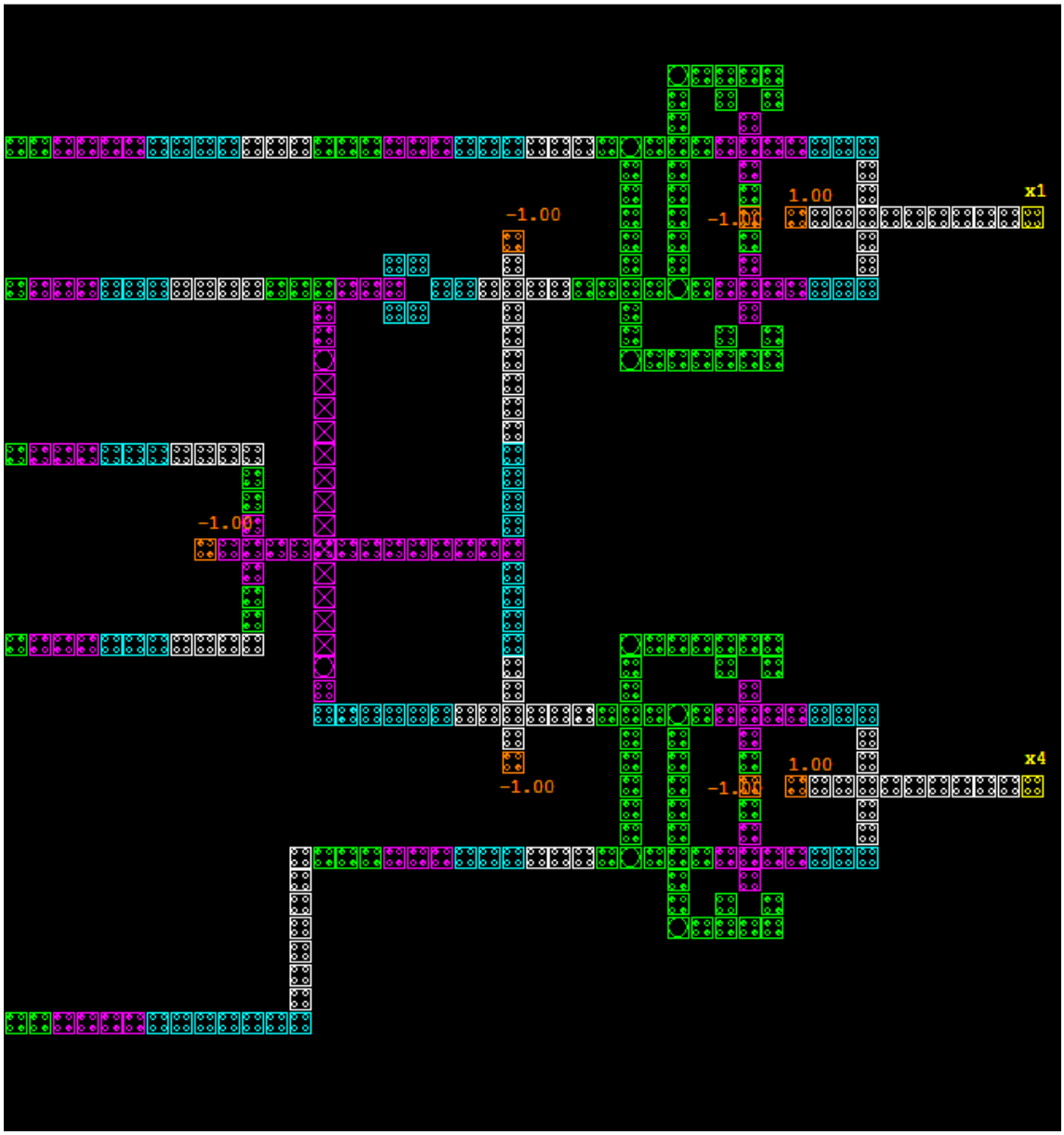
QCA realizacija popravljanja napake je prikazana na Sliki 22. Obsega 792 celic. Zaradi velikosti strukture je levi del sheme bolj pregledno prikazan na Sliki 23, desni del pa na Sliki 24.




*Slika 22: QCA realizacija popravljanja napake – celotna struktura*




*Slika 23: QCA realizacija popravljanja napake – levi del*

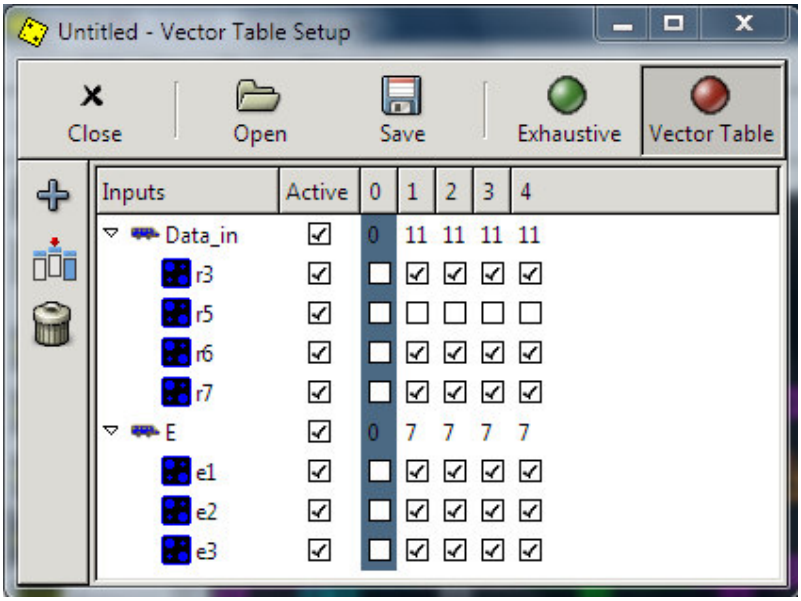



*Slika 24: QCA realizacija popravljanja napake – desni del*

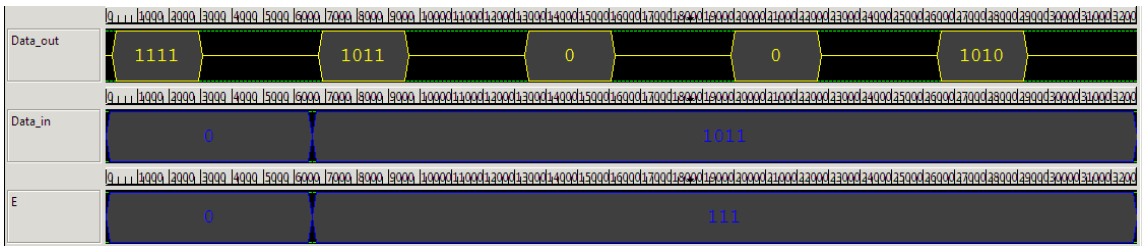
Rezultat simulacije je na Sliki 25. Zaradi preglednosti niso prikazane vse kombinacije. Izhodi pri podanih vhodih ustrezajo pravilnostnim tabelam iz poglavja 2.3. Izhodne vrednosti so za 3 urine periode zamaknjene glede na vhode, kar je na sliki označeno s puščico. Dve periodi prispeva sinhronizacija neravnih linij in razvejitev. Po teh dveh periodah vhodi vezja prispejo na vhode XOR vrat. Slednje nato potrebujejo še 1 periodo, da določijo končni izhod – skupna zakasnitev je tako 3 periode.

Primer simulacije na konkretnem primeru prikazuje popravljanje bloka 0110011, ki ima napačen sedmi bit. Vezje za detekcijo zato nastavi E=[111], podatkovni biti pa so 1011 – paritetni biti nas ne zanimajo. Na Sliki 26 sta vektorska vhoda za ta primer, na Sliki 27 pa rezultat simulacije – vidimo, da se napačni (zadnji) bit resnično popravi, tako da na izhodu dobimo pravilne podatkovne bite 1010.


*Slika 25: rezultati simulacije popravljanja napake*




*Slika 26: vhodna vektorja za popravljanje napake na 7. bitu pri podatkovnih bitih 1011*




*Slika 27: rezultati simulacije za popravljanje napake na zadnjem bitu*

# 4. Ugotovitve in zaključek

Realizirana vezja se izkažejo kot stabilna – pri simulaciji v vseh primerih vhodov dobimo pravilne izhode. Delovanje vezja smo preverjali z bistabilno simulacijo, ki pa smo ji zaradi večje natančnosti spremenili naslednje parametre:

- število vzorcev (*number of samples*): 32800,
- konvergenčna toleranca (*convergence tolerance*): 0.000001,
- maksimalno število iteracij za vzorec (*maximum iterations per sample*): 1100.

Vsa tri vezja skupaj obsegajo 2901 celic.

XOR gradnik smo v QCA zasnovali še v alternativni obliki na podlagi NAND vrat s ciljem izogniti se križanju povezav, a se taka izvedba ni izkazala za bistveno boljšo oz. je bila celo manj stabilna od na koncu uporabljene.

Kodirnik bi lahko kot QCA izdelali bolj optimalno, tj. s petimi namesto šestimi XOR vrati. Izraz ($x_3$ xor $x_{4)}$ nastopa v dveh izhodih, zato bi lahko za njegovo izvedbo porabili ena XOR vrata, izhod pa nato razvejili naprej.

QCA sheme so se izkazale za zelo občutljive, zato smo pri realizaciji morali biti pozorni na več faktorjev, kar se je izkazalo za učinkovito pri ohranjanju stabilnosti sistema in vplivalo na pravilno delovanje modulov:

- segmenti, ki so v različnih urinih fazah, morajo biti približno enako dolgi,
- na zavojih linij se zamenja urina faza,
- izogibanje predolgim povezam, oz. razbitje dolgih z različnimi fazami,
- izogibanje križanju povezav (tudi v več plasteh),
- simetrija pri postavljanju blokov (npr. XOR elementov),
- vsaj 2 celici z isto fazo na izhodu blokov (XOR, AND, NOT ipd.),
- več je manj - z večanjem števila celic se zelo hitro povečuje tudi možnost napake, oz. se manjše napake seštevajo, kar pa pripelje do nestabilnosti sistema in nepredvidljivih rezultatov.

# 5. Viri


- http://www.cs.uwp.edu/Classes/Cs340/Projects/PP1/Project1.pdf
- http://biobio.loc.edu/chu/web/Courses/Cosi460/hamming_codes.htm