DİJİTAL SİSTEM TASARIMI PROJE RAPORU

VHDL ile Konvolüsyonel Kodlayıcı Tasarımı

Ahmet Salih BALANDİ

1. Giriş

Dijital haberleşme sistemlerinde iletilen veriler, kanal gürültüsü ve bozulmalar nedeniyle hatalara maruz kalabilmektedir. Bu hataların etkisini azaltmak amacıyla hata düzeltme kodlama teknikleri kullanılmaktadır. Bu tekniklerden biri olan konvolüsyonel encoderlar, giriş bitlerini geçmiş bitlerle ilişkilendirerek birden fazla çıkış biti üretir ve böylece alıcı tarafta hata tespit ve düzeltme imkânı sağlar.

Bu proje kapsamında, kodlama oranı R = 1/3 ve kısıt uzunluğu K = 3 olan bir konvolüsyonel kodlayıcı tasarlanmıştır. Tasarım süreci; durum diyagramının oluşturulması, durum geçiş tablosunun çıkarılması, mantık devresinin elde edilmesi, VHDL behavioral modelinin yazılması ve testbench ile doğrulanması adımlarını içermektedir.

2. Konvolüsyonel Kodlayıcının Teorik Yapısı

Tasarlanan kodlayıcıda her bir giriş biti için üç adet çıkış üretilmektedir. Bu durum, R = 1/3 kodlama oranı anlamına gelmektedir. Kodlayıcı yapısında giriş verisi bir shift register üzerinden işlenmekte ve geçmiş girişler de çıkışların hesaplanmasına dâhil edilmektedir.

Bu projede kullanılan üreteç polinomları aşağıdaki gibidir:

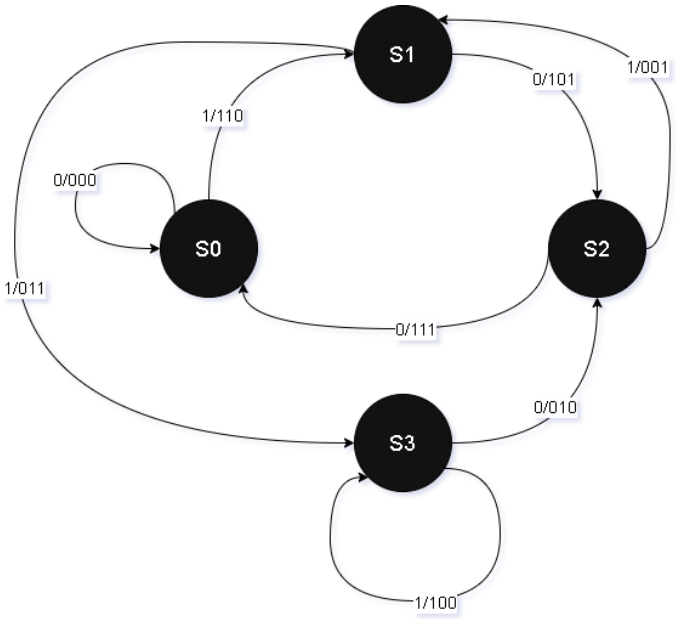
- G1 = 111
- G2 = 101
- G3 = 011

Çıkışlar sadece mevcut girişe değil, aynı zamanda önceki giriş bitlerine de bağlıdır. Bu durum sistemin memory-based bir yapı olmasını sağlar ve FSM (Finite State Machine) yaklaşımını zorunlu kılar.

3. Durum Diyagramı ve Mantıksal Anlamı

Encoderin constraint length değeri K = 3 olduğu için, sistemin belleğinde iki adet geçmiş bit tutulmaktadır. Bu da toplam $2^2 = 4$ farklı durum oluşmasına neden olur. Bu durumlar S0, S1, S2 ve S3 olarak tanımlanmıştır. Durum diyagramında her durum, shift registerdaki önceki iki bitin kombinasyonunu temsil eder. Sisteme uygulanan her yeni giriş biti ile birlikte yeni giriş registera eklenir, en eski bit kaydırılarak çıkarılır ve sistem bir sonraki duruma geçer

Bu mekanizma, mevcut durum ve giriş bilindiğinde hem bir sonraki durum hem de çıkışların kesin olarak belirlenmesini sağlar ve deterministik bir yapı oluşturur. Bu özellik, tasarımın Mealy tipi FSM olarak modellenmesine imkân tanımaktadır.



Şekil 1: Durum Diyagramı

4. Durum Geçiş Tablosunun Açıklaması

Durum geçiş tablosu, her bir mevcut durum ve giriş kombinasyonu için bir sonraki durumu ve üretilen çıkış bitlerini göstermektedir. Bu tablo oluşturulurken üreteç polinomları doğrudan kullanılmıştır. Örneğin, G1 = 111 polinomu; mevcut giriş, bir önceki giriş ve iki önceki giriş bitlerinin XOR'lanmasıyla Out1 çıkışını üretmektedir. Benzer şekilde diğer polinomlar da ilgili çıkışları belirlemektedir.

Bu tablo sayesinde, durum diyagramı ile matematiksel kodlama kuralları arasında doğrudan bir ilişki kurulmuştur. Böylece teorik model, donanımsal tasarıma aktarılabilir hâle getirilmiştir.

Present State	P. State (Binary)	Input	Next State	Next State (Binary)	d(n)	d(n-1)	d(n-2)	Out1	Out2	Out3
S0	00	0	S0	00	0	0	0	0	0	0
S0	00	1	S1	10	1	0	0	1	1	0
S1	10	0	S2	01	0	1	0	1	0	1
S1	10	1	S3	11	1	1	0	0	1	1
S2	01	0	S0	00	0	0	1	1	1	1
S2	01	1	S1	10	1	0	1	0	0	1
S3	11	0	S2	01	0	1	1	0	1	0
S3	11	1	S3	11	1	1	1	1	0	0

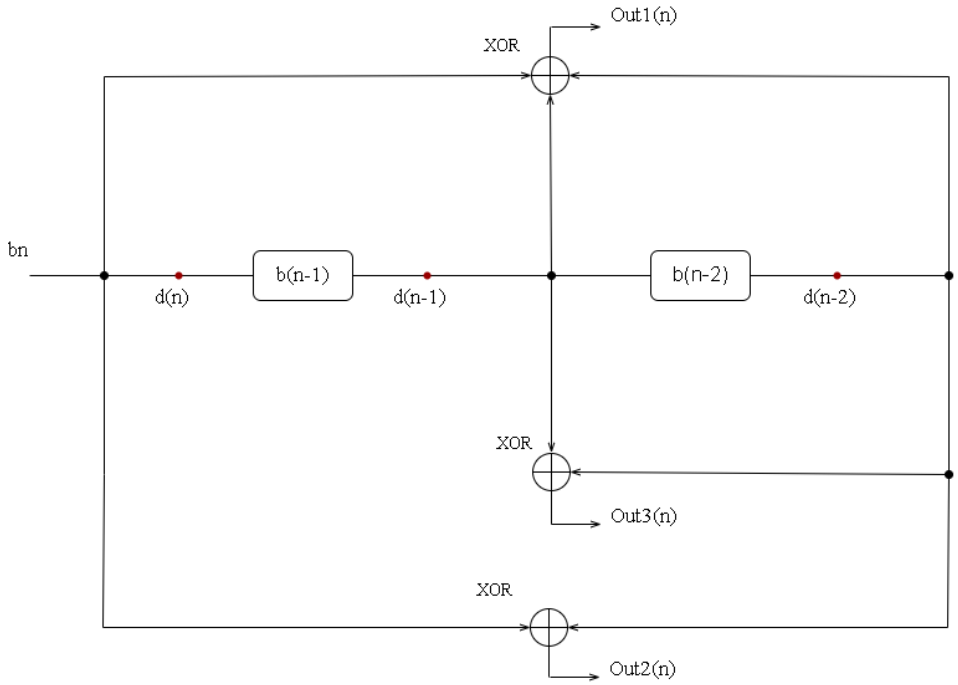
Tablo 1: Durum Geçiş Tablosu

5. Mantık Devresinin Tasarımı

Durum geçiş tablosu esas alınarak, kodlayıcının mantık devresi tasarlanmıştır. Devre yapısı temel olarak şu bloklardan oluşmaktadır:

- D-tipi flip-floplar (durum belleği için)
- XOR kapıları (register polinomlarını gerçekleştirmek için)
- Kombinasyonel mantık (bir sonraki durum ve çıkış hesaplaması için)

Bu yapıda flip-floplar, sistemin senkron çalışmasını sağlar. Saat sinyalinin her pozitif kenarında durum güncellenirken, çıkışlar anlık giriş ve mevcut duruma bağlı olarak hesaplanır. Bu da FSM davranışının donanımsal karşılığıdır.



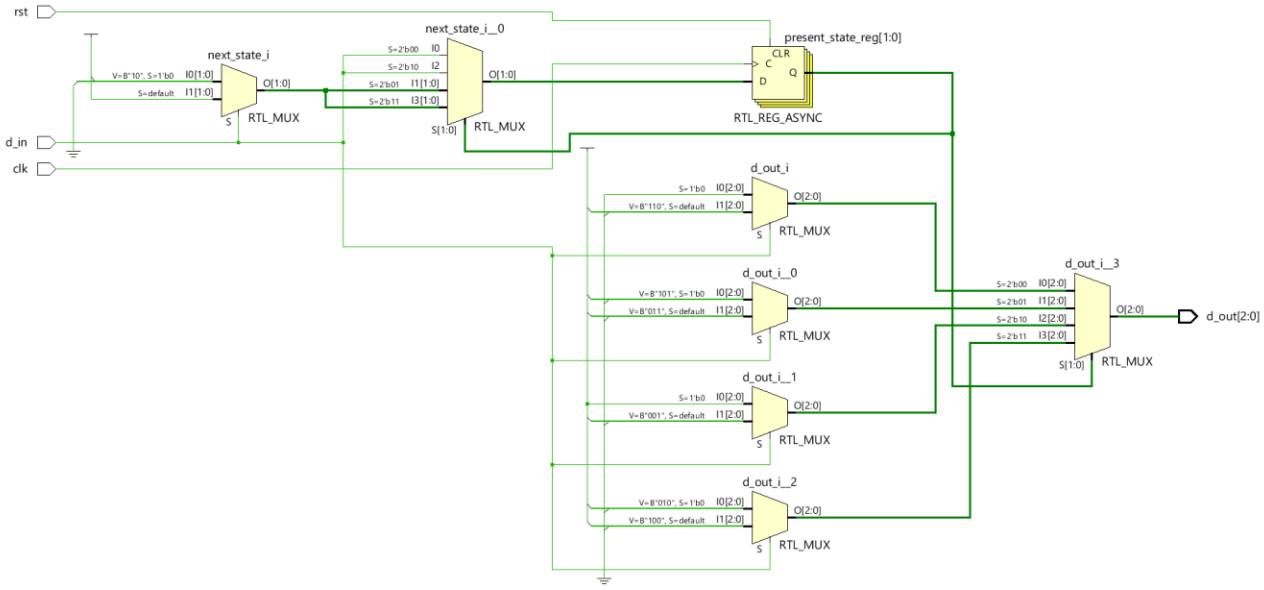
Şekil 2: Mantık Devresi

6. VHDL Behavioral Model ve Şematiği

Mantık devresinin donanımsal davranışı, VHDL dilinde behavioral modelleme yaklaşımıyla kodlanmıştır. Kod yapısı genel olarak iki ana süreçten oluşmaktadır:

1. Senkron süreç: Saat ve reset sinyaline bağlı olarak mevcut durumun güncellenmesi
2. Kombinasyonel süreç: Mevcut duruma ve girişe bağlı olarak bir sonraki durum ve çıkışların hesaplanması

Bu ayrım, FSM tasarım prensiplerine uygun bir yapı sunar ve kodun okunabilirliğini artırır. Reset sinyali ile sistem başlangıç durumu olan S0'a alınmaktadır.



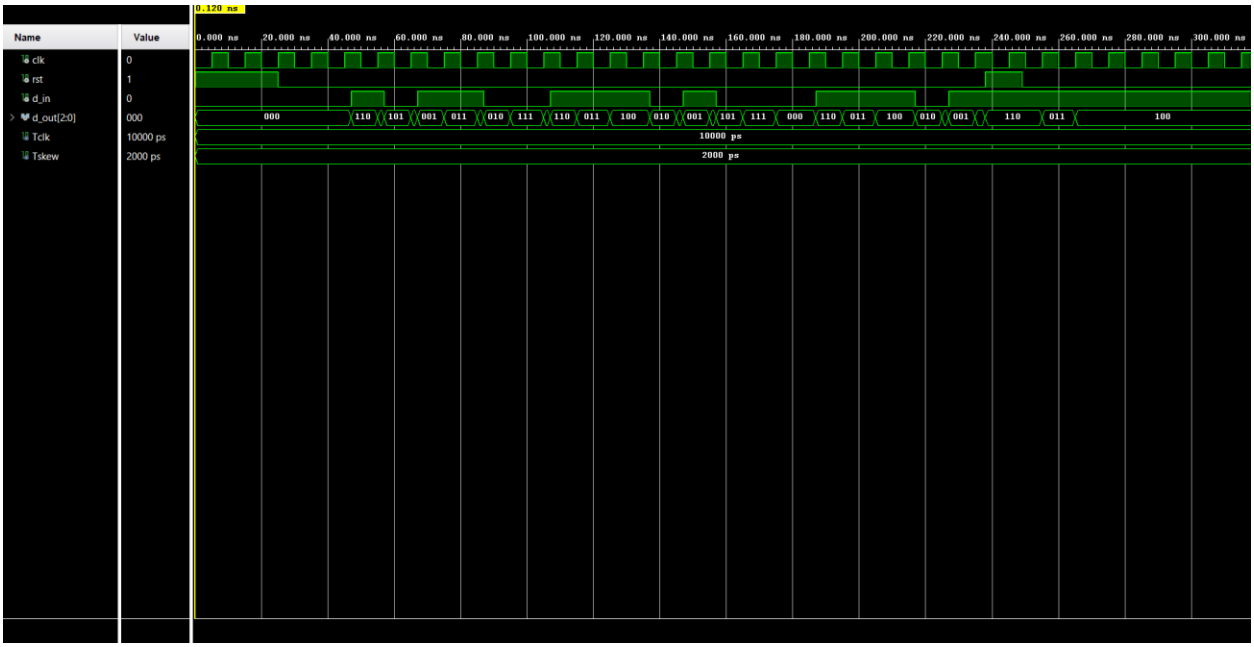
Şekil 3: Encoder Şematiği

(Kodlar sayfa sonunda yer almaktadır.)

7. Testbench ve Simülasyon Sonuçları

Tasarımın doğruluğunu test etmek amacıyla bir VHDL testbench hazırlanmıştır. Testbench içerisinde saat sinyali üretilmiş, farklı giriş bit dizileri uygulanmış ve tüm durum geçişleri kapsanacak şekilde senaryolar oluşturulmuştur.

Simülasyon sonuçları incelendiğinde, her giriş dizisi için üretilen çıkışların teorik olarak beklenen konvolüsyonel kod çıktılarıyla birebir örtüştüğü gözlemlenmiştir. Bu durum, hem FSM modelinin hem de VHDL kodunun doğru şekilde çalıştığını göstermektedir.



Şekil 4: Simülasyon Sonucu. Burada simülasyon kararlılığı için input ve clock edge'leri birbirlerine asenkrondur.

(Kodlar sayfa sonunda yer almaktadır.)

8. Sonuç

Bu projede, R = 1/3 ve K = 3 parametrelerine sahip bir konvolüsyonel kodlayıcı, FSM yaklaşımı kullanılarak başarıyla tasarlanmıştır. Durum diyagramı, durum geçiş tablosu ve mantık devresi arasında tutarlı bir yapı oluşturulmuş; bu yapı VHDL behavioral model ile donanımsal olarak gerçekleştirilmiştir.

Testbench simülasyonları, sistemin tüm durum ve giriş kombinasyonlarında doğru çalıştığını doğrulamıştır. Bu çalışma, konvolüsyonel kodlayıcıların dijital sistemler ve hata düzeltme teknikleri açısından önemini açık bir şekilde ortaya koymaktadır.

Ek: VHDL Kodları

(carbon.now.sh sitesinden oluşturulmuştur)

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity vhdl_midterm_project is
    Port (
        clk   : in  STD_LOGIC; -- Clock signal
        rst   : in  STD_LOGIC; -- Asynchronous Reset
        d_in  : in  STD_LOGIC; -- Input d(n)
        d_out  : out STD_LOGIC_vector(2 downto 0) -- Out1(n),Out2(n),Out3(n)
    );
end vhdl_midterm_project;

architecture Behavioral of vhdl_midterm_project is

    -- Define the states as shown in your diagram
    -- S0="00", S1="10", S2="01", S3="11"
    type state_type is (S0, S1, S2, S3);
    signal present_state, next_state : state_type;

begin

    -- Process 1: Synchronous Logic (State Memory)
    -- Updates current state on rising clock edge
    sync_proc: process(clk, rst)
    begin
        if rst = '1' then
            present_state <= S0; -- Reset to initial state 00
        elsif clk'event and clk='1' then
            present_state <= next_state;
        end if;
    end process;

    -- Process 2: Combinational Logic (Next State & Output Logic)
    -- Determines next state and outputs based on Input and Current State
    comb_proc: process(present_state, d_in)
    begin
        -- Default assignments to prevent latches
        next_state <= present_state;
        d_out<="000";

        case present_state is

            -- State S0 (d(n-1)=0, d(n-2)=0)
            when S0 =>
                if d_in = '0' then
                    next_state <= S0;
                    d_out<="000";
                else
                    next_state <= S1;
                    d_out<="110";
                end if;

            -- State S1 (d(n-1)=1, d(n-2)=0)
            when S1 =>
                if d_in = '0' then
                    next_state <= S2;
                    d_out<="101";
                else
                    next_state <= S3;
                    d_out<="011";
                end if;

            -- State S2 (d(n-1)=0, d(n-2)=1)
            when S2 =>
                if d_in = '0' then
                    next_state <= S0;
                    d_out<="111";
                else
                    next_state <= S1;
                    d_out<="001";
                end if;

            -- State S3 (d(n-1)=1, d(n-2)=1)
            when S3 =>
                if d_in = '0' then
                    next_state <= S2;
                    d_out<="010";
                else
                    next_state <= S3;
                    d_out<="100";
                end if;

        end case;
    end process;

end Behavioral;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_vhdl_midterm_project is
end entity;

architecture sim of tb_vhdl_midterm_project is

    signal clk   : std_logic := '0';
    signal rst   : std_logic := '0';
    signal d_in  : std_logic := '0';
    signal d_out : std_logic_vector(2 downto 0);

    constant Tclk : time := 10 ns;
    constant Tskew : time := 2 ns;   -- input edge ile clk edge çakışmasın

begin

    -- DUT
    uut: entity work.vhdl_midterm_project
        port map (
            clk   => clk,
            rst   => rst,
            d_in  => d_in,
            d_out => d_out
        );

    -- clock
    clk <= not clk after Tclk/2;

    stim_proc: process
        constant seq : std_logic_vector := "01011001110100011101";
    begin
        -- async reset
        d_in <= '0';
        rst  <= '1';
        wait for 25 ns;
        rst  <= '0';

        wait until rising_edge(clk);

        for i in seq'range loop
            wait for Tskew;          -- input edge ile clk edge çakışmasın
            d_in <= seq(i);

            wait until rising_edge(clk);

            report "t=" & time'image(now) &
                   "  applied(d_in)=" & std_logic'image(d_in) &
                   "  observed(d_out)=" &
                   std_logic'image(d_out(2)) &
                   std_logic'image(d_out(1)) &
                   std_logic'image(d_out(0));
        end loop;

        report "---- Async reset test ----";
        wait for 3 ns;
        rst <= '1';
        wait for 11 ns;
        rst <= '0';

        wait until rising_edge(clk);
        wait for Tskew;
        d_in <= '1';
        wait until rising_edge(clk);
        report "After async reset: d_out=" &
               std_logic'image(d_out(2)) &
               std_logic'image(d_out(1)) &
               std_logic'image(d_out(0));

        report "SIM DONE" severity note;
        wait;
    end process;

end architecture;
```