Univerzitet u Nišu

Elektronski fakultet
Katedra za Elektroniku

Predmet:
DSP arhitekture i algoritmi

SerDes

(8b10b arhitektura)

Mentor:
prof. Mile Stojčev

Student:
Željko Banković 12154

Jul, 2010

Sadržaj

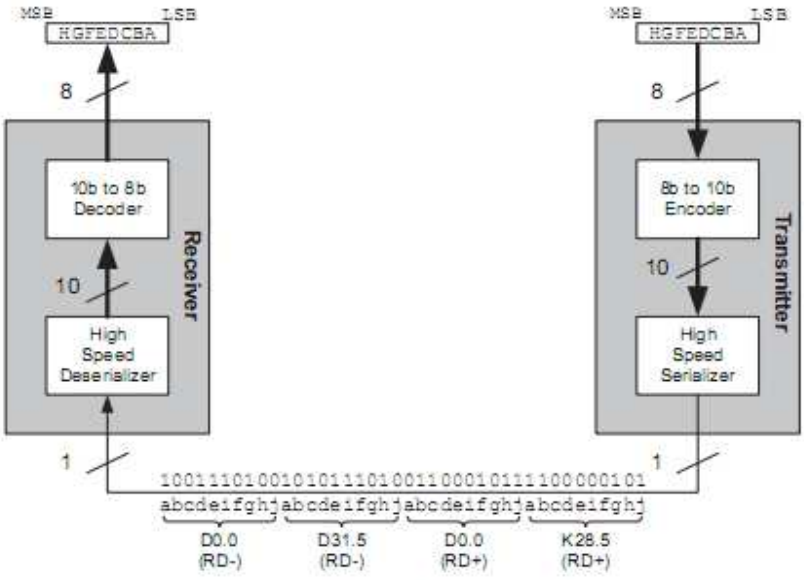
SerDes

(8b10b arhitektura)

SerDes (Serializer/Deserializer) se sastoji od dva bloka Parallel In Serial Out (PISO) ili paralelno-serijski konvertor i Serial In Parallel Out (SIPO) ili serijsko-paralelni konvertor. Ova dva bloka rade zajedno da bi se postigla velika brzina pri prenosu podataka. Osnova rada se zasniva na konvertovanju podataka iz paralelnog u serijski prenos i suprotno (na predajnoj strani se podaci upisuju paralelno, a šalju serijski, dok prijemni blok prihvata podatke serijski pa nakon toga konvertuje i šalje na paralelni izlaz). Generalno postoje četiri arhitekture SerDes-a: Parallel clock SerDes, Embedded clock SerDes, 8b/10b SerDes i Bit interleaved SerDes.

1. 8b10b arhitektura

Arhitektura 8b10b realizovana je od strane IBM-a 1983 godine. Ova arhitektura se koristi za brzi serijski prenos podataka. Jedna od glavnih prednosti ovakve arhitekture je u tome što je broj jedinica i nula koji se prenosi približno jednak, tj može da se razlikuje samo za ±2. Na ovaj način bilans snage je 50-50 [%]. Blok šema prikazana je na slici 1.



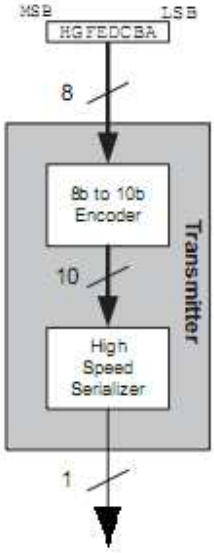
Slika 1.

Encoder na strani predajnika (Transmitter-a) svaki bajt (8-bita) sa ulaza koje prihvata paralelno, obrađuje i konvertuje u 10-bitni podatak koji se posle šalje serijski ka prijemniku preko shift registra. Shift registar Parallel In Serial Out (PISO) ili paralelno-serijski konvertor u suštini 10-bitni podatak iz Encodera prima paralelno i šalje serijski preko medijuma za prenos do Receivera tj prijemnika.

Na drugoj strani se nalazi prijemnik. Kao što se sa slike 1 vidi prijemnik se sastoji od jednog serijsko-paralelnog shift registra SIPO (Serial-in Parallel-out 10-bit Shift Register) i Decodera. Zadatak Decodera jeste da paralelno prihvati podatak iz shift registra, obradi i pošalje na izlaz 8-bitni podatak. Šema po kojoj Decoder pretvara 10-bitni ulazni podatak u 8-bitni izlazni identična je šemi kojom Encoder šalje 10-bitne podatke formirane na osnovu 8-bitnih podataka, samo je logika obrnuta.

2. Transmitter – predajnik

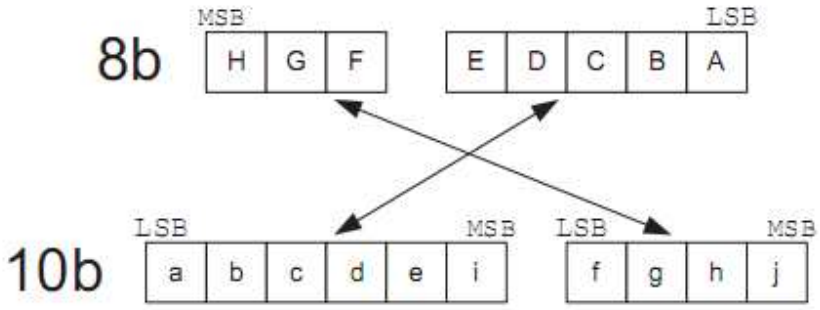
Blok šema predajnika prikazana je na slici 2. Generalno predajnik se sastoji od dva dela Encodera i shift registra. Blok Encoder je sastavljen od encodera i RAM memorije.



Slika 2.

2.1 Encoder

Princip rada Encodera je sledći. Ulazni podatak od 8-bita Encoder podeli u dve celine – tri viša bita i pet nižih (slika 3).



Slika 3.

Svakoj od ovih celina se dodaje jos po jedan bit (na slici su to bitovi ***i*** i ***j***). Na taj način se od 5-bitnog podatka formira 6-bitni i od 3-bitnog 4-bitni podatak. Spajanjem ova dva dela (6-bitnog i 4-bitnog podatka) dobija se 10-bitni podatak koji se salje na izlaz. Tabele koje se koriste za formiranje 3b/4b i 5b/6b podataka date su na slickama 4 i 5, respektivno.

3b Decimal	3b Binary (HGF)	4b Binary (fghi)
0	000	0100 or 1011
1	001	1001
2	010	0101
3	011	0011 or 1100
4	100	0010 or 1101
5	101	1010
6	110	0110
7	111	0001 or 1110 or 1000 or 0111

Slika 4.

5b Decimal	5b Binary (EDCBA)	6b Binary (abcdei)
0	00000	100111 or 011000
1	00001	011101 or 100010
2	00010	101101 or 010010
3	00011	110001
4	00100	110101 or 001010
5	00101	101001
6	00110	011001
7	00111	111000 or 000111
8	01000	111001 or 000110
9	01001	100101
10	01010	010101
11	01011	110100
12	01100	001101
13	01101	101100
14	01110	011100
15	01111	010111 or 101000
16	10000	011011 or 100100
17	10001	100011
18	10010	010011
19	10011	110010
20	10100	001011
21	10101	101010
22	10110	011010
23	10111	111010 or 000101
24	11000	110011 or 001100
25	11001	100110
26	11010	010110
27	11011	110110 or 001001
28	11100	001110
29	11101	101110 or 010001
30	11110	011110 or 100001
31	11111	101011 or 010100

Slika 5.

Kao što se vidi sa slika 4 i 5, za određene ulazne podatke postoje po dva moguća izlaza. Odluka koja će vrednost da se koristi zavisi od vrednosti bita RD (biće kasnije detaljnije objašnjeno u samoj realizaciji Encodera).

Tabelama koje su na slikama 4 i 5 pokrivene su sve 256 moguće kombinacije za prenos podataka (data). Međutim radi sinhronizacije predajnika i prijamnika koriste se nizovi tzv komandni (command) koji označavaju početak frejma, kraj frejma i slično. Za odvajanje komandnih nizova od nizova podataka korisiti se jedan dodatni bit koji u zavisnosti od svoje vrednosti kodira ulazne podatke prema tabelam 4 i 5 ako se radi o podacima (data) tj ako je njegova vrednost “0” ili na osnovu tabele na slici 6 ako se radi o komandnim bitovima (command) tj ako je njegova vrednost “1”.

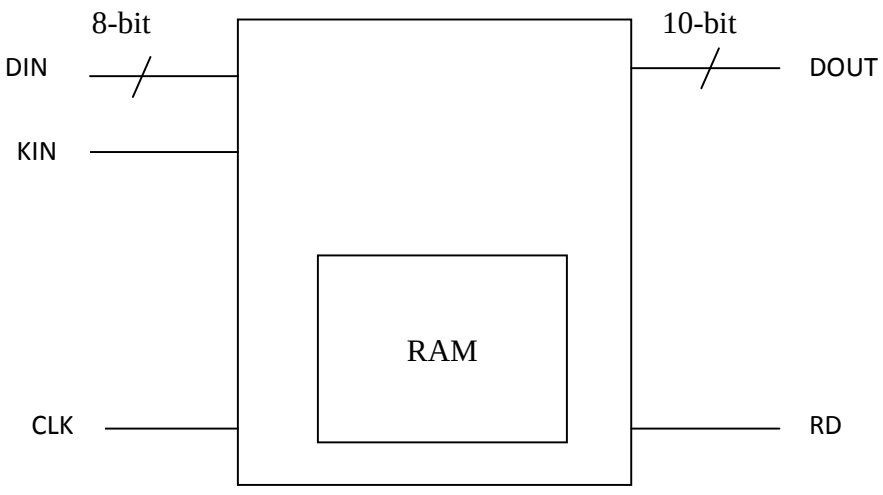
	HGF EDCBA	abcdei fghj	abcdei fghj
K.28.0	000 11100	001111 0100	110000 1011
K.28.1	001 11100	001111 1001	110000 0110
K.28.2	010 11100	001111 0101	110000 1010
K.28.3	011 11100	001111 0011	110000 1100
K.28.4	100 11100	001111 0010	110000 1101
K.28.5	101 11100	001111 1010	110000 0101
K.28.6	110 11100	001111 0110	110000 1001
K.28.7	111 11100	001111 1000	110000 0111
K.23.7	111 10111	111010 1000	000101 0111
K.27.7	111 11011	110110 1000	001001 0111
K.29.7	111 11101	101110 1000	010001 0111
K.30.7	111 11110	011110 1000	100001 0111

Slika 6.

Nakon što Encoder izvrši konverziju, podatak se upisuje u shift registar i šalje ka prijemniku kroz komunikacioni modul.

2.2 Realizacija Encoder-a

Blok šema Encodera data je na slici 7. Pinout je dat na slici 8.



Slika 7.

Naziv pina	Tip (ulazni/izlazni)	Dužina (u bitovima)	Opis pina
CLK	Ulazni	1	Takt kola
KIN	Ulazni	1	0- šalju se podaci; 1- šalju se komandne reči
DIN	Ulazni	8	Ulazni podaci
RD	Izlazni (opciono)	1	Pin koji pokazuje da li je prošlo više jedinica ili nula
DOUT	Izlazni	10	Izlazni podaci

Slika 8.

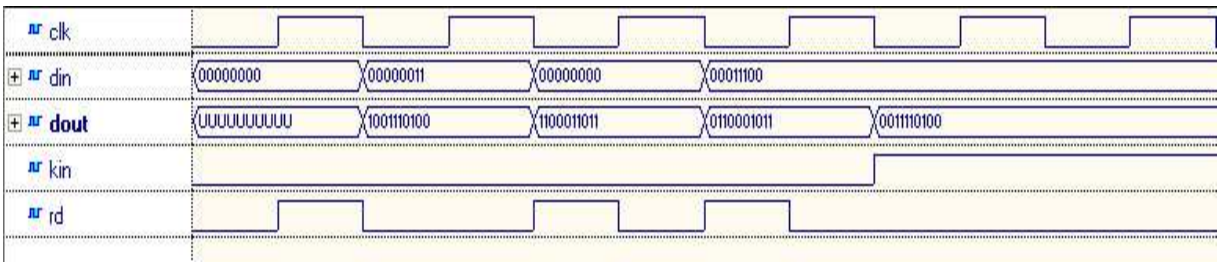
Kako kolo radi?

Ulazni podatak od 8-bita Encoder podeli u dve celine – tri viša bita i pet nižih. Vrednost signala RD se inicijalno postavi na “0”. Ovaj bit menja svoju vrednost na osnovu broja jedinica i nula u svakoj od celina. Ako je broj “1” veći od broja “0” onda se RD postavlja na “1” što znači da sledeća celina treba da dobije onu kombinaciju gde ima više “0” i obrnuto.

Na osnovu vrednosti KIN formira se 10-bitni izlazni podatak (ako je KIN na nuli onda se radi o podacima “data” i kodiranje se vrši na osnovu tabela 4 i 5, u suprotnom ako je vrednost jedan radi se o komandnim rečima “command”, tabela na slici 6).

U samom Encoderu se nalazi RAM memorija u kojoj su upisane tabela za formiranje podataka “data” i komandnih reči “command”. Izlaz se jednostavno formira isčitavanjem odgovarajućih kolona iz tabela, pri tome vodeći računa o vrednostima bitova KIN i RD. Ceo kod Encodera opisan na VHDL-u dat je u dodatku.

2.3 Testbench

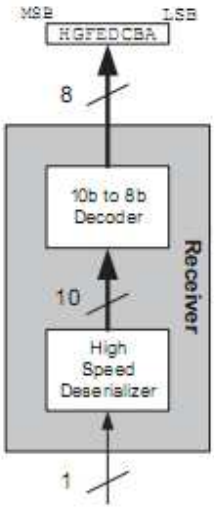


Slika 9.

Na slici 9 je dat oblik signala Encodera. Početna vrednost signala RD je “0” kao i vrednost KIN-a, što znači da se šalju klasični podaci “data” . Na ulazu se prvo dovodi kombinacija “00000000” što na izlazu generiše kombinaciju “1001110100” zato što je RD=“0” (rezultat se dobija na zadnju ivicu takta). S’obzirom da izlaz “1001110100” ima podjednak broj jedinica i nula vrednost signala RD će ostati isto na logičkoj “0”. Sledeći ulazni podatak je “00000011”. Za ovakav ulaz na izlazu se dobije kombinacija “1100011011”. Ova kombinacija ima veći broj jedini od nula (6 naspram 4), što dovodi da RD promeni svoju vrednost. Zatim se na ulaz ponovo dovodi kombinacija “00000000” ali ovog puta se izlaz razlikuje upravo zbog vrednosti RD-a. Sad izlaz ima vrednost “0110001011”. S’obzirom da je i ovde podjednak broj jedinica i nula RD zadržava staru vrednost. U narednom trenutku kombinacija koja se šalje je komandna reč “command”. S’obzirom da generisani izlaz za zadatu komadnu ima isti broj jedinica i nula signal RD zadrzava nadalje staru vrednost, što se može i videti sa slike. Ovim je dokazan ispravan rad Encodera.

3. Receiver – prijemnik

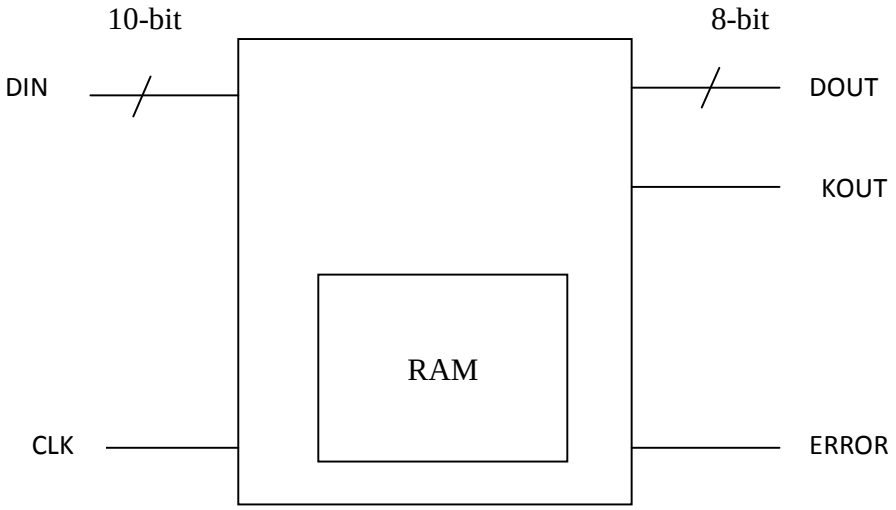
Blok šema prijemnika prikazana je na slici 10. Generalno prijemnik se sastoji od dva dela Decodera i shift registra. Blok Decoder je sastavljen od decodera i RAM memorije.



Slika 10.

3.1 Realizacija Decoder-a

Blok šema Decodera data je na slici 11. Pinout je dat na slici 12.



Slika 11.

Naziv pina	Tip (ulazni/izlazni)	Dužina (u bitovima)	Opis pina
CLK	Ulazni	1	Takt kola
DIN	Ulazni	10	Ulazni podaci
KOUT	Izlazni (opciono)	1	0- primljeni su podaci; 1- primljene su komandne reči
ERROR	Izlazni	1	Greška u prenosu
DOUT	Izlazni	8	Izlazni podaci

Slika 12.

Kako kolo radi?

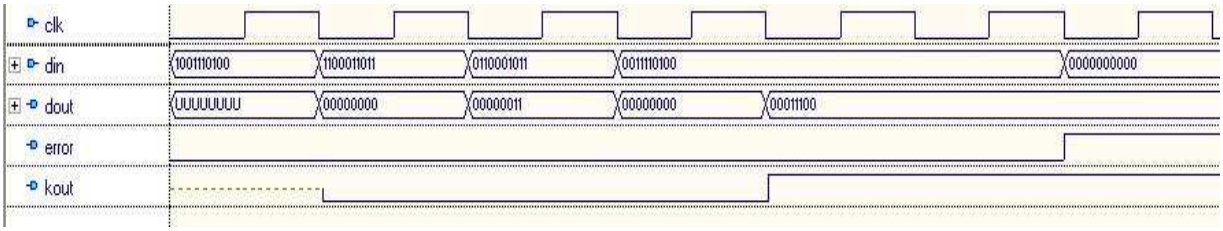
Ulazni podatak koji je 10-bitni se jednostavno upoređuje sa podacima kojima se nalaze u RAM memoriji. Proces je isti kao i prilikom formiranja izlaznog podataka kod Encodera samo je logika invertovana. Podatak sa ulaza se deli u dve celine. Prvu celinu čine 6-bita, a drugu 4-bita. Zatim se vrednosti ovih celina pretraže u tabelama 4 i 5 kako bi se pročitali njihovi 5-bitni tj 3-bitni ekvivalenti. Ako se desi da ne postoje vrednosti u tabelama 4 ili 5, onda se celokupni ulazni niz od 10 bita proverava sa vrednostima koje su upisane u memoriju i koje se odnose na

komandne reči. Ukoliko ni u tom slučaju ne postoji poklapanje, znači da je došlo do greške prilikom prenosa podataka (što će da dovede da se signal ERROR postavi na logičku “1”).

Signal KOUT postavlja svoju vrednost na “0” je ulazni podatak “data” ili na “1” ako ulazni podatak predstavlja komandnu reč “command”.

Izlaz od 8-bita dobija vrednost na osnovu pročitanih vrednosti iz tabela.

3.2 Testbench



Slika 13.

Na slici 13 su dati talasni oblici Decodera. Na ulaz Decodera je dovedena ista sekvenca koja je poslata od strane Encodera. Najpre na ulaz je dovedena kombinacija “1001110100”, što kada se prevede tj dekodira na izlazu daje kombinaciju “00000000”, a signal KOUT postavlja na “0” što znači da se radi o podacima “data”. Naredni podatak koji je pristigo je “1100011011”, što kad se prevede u 8-bitni podataka, na izlazu daje kombinaciju “00000011”. Zatim je na ulaz došla kombinacija “0110001011” što se takođe prevodi u kombinaciju “00000000”. Nakon toka je došla kobinacija “0011110100”. Dekoder j prepoznao da se radi o komandnoj reči i signal KOUT postavio na logičku jedinicu. Nakon ovog signal na ulazu se javila kombinacija koju Decoder nije uspeo da prepozna tj ne postoji odgovarajući 8-bitni ekvivalent za kombinaciju “0000000000” pa je Decoder prijavio grešku tako što je signal ERROR postavio na logičku jednicu. Ovim talasnim oblicima je potvrđeno da Decoder ispravno radi.

4. Zaključak

Jedna od glavnih prednosti ovakve arhitekture je u tome što je broj jedinica i nula koji se prenosi približno jednak, tj može da se razlikuje samo za ±2. Na ovaj način bilans snage je 50-50 [%] i srednja vrednost signala je nula.

5. Dodatak

5.1 Encoder - VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity encoder is
     port
     (
     clk, kin: in std_logic;
     din : in std_logic_vector(7 downto 0);
     dout : out std_logic_vector(9 downto 0);
     rd : out  std_logic
     );
end encoder;

architecture encoder of encoder is
signal rds : std_logic ;
signal enable : std_logic;
signal din5 : std_logic_vector(4 downto 0)  ;
signal din3 : std_logic_vector(2 downto 0) ;
signal dout6 : std_logic_vector(5 downto 0) ;
signal dout4 : std_logic_vector(3 downto 0) ;
signal douts : std_logic_vector(9 downto 0) ;

begin
     process (clk, kin, din)
     begin
          rds<='0';
          din5<=din(4 downto 0);
          din3<=din(7 downto 5);
          enable<='0';
          -- 5b6b data--
          if (clk'event and clk ='1') then
          if (kin='0') then

          case din5 is
               when "00000" =>
                    if (rds='0') then
                         dout6<="100111";
                         rds<= not rds;
```

```
     elsif (rds='1') then
          dout6<="011000";
          rds<= not rds;
     end if;

when "00001" =>
     if (rds='0') then
          dout6<="011101";
          rds<= not rds;
     elsif (rds='1') then
          dout6<="100010";
          rds<= not rds;
     end if;

when "00010" =>
     if (rds='0') then
          dout6<="101101";
          rds<= not rds;
     elsif (rds='1') then
          dout6<="010010";
          rds<= not rds;
     end if;

when "00011" =>     dout6<="110001";



when "00100" =>
     if (rds='0') then
          dout6<="110101";
          rds<= not rds;
     elsif (rds='1') then
          dout6<="001010";
          rds<= not rds;
     end if;

when "00101" =>     dout6<="101001";



when "00110" =>     dout6<="011001";


when "00111" =>
```

```
     if (rds='0') then
          dout6<="111000";
          rds<= not rds;
     elsif (rds='1') then
          dout6<="000111";
          rds<= not rds;
     end if;

when "01000" =>
     if (rds='0') then
          dout6<="111001";
          rds<= not rds;
     elsif (rds='1') then
          dout6<="000110";
          rds<= not rds;
     end if;

when "01001" =>     dout6<="100101";

when "01010" =>     dout6<="010101";

when "01011" =>     dout6<="110100";

when "01100" =>     dout6<="001101";

when "01101" =>     dout6<="101100";

when "01110" =>     dout6<="011100";

when "01111" =>
     if (rds='0') then
          dout6<="010111";
          rds<= not rds;
     elsif (rds='1') then
          dout6<="101000";
          rds<= not rds;
     end if;

when "10000" =>
     if (rds='0') then
          dout6<="011011";
          rds<= not rds;
     elsif (rds='1') then
          dout6<="100100";
          rds<= not rds;
     end if;
```

```
when "10001" =>     dout6<="100011";

when "10010" =>     dout6<="010011";

when "10011" =>     dout6<="110010";

when "10100" =>     dout6<="001011";

when "10101" =>     dout6<="101010";

when "10110" =>     dout6<="011010";

when "10111" =>
     if (rds='0') then
          dout6<="111010";
          rds<= not rds;
     elsif (rds='1') then
          dout6<="000101";
          rds<= not rds;
     end if;

when "11000" =>
     if (rds='0') then
          dout6<="110011";
          rds<= not rds;
     elsif (rds='1') then
          dout6<="001100";
          rds<= not rds;
     end if;

when "11001" =>     dout6<="100110";

when "11010" =>     dout6<="010110";

when "11011" =>
     if (rds='0') then
          dout6<="110110";
          rds<= not rds;
     elsif (rds='1') then
          dout6<="001001";
          rds<= not rds;
     end if;

when "11100" =>     dout6<="001110";

when "11101" =>
     if (rds='0') then
```

```
                    dout6<="101110";
                    rds<= not rds;
               elsif (rds='1') then
                    dout6<="010001";
                    rds<= not rds;
               end if;

          when "11110" =>
               if (rds='0') then
                    dout6<="011110";
                    rds<= not rds;
               elsif (rds='1') then
                    dout6<="100001";
                    rds<= not rds;
               end if;

          when others  =>
               if (rds='0') then
                    dout6<="101011";
                    rds<= not rds;
               elsif (rds='1') then
                    dout6<="010100";
                    rds<= not rds;
               end if;

    end case;
    end if;
    end if;
--end 5b6b data--


-- 3b4b data--
if(clk'event and clk='0') then
     enable<='1';
    if (kin='0') then

    case din3 is
          when "000" =>
               if (rds='0') then
                    dout4<="1011";
                    rds<= not rds;
               elsif (rds='1') then
                    dout4<="0100";
                    rds<= not rds;
               end if;

    when "001" =>  dout4<="1001";
```

```
          when "010" =>  dout4<="0101";

          when "011" =>
                    if (rds='0') then
                         dout4<="1100";
                         rds<= not rds;
                    elsif (rds='1') then
                         dout4<="0011";
                         rds<= not rds;
                    end if;

          when "100" =>
                    if (rds='0') then
                         dout4<="1101";
                         rds<= not rds;
                    elsif (rds='1') then
                         dout4<="0010";
                         rds<= not rds;
                    end if;

          when "101" =>  dout4<="1010";

          when "110" =>  dout4<="0110";

          when others =>
               if (rds='0') then
                    if( (din5="10001") or (din5="10010") or
(din5="10100") ) then
                              dout4<="0111";
                             rds<= not rds;
                    else
                         dout4<="1110";
                         rds<= not rds;
                    end if;
               elsif (rds='1') then
                    if( (din5="01011") or (din5="01101") or
(din5="01110") )  then
                              dout4<="1000";
                             rds<= not rds;
                    else
                             dout4<="0001";
                              rds<= not rds;
                    end if;
```

```
        end if;
        end case;
        end if;
        end if;
--end 3b4b data--

--12 controls--
if (kin='1') then

     case din is
          when "00011100" =>
               if (rds='0') then
                    douts<="0011110100";

               elsif (rds='1') then
                    douts<="1100001011";

               end if;

          when "00111100" =>
               if (rds='0') then
                    douts<="0011111001";
                    rds<= not rds;
               elsif (rds='1') then
                    douts<="1100000110";
                    rds<= not rds;
               end if;

          when "01011100" =>
               if (rds='0') then
                    douts<="0011110101";
                    rds<= not rds;
               elsif (rds='1') then
                    douts<="1100001010";
                    rds<= not rds;
               end if;


          when "01111100" =>
               if (rds='0') then
                    douts<="0011110011";
                    rds<= not rds;
               elsif (rds='1') then
                    douts<="1100001100";
                    rds<= not rds;
               end if;
```

```
when "10011100" =>
     if (rds='0') then
          douts<="0011110010";

     elsif (rds='1') then
          douts<="1100001101";

     end if;

when "10111100" =>
     if (rds='0') then
          douts<="0011111010";
          rds<= not rds;
     elsif (rds='1') then
          douts<="1100000101";
          rds<= not rds;
     end if;


when "11011100" =>
     if (rds='0') then
          douts<="0011110110";
          rds<= not rds;
     elsif (rds='1') then
          douts<="1100001001";
          rds<= not rds;
     end if;


when "11111100" =>
     if (rds='0') then
          douts<="0011111000";

     elsif (rds='1') then
          douts<="1100000111";

     end if;



when "11110111" =>
     if (rds='0') then
          douts<="1110101000";

     elsif (rds='1') then
          douts<="0001010111";
```

```
                end if;



           when "11111011" =>
                if (rds='0') then
                     douts<="1101101000";

                elsif (rds='1') then
                     douts<="0010010111";

                end if;


           when "11111101" =>
                if (rds='0') then
                     douts<="1011101000";

                elsif (rds='1') then
                     douts<="0100010111";

                end if;



           when others =>
                if (rds='0') then
                     douts<="0111101000";

                elsif (rds='1') then
                     douts<="1000010111";

                end if;


      end case;
      end if;
      end process;

      dout<=douts when kin='1' else
             dout6&dout4 when enable='1';


           rd<=rds;
end encoder;
```

5.2 Decoder – VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity decoder is
     port
     (
     clk : in std_logic;
     kout, error : out std_logic;
     din : in std_logic_vector(9 downto 0);
     dout : out std_logic_vector(7 downto 0)
     );
end decoder;

architecture decoder of decoder is
signal rd, enable, enablec : std_logic;
signal dout5 : std_logic_vector(4 downto 0);
signal dout3 : std_logic_vector(2 downto 0);
signal din6 : std_logic_vector(5 downto 0);
signal din4 : std_logic_vector(3 downto 0);
signal douts : std_logic_vector(7 downto 0);
signal error6, error4, error10 : std_logic;



begin
     process (clk, din)
     begin
          din6<=din(9 downto 4);
          din4<=din(3 downto 0);
          if(clk'event and clk='1') then
               enable<='0';
          case din6 is
               when "100111" => dout5<="00000";   error6 <='0';

               when "011000" => dout5<="00000";   error6 <='0';

               when "011101" => dout5<="00001";    error6 <='0';

               when "100010" => dout5<="00001";    error6 <='0';

               when "101101" => dout5<="00010";    error6 <='0';
```

```
when "010010" => dout5<="00010";    error6 <='0';
when "110001" => dout5<="00011";   error6 <='0';
when "110101" => dout5<="00100";   error6 <='0';
when "001010" => dout5<="00100";    error6 <='0';
when "101001" => dout5<="00101";    error6 <='0';
when "011001" => dout5<="00110";    error6 <='0';
when "111000" => dout5<="00111";    error6 <='0';
when "000111" => dout5<="00111";    error6 <='0';
when "111001" => dout5<="01000";    error6 <='0';
when "000110" => dout5<="01000";    error6 <='0';
when "100101" => dout5<="01001";    error6 <='0';
when "010101" => dout5<="01010";    error6 <='0';
when "110100" => dout5<="01011";    error6 <='0';
when "001101" => dout5<="01100";    error6 <='0';
when "101100" => dout5<="01101";    error6 <='0';
when "011100" => dout5<="01110";    error6 <='0';
when "010111" => dout5<="01111";    error6 <='0';
when "101000" => dout5<="01111";    error6 <='0';
when "011011" => dout5<="10000";   error6 <='0';
when "100100" => dout5<="10000";    error6 <='0';
when "100011" => dout5<="10001";    error6 <='0';
when "010011" => dout5<="10010";    error6 <='0';
when "110010" => dout5<="10011";    error6 <='0';
when "001011" => dout5<="10100";    error6 <='0';
```

```
      when "101010" => dout5<="10101";    error6 <='0';
      when "011010" => dout5<="10110";    error6 <='0';
      when "000101" => dout5<="10111";    error6 <='0';
      when "111010" => dout5<="10111";    error6 <='0';
      when "110011" => dout5<="11000";    error6 <='0';
      when "001100" => dout5<="11000";    error6 <='0';
      when "100110" => dout5<="11001";   error6 <='0';
      when "010110" => dout5<="11010";    error6 <='0';
      when "110110" => dout5<="11011";    error6 <='0';
      when "001001" => dout5<="11011";    error6 <='0';
      when "001110" => dout5<="11100";    error6 <='0';
      when "101110" => dout5<="11101";    error6 <='0';
      when "010001" => dout5<="11101";   error6 <='0';
      when "011110" => dout5<="11110";   error6 <='0';
      when "100001" => dout5<="11110";   error6 <='0';
      when "101011" => dout5<="11111";   error6 <='0';
      when "010100" => dout5<="11111";   error6 <='0';
      when others => error6 <= '1';


 end case;
 end if;
 --end 6b5b data--

--4b3b data--
  if(clk'event and clk='0') then
  enable<='1';
 case din4 is
```

```
     when "1011" => dout3<="000"; error4 <='0';

     when "0100" => dout3<="000";  error4 <='0';

     when "1001" => dout3<="001";  error4 <='0';

     when "0101" => dout3<="010";  error4 <='0';

     when "1100" => dout3<="011"; error4 <='0';

     when "0011" => dout3<="011";   error4 <='0';

     when "1101" => dout3<="100";   error4 <='0';

     when "0010" => dout3<="100";   error4 <='0';

     when "1010" => dout3<="101";   error4 <='0';

     when "0110" => dout3<="110";   error4 <='0';

     when "0111" => dout3<="111";  error4 <='0';

     when "1110" => dout3<="111";  error4 <='0';

     when "1000" => dout3<="111";   error4 <='0';

     when "0001" => dout3<="111";   error4 <='0';

     when others => error4 <= '1';  error4 <='0';

     end case;
--end 4b3b data---

--12 controls--
     enablec<='0';
case din is
     when "0011110100" => douts<="00011100";
                    error10 <='0'; enablec<='1';

     when "1100001011" => douts<="00011100";
                       error10 <='0';  enablec<='1';

     when "0011111001" => douts<="00111100";
                       error10 <='0';  enablec<='1';

     when "1100000110" => douts<="00111100";
                       error10 <='0';  enablec<='1';
```

```
  when "0011110101" => douts<="01011100";
                       error10 <='0'; enablec<='1';

  when "1100001010" => douts<="01011100";
                         error10 <='0';enablec<='1';

  when "0011110011" => douts<="01111100";
                       error10 <='0'; enablec<='1';

  when "1100001100" => douts<="01111100";
                       error10 <='0'; enablec<='1';

  when "0011110010" => douts<="10011100";
                       error10 <='0'; enablec<='1';

  when "1100001101" => douts<="10011100";
                       error10 <='0'; enablec<='1';

  when "0011111010" => douts<="10111100";
                       error10 <='0'; enablec<='1';

  when "1100000101" => douts<="10111100";
                       error10 <='0'; enablec<='1';

  when "0011110110" => douts<="11011100";
                       error10 <='0'; enablec<='1';

  when "1100001001" => douts<="11011100";
                       error10 <='0'; enablec<='1';

  when "0011111000" => douts<="11111100";
                       error10 <='0'; enablec<='1';

  when "1100000111" => douts<="11111100";
                       error10 <='0'; enablec<='1';

  when "1110101000" => douts<="11110111";
                       error10 <='0'; enablec<='1';

when "0001010111" => douts<="11110111";
                       error10 <='0'; enablec<='1';

  when "1101101000" => douts<="11111011";
                       error10 <='0';  enablec<='1';

  when "0010010111" => douts<="11111011";
```

```
                                error10 <='0'; enablec<='1';

                when "1011101000" => douts<="11111101";
                                error10 <='0'; enablec<='1';

                when "0100010111" => douts<="11111101";
                                error10 <='0'; enablec<='1';

                when "0111101000" => douts<="11111110";
                                error10 <='0';  enablec<='1';

                when "1000010111" => douts<="11111110";
                                error10 <='0';  enablec<='1';

                when others => error10 <='1'; enablec<='1';

           end case;
           end if;
     end process;

dout <= dout3&dout5 when (error4='0' and error6='0' and
                                                    enable='1')
           else
           douts when error10 = '0' and enable='1';

error <= '1' when ((error4='1' or error6='1') and error10 = '1')
                else
                '0';

kout <= '0' when  (error4='0' and error6='0' and error10='1' and
                                                    enablec ='1')
                else
               '1' when error10 = '0' and enablec ='1';

     end decoder;
```

6. Literatura

- http://www.xilinx.com/support/documentation/ip_documentation/encode_8b10b.pdf
- http://www.xilinx.com/support/documentation/ip_documentation/decode_8b10b.pdf
- http://en.wikipedia.org/wiki/SerDes
- http://en.wikipedia.org/wiki/8B/10B_encoding