KOCAELİ ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ

UART İLE KLAVYE KARAKTERLERİNİN ALGILANMASI ve ASCII KODLARININ SEVEN SEGMENT ÜZERİNDE GÖSTERİMİ

MEH 443:FPGA İLE SAYISAL TASARIMA GİRİŞ

PROJE RAPORU

Emre ALABAY

150207051

Bölümü: Elektronik ve Haberleşme Mühendisliği

Danışman: Dr. Anıl ÇELEBİ

KOCAELİ, 2018

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

KISALTMALAR ve TERİMLER

UART: Universal Asyncronous Receiver-Transmitter

FPGA: Field Programmable Gate Array

SSD: Seven Segment Display

GPIO: General Purpose Input Output

I2C(IIC): Inter Intergrated Circuit

SPI: Serial Peripheral Interface

ASCII: American Standard Code for Information Interchange

PCB: Printed Circuit Board

UART İLE KLAVYE KARAKTERLERİNİN ALGILANMASI ve ASCII KODLARININ SEVEN SEGMENT ÜZERİNDE GÖSTERİMİ

Emre ALABAY

Anahtar Kelimeler: UART Haberleşmesi, FPGA Programlama, GPIO Arayüzü, Seven Segment Display

Özet: Günümüzde sistemler arası veri akatarımı I2C, SPI, Ethernet gibi protokollerle sağlanmakta olup UART haberleşmesi ise genelikle kontrol amacı ile kullanılmaktadır.

Projemizde kullanıcı tarafından klavyede basılan karakterlere karşılık gelen ascii kodlarını hexadecimal olarak seven segmentte göstermek amaçlanmıştır.

1.GİRİŞ

Projemizin yazılım kısmı dört bölümden oluşmaktadır. Bunlar uart modülü, saat bölücü modülü, binary-ssd decoder modülü ve top modüldür. Top modülü diğer modülleri içeren, modüllerin giriş-çıkışlarının birbirleriyle ilişkilendiriliği ve sinyal atamalarının yapıldığı bölümdür.

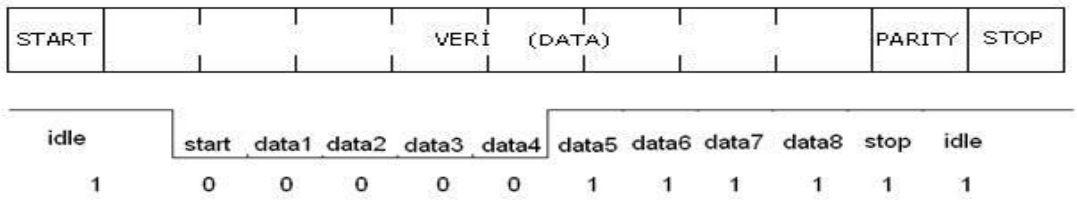
Projemizde kullandığımız Zybo Z7-20 FPGA kitinde UART IP'si bulunmaktadır fakat VHDL ile UART haberleşmesi sağlanacaktır. FPGA kartının programlanması Vivado 2018.2 programında gerçekleştirilmiştir.

Projenin donanım tasarımı kısmında ise Altium Designer programı kullanılmıştır.

2.UART HABERLEŞMESİ

UART asenkron bir haberleşme protokolüdür. Bunun sebebi data ile herhangi bir clock bilgisinin gönderilmemesidir. Alınan data ile gönderilen datanın senkronizasyonun sağlanması ve gelen verinin anlamlı hale gelmesi için baudrate terimi geliştirilmiştir. Baudrate datanın gönderilme hızını belirler ve böylelikle alıcı taraf bu hıza göre gelen sinyalden örnekler almaktadır.

Projemizde 8 bit data biti, 1start biti ve 1 stop bitinden oluşan 10 bitlik ve 115200 baudrate değerine sahip UART haberleşmesi kullanılacaktır.



Şekil 2. 1: UART Haberleşmesi Sinyalleri

2.1 UART Modülü Tasarımı

Projemizde klavyeden basılan karakterlerin ASCII kodlarının seven segmentte gösterilmesi gerçekleştirileceği yani FPGA kartının transmitter olarak değil sadece receiver olarak kullanılacağı için UART RX modülü tasarlanacaktır.

Doğru veri alışverişinin gerçekleştirilmesi için modül tasarımında en çok dikkat edilmesi gereken noktalar, sayıcı ile baudrate oluşturulması ve gelen veriden ne zaman örnek alınacağıdır.[1]

UART haberleşmesi sıralı bir işleyiş içerisinde olduğu için UART_RX modülünün State Machine(Durum Makinesi) mantığı ile yazılması uygun görülmüştür. Durum makinelerinde birden çok durum mevcuttur ve koşullara bağlı olarak bir durumdan diğer duruma geçilir. İşleyiş döngü şeklinde devam eder. UART modülü tasarlanırken input olarak i_Clk saat sinyali ve i_RX_Serial adında USB_TTL dönüştürücünün TX pinini bağladığımız gelen seri datanın alındığı başka bir giriş değişkeni daha tanımlanmıştır. Alınan verinin seven segment modülüne gönderilmesi için gelen verinin tutulduğu o_RX_Byte adında logic vector

tanımlanmıştır. Ayrıca baudrate ayarı counter ile sağlandığı için bir bitin örneklenmesi için gerekli clock değerini tanımlayan g_CLKS_PER_BIT değişkeni tanımlanmıştır. Bu değer denklem (1) ile hesaplanır.

clks_per_bit = i_clk(125 MHz) / 115200 = 1085 (1)

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

entity UART_RX is
generic (
    g_CLKS_PER_BIT : integer := 1085
    );
    Port ( i_Clk : in STD_LOGIC;
           i_RX_Serial : in STD_LOGIC;
           o_RX_Byte : out STD_LOGIC_VECTOR (7 downto 0));
end UART_RX;
```

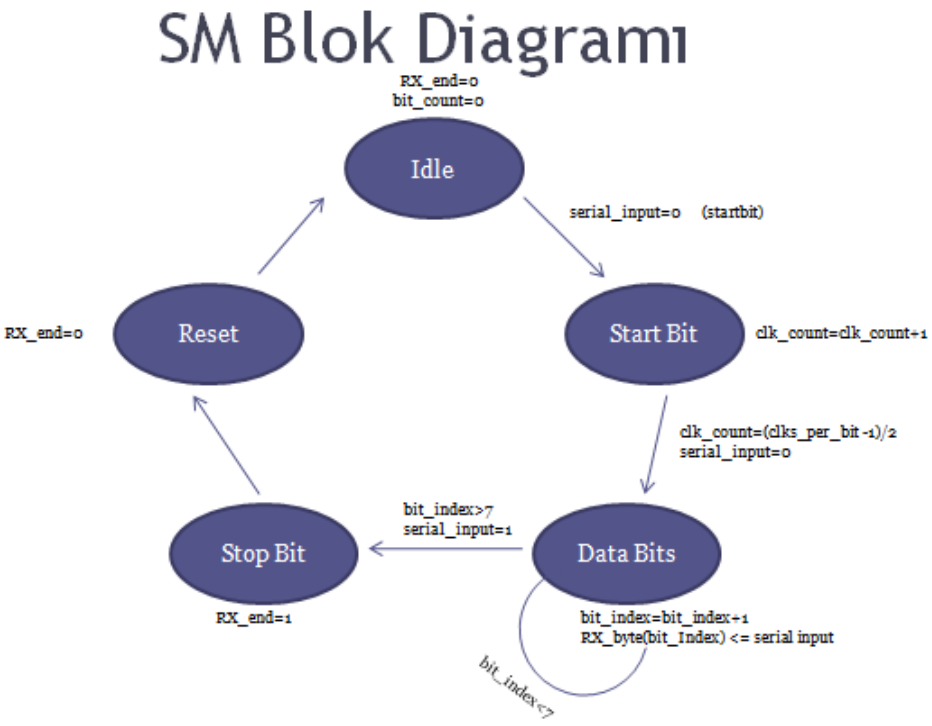
Şekil 2. 2: UART Modülü Giriş Çıkışları

Port tanımlamalarının ardından durum makinesinin state' lerini tanımlamak adına t_SM_Main adında enumerated tipinde bir değişken tanımlanmıştır. Process içerisinde kullanılmak üzere de state'lerin tutulduğu r_SM_Main, counter değerinin tutulduğu r_Clk_Count, gelen seri datanın tutulduğu r_RX_Byte, indislerinin tutulduğu r_Bit_Index ve iletimin tamamlandığını ifade eden r_RX_end sinyali tanımlanmıştır.

```
architecture RTL of UART_RX is

  type t_SM_Main is (s_Idle, s_RX_Start_Bit, s_RX_Data_Bits,
                     s_RX_Stop_Bit, s_Cleanup);
  signal r_SM_Main   : t_SM_Main := s_Idle;

  signal r_Clk_Count : integer range 0 to g_CLKS_PER_BIT-1 := 0;
  signal r_Bit_Index : integer range 0 to 7 := 0;
  signal r_RX_Byte   : std_logic_vector(7 downto 0) := (others => '0');
  signal r_RX_end     : std_logic := '0';
```

Şekil 2. 3: Sinyal Tanımlamaları

İşlemesi planlanan durum makinesi blok diagramı Şekil 2. 1. 2 de görselleştirilmiştir.



Şekil 2. 4: Durum makinesi blok diagramı

Durum makinesi veri transferi olmadığı durumda s_Idle durumundadır. s_Idle durumunda iken seri data hattı high yani lojik "1" durumundadır. Haberleşme başladığında bu hat lojik "0" a çekilir. Hattın "0" a çekildiği kontrol edilerek bir sonraki s_RX_Start_Bit durumuna geçilir aksi takdirde s_Idle durumunda beklenir.

```
UART_RX : process (i_Clk)
begin
  if rising_edge(i_Clk) then

    case r_SM_Main is

      when s_Idle =>
        r_RX_end    <= '0';
        r_Clk_Count <= 0;
        r_Bit_Index <= 0;

        if i_RX_Serial = '0' then
          r_SM_Main <= s_RX_Start_Bit;
        else
          r_SM_Main <= s_Idle;
        end if;
```

Şekil 2. 5: s_Idle Durumu

Haberleşme başladığında r_Clk_Count sayıcı sinyali saymaya başlar. g_CLKS_PER_BIT-1/2 değerine eşit olduğunda yani seri datanın ortasında iken örnek alınır böylece datanın sıfır olup olmadığından emin olunur ve sayıcı sıfırlanır ardından s_RX_Data_Bits durumuna geçilir.

```
when s_RX_Start_Bit =>
  if r_Clk_Count = (g_CLKS_PER_BIT-1)/2 then
    if i_RX_Serial = '0' then
      r_Clk_Count <= 0;
      r_SM_Main   <= s_RX_Data_Bits;
    else
      r_SM_Main   <= s_Idle;
    end if;
  else
    r_Clk_Count <= r_Clk_Count + 1;
    r_SM_Main   <= s_RX_Start_Bit;
  end if;
```

Şekil 2. 6: s_RX_Start_Bit Durumu

Counter baudrate' i sağlayan g_CLKS_PER_BIT değerine kadar sayar ve seri portun o anki değerini r_RX_Byte(r_Bit_Index) sinyaline atar. r_Bit_Index değeri 7' ye ulaştığında yani tüm seri data alındığında s_RX_Stop_Bit durumuna geçilir.

```
when s_RX_Data_Bits =>
  if r_Clk_Count < g_CLKS_PER_BIT-1 then
    r_Clk_Count <= r_Clk_Count + 1;
    r_SM_Main   <= s_RX_Data_Bits;
  else
    r_Clk_Count             <= 0;
    r_RX_Byte(r_Bit_Index) <= i_RX_Serial;


    if r_Bit_Index < 7 then
      r_Bit_Index <= r_Bit_Index + 1;
      r_SM_Main   <= s_RX_Data_Bits;
    else
      r_Bit_Index <= 0;
      r_SM_Main   <= s_RX_Stop_Bit;
    end if;
  end if;
```

Şekil 2. 7: s_RX_Data_Bits Durumu

Stop bitinin ardından tekrar s_Idle durumuna geçilerek yeni data beklenir.

```
        when s_RX_Stop_Bit =>

          if r_Clk_Count < g_CLKS_PER_BIT-1 then
            r_Clk_Count <= r_Clk_Count + 1;
            r_SM_Main   <= s_RX_Stop_Bit;
          else
            r_RX_end     <= '1';
            r_Clk_Count <= 0;
            r_SM_Main   <= s_Cleanup;
          end if;

        when s_Cleanup =>
          r_SM_Main <= s_Idle;
          r_RX_end   <= '0';

        when others =>
          r_SM_Main <= s_Idle;

      end case;
    end if;
  end process UART_RX;
```

Şekil 2. 8: s_RX_Stop_Bits,s_Cleanup Durumları

3.SEVEN SEGMENT DISPLAY

3.1 Binary-SSD Decoder Modülü Tasarımı

UART ile gelen veri 8 bitliktir. Bu verinin 2 digitli seven segmentte gösterilebilmesi için 4'er bit olarak bölünmüştür ve 4x7 seven segment decoder[2] tasarlanmıştır. Decoder tasarlanırken ortak anot seven segment için her hexadecimal sayının karşılığını gösteren Tablo 3.1 oluşturulmuştur.

Tablo 3. 1: Ortak Anot Doğruluk Tablosu

DIGIT	DPGFEDBCA	DP	G	F	E	D	C	B	A
0	X"C0"	1	1	0	0	0	0	0	0
1	X"F9"	1	1	1	1	1	0	0	1
2	X"A4"	1	0	1	0	0	1	0	0
3	X"B0"	1	0	1	1	0	0	0	0
4	X"99"	1	0	0	1	1	0	0	1
5	X"92"	1	0	0	1	0	0	1	0
6	X"82"	1	0	0	0	0	0	1	0
7	X"F8"	1	1	1	1	1	0	0	0
8	"80"	1	0	0	0	0	0	0	0
9	X"90"	1	0	0	1	0	0	0	0
A	X"88"	1	0	0	0	1	0	0	0
B	X"83"	1	0	0	0	0	0	1	1
C	X"C6"	1	1	0	0	0	1	1	0
D	X"A1"	1	0	1	0	0	0	0	1
E	X"86"	1	0	0	0	0	1	1	0
F	X"8E"	1	0	0	0	1	1	1	0

Tasarlanan modülün VHDL kodu aşağıdaki gibidir.

```
entity Binary_To_7Segment is
    Port (
           i_Clk : in STD_LOGIC;
           i_Binary_Num : in STD_LOGIC_VECTOR (3 downto 0);
           o_Segment_A : out STD_LOGIC;
           o_Segment_B : out STD_LOGIC;
           o_Segment_C : out STD_LOGIC;
           o_Segment_D : out STD_LOGIC;
           o_Segment_E : out STD_LOGIC;
           o_Segment_F : out STD_LOGIC;
           o_Segment_G : out STD_LOGIC);

end Binary_To_7Segment;

architecture RTL of Binary_To_7Segment is

  signal hex_code : std_logic_vector(7 downto 0) := (others => '0');

begin

  process (i_Clk) is
  begin
    if rising_edge(i_Clk) then
      case i_Binary_Num is
        when "0000" =>          hex_code <= X"C0";
        when "0001" =>          hex_code <= X"F9";
        when "0010" =>          hex_code <= X"A4";
        when "0011" =>          hex_code <= X"B0";
        when "0100" =>          hex_code <= X"99";
        when "0101" =>          hex_code <= X"92";
        when "0110" =>          hex_code <= X"82";
        when "0111" =>          hex_code <= X"F8";
        when "1000" =>          hex_code <= X"80";
        when "1001" =>          hex_code <= X"90";
        when "1010" =>          hex_code <= X"88";
        when "1011" =>          hex_code <= X"83";
        when "1100" =>          hex_code <= X"C6";
        when "1101" =>          hex_code <= X"A1";
        when "1110" =>          hex_code <= X"86";
        when "1111" =>          hex_code <= X"8E";
      end case;
    end if;
  end process;

  o_Segment_A <= hex_code(0);
  o_Segment_B <= hex_code(1);
  o_Segment_C <= hex_code(2);
  o_Segment_D <= hex_code(3);
  o_Segment_E <= hex_code(4);
  o_Segment_F <= hex_code(5);
  o_Segment_G <= hex_code(6);

end architecture RTL;
```

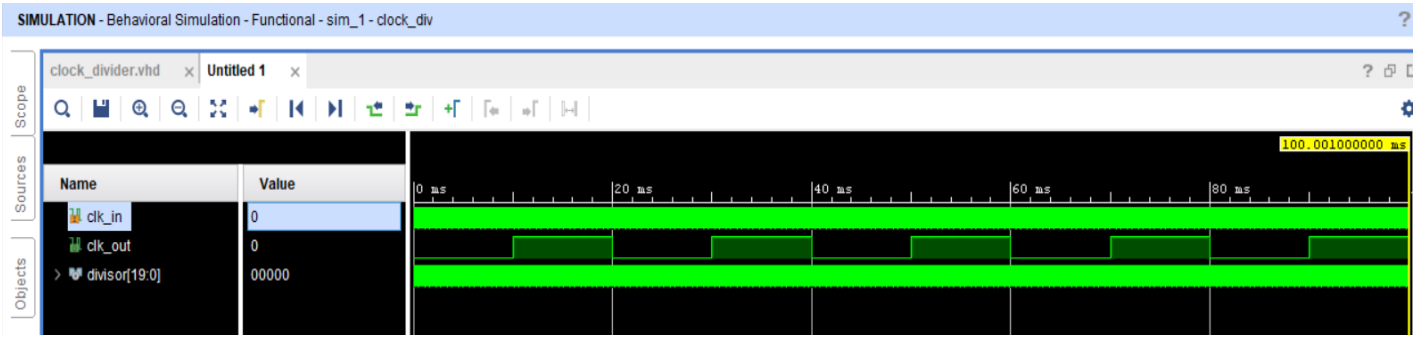
Şekil 3. 1 :SSD Modülü

3.2 Clock Divider Modülü Tasarımı

PMOD seven segmentin[3] digit seçimi için tek bir pin olup digit değişimi için devredeki schmitt-trigger inverter entegresi kullanılmaktadır. Bu değişim için de işlemci frekansından daha düşük frekanslı bir saat sinyali gereklidir. Bu nedenle saat sinyalini 125 MHz'den 50Hz'e dönüştüren Şekil 3.3'te verilen clock_divider modülü tasarlanmıştır. Saat bölücünün simülasyonu da Şekil 3.4' te mevcuttur.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity clock_divider is

port (
   i_Clk: in std_logic;
   o_Clk_Slow: out std_logic
  );
end clock_divider;

architecture RTL of clock_divider is
signal divisor: std_logic_vector(23 downto 0):=(others =>'0');
begin
 process(i_Clk)
 begin
 if(rising_edge(i_Clk)) then
    divisor <= divisor + x"000001";
 if(divisor>=x"26259F") then
    divisor <= x"000000";
 end if;
 end if;
 end process;
 o_Clk_Slow <= '0' when divisor < x"1312D0" else '1';
 end architecture RTL;
```

Şekil 3. 3 :Clock_divider VHDL kodu



Şekil 3. 2 : Clock_divider simülasyonu

4. TOP MODÜL

Top modül tasarımında Uart modülü ile alınan veri 4 bitlik iki parçaya bölünerek iki digitte ayrı ayrı gösterilmiştir. Bunun için top modülde gelen veriyi aldığımız in_UART_RX ve segmentleri yakmak için kullandığımız out_Segment pinleri Şekil4. 1'deki gibidir.

```
entity UART_RX_To_7_Seg_Top is
  port (

    i_Clk         : in std_logic;
    in_UART_RX     : in std_logic;

    out_Segment_A  : out std_logic;
    out_Segment_B  : out std_logic;
    out_Segment_C  : out std_logic;
    out_Segment_D  : out std_logic;
    out_Segment_E  : out std_logic;
    out_Segment_F  : out std_logic;
    out_Segment_G  : out std_logic;
    out_CLK_ANODE : out std_logic

    );

end entity UART_RX_To_7_Seg_Top;
```

Şekil 4. 1 : Top modül giriş ve çıkışları

İki digit için ayrı ayrı sinyaller tanımlanmıştır. Saat bölücü sinyali çıkışındaki yavaşlatılmış sinyale bağlı olarak digitler sürekli olarak değişmektedir ve buna bağlı olarak out_Segment çıkışlarına değişimli olarak bu sinyaller atanmaktadır. Bu akışı gösteren VHDL kod Şekil 4.2' de gösterilmiştir.

```
architecture RTL of UART_RX_To_7_Seg_Top is


  signal w_RX_Byte : std_logic_vector(7 downto 0);
  signal w_Segment1_A, w_Segment2_A : std_logic;
  signal w_Segment1_B, w_Segment2_B : std_logic;
  signal w_Segment1_C, w_Segment2_C : std_logic;
  signal w_Segment1_D, w_Segment2_D : std_logic;
  signal w_Segment1_E, w_Segment2_E : std_logic;
  signal w_Segment1_F, w_Segment2_F : std_logic;
  signal w_Segment1_G, w_Segment2_G : std_logic;
  signal w_seg_select : std_logic;
```

Şekil 4. 2 : Sinyal tanımlamaları

Alt modüllerin top modüle çağrılması, port yönlendirmeleri ve slow clock' a bağlı segment değişimi Şekil 4.3 'teki VHDL kodda mevcuttur.

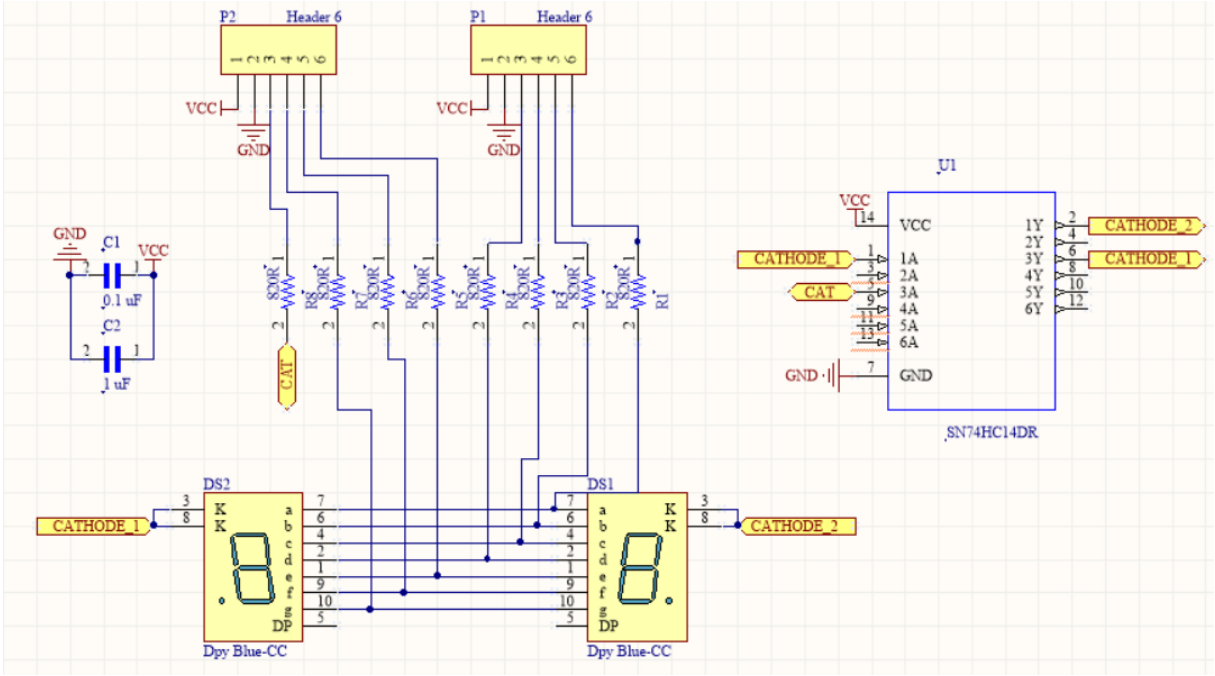
```
begin

  UART_RX : entity work.UART_RX
    generic map (
      g_CLKS_PER_BIT => 1085)            -- 125,000,000 / 115,200
    port map (
      i_Clk       => i_Clk,
      i_RX_Serial => in_UART_RX,

      o_RX_Byte   => w_RX_Byte);


  SevenSeg1: entity work.Binary_To_7Segment
    port map (
      i_Clk        => i_Clk,
      i_Binary_Num => w_RX_Byte(7 downto 4),
      o_Segment_A  => w_Segment1_A,
      o_Segment_B  => w_Segment1_B,
      o_Segment_C  => w_Segment1_C,
      o_Segment_D  => w_Segment1_D,
      o_Segment_E  => w_Segment1_E,
      o_Segment_F  => w_Segment1_F,
      o_Segment_G  => w_Segment1_G
      );


  SevenSeg2: entity work.Binary_To_7Segment
    port map (
      i_Clk        => i_Clk,
      i_Binary_Num => w_RX_Byte(3 downto 0),
      o_Segment_A  => w_Segment2_A,
      o_Segment_B  => w_Segment2_B,
      o_Segment_C  => w_Segment2_C,
      o_Segment_D  => w_Segment2_D,
      o_Segment_E  => w_Segment2_E,
      o_Segment_F  => w_Segment2_F,
      o_Segment_G  => w_Segment2_G
      );

   clock_divider: entity work.clock_divider
     port map (
        i_Clk          => i_Clk,
        o_Clk_Slow     => w_seg_select
        );

   Segment_Select :
      process(w_seg_select)
      begin
        if w_seg_select ='1' then

            out_Segment_A <= w_Segment2_A;
            out_Segment_B <= w_Segment2_B;
            out_Segment_C <= w_Segment2_C;
            out_Segment_D <= w_Segment2_D;
            out_Segment_E <= w_Segment2_E;
            out_Segment_F <= w_Segment2_F;
            out_Segment_G <= w_Segment2_G;

        elsif w_seg_select ='0' then

            out_Segment_A <= w_Segment1_A;
            out_Segment_B <= w_Segment1_B;
            out_Segment_C <= w_Segment1_C;
            out_Segment_D <= w_Segment1_D;
            out_Segment_E <= w_Segment1_E;
            out_Segment_F <= w_Segment1_F;
            out_Segment_G <= w_Segment1_G;
        end if;
    end process Segment_Select;
      out_CLK_ANODE <= w_seg_select;
end architecture RTL;
```

Şekil 4. 3 : Top Modül

5.PMOD SSD PCB TASARIMI

5.1 Şematik Tasarımı

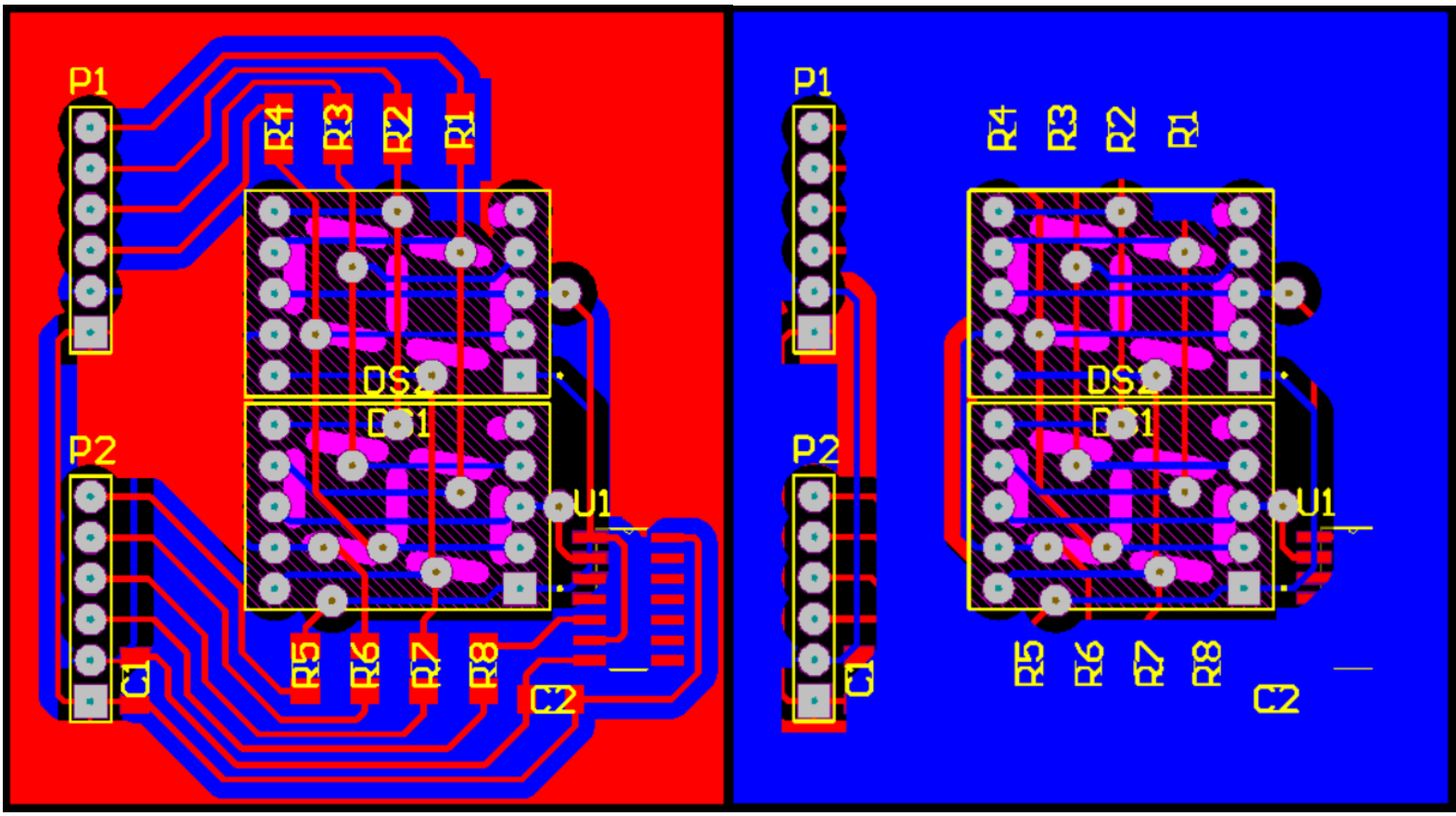
Şematik tasarlanırken digilent firmasının sitesindeki şematikten[3] yararlanılmıştır. Tasarım yapılırken iki ayrı digit ve schmitt_trigger inverter'ın muadili[4] kullanlmıştır. Kullanılan entegre, dirençler ve kapasite smd kılıf olarak seçilmiştir.Smd direnç.[5] ve shmitt-trigger[6] entegresi şematik ve pcb kütüphaneleri Snapeda adlı siteden hazır olarak alınmıştır . Smd direnç ve kapasiteler 1206 kılıfında seçilmiştir.Çizilen şematik Şekil 4.1'deki gibidir.

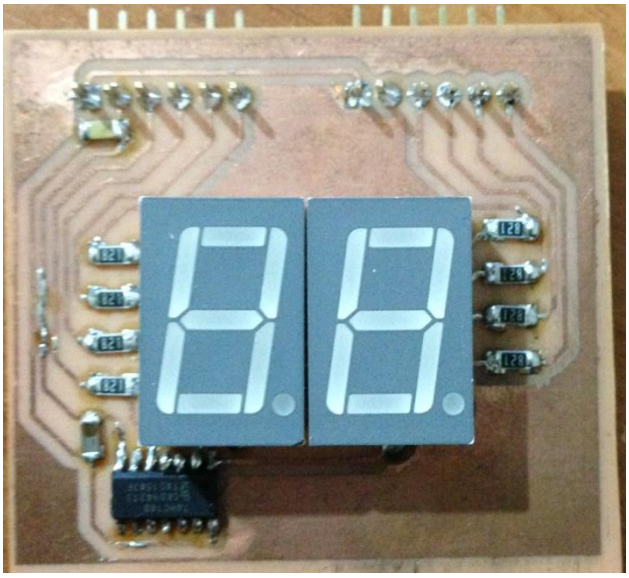


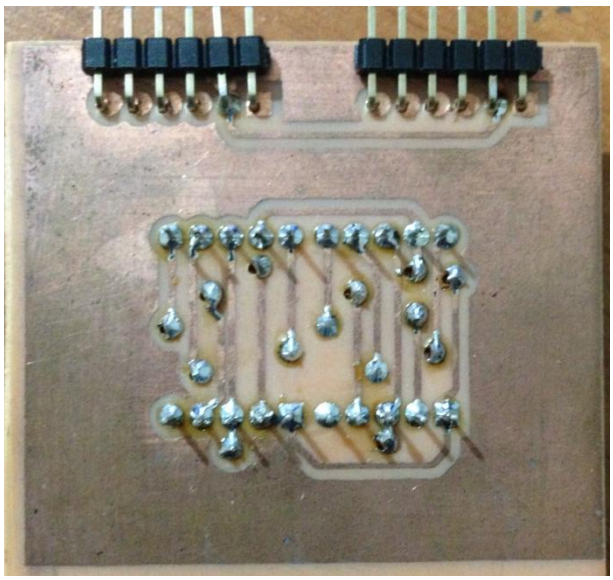
Şekil 5. 1 : Şematik Tasarımı

5.2 PCB Tasarımı

Pcb tasarlanırken çift yüzlü baskı devre yapılması planlanmıştır. Hem top layer hem de bottom layer' da yollar mevcuttur. Top layer için Vcc polygon yapılırken bottom layer' da da GND polygon yapılmıştır. Layer' lar arası geçişlerde via' lar kullanılmıştır. Tasarlanan PCB Şekil 5. 2'deki gibidir.







TOP LAYER

BOTTOM LAYER

Şekil 5. 2 : PCB Tasarımı

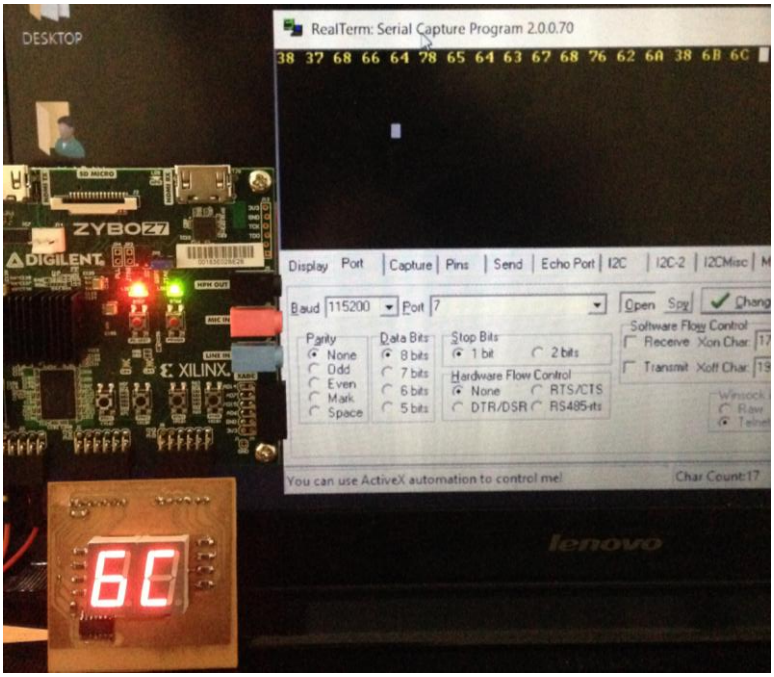
6. TASARIMIN TEST EDİLMESİ

Tasarımın test edilmesi için tüm modüller top_module adında bir modülde birleştirilmiştir ve tasarlanan pmod FPGA kartına takılarak tasarımın test edilmesine geçilmiştir. Pmod için PMOD JB ve PMOD JC portları; Uart_RX sinyali içinde JD portu kulanılmıştır. kullanılmıştır. Clock sinyali olarak da 125 MHz'lik sistem saati kullanımıştır. Pin atamaları Şekil 6.1'de gösterilmiştir.

```
##Pmod Header JB (Zybo Z7-20 only)
set_property -dict { PACKAGE_PIN V8    IOSTANDARD LVCMOS33     } [get_ports { out_Segment_A }]; #IO_L15P_T2_DQS_13 Sch=jb_p[1]
set_property -dict { PACKAGE_PIN W8    IOSTANDARD LVCMOS33     } [get_ports { out_Segment_B }]; #IO_L15N_T2_DQS_13 Sch=jb_n[1]
set_property -dict { PACKAGE_PIN U7    IOSTANDARD LVCMOS33     } [get_ports { out_Segment_C }]; #IO_L11P_T1_SRCC_13 Sch=jb_p[2]
set_property -dict { PACKAGE_PIN V7    IOSTANDARD LVCMOS33     } [get_ports { out_Segment_D }]; #IO_L11N_T1_SRCC_13 Sch=jb_n[2]
#set_property -dict { PACKAGE_PIN Y7    IOSTANDARD LVCMOS33     } [get_ports { jb[4] }]; #IO_L13P_T2_MRCC_13 Sch=jb_p[3]
#set_property -dict { PACKAGE_PIN Y6    IOSTANDARD LVCMOS33     } [get_ports { jb[5] }]; #IO_L13N_T2_MRCC_13 Sch=jb_n[3]
#set_property -dict { PACKAGE_PIN V6    IOSTANDARD LVCMOS33     } [get_ports { jb[6] }]; #IO_L22P_T3_13 Sch=jb_p[4]
#set_property -dict { PACKAGE_PIN W6    IOSTANDARD LVCMOS33     } [get_ports { jb[7] }]; #IO_L22N_T3_13 Sch=jb_n[4]


##Pmod Header JC
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33     } [get_ports { out_Segment_E }]; #IO_L10P_T1_34 Sch=jc_p[1]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33     } [get_ports { out_Segment_F }]; #IO_L10N_T1_34 Sch=jc_n[1]
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33     } [get_ports { out_Segment_G }]; #IO_L1P_T0_34 Sch=jc_p[2]
set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33     } [get_ports { out_CLK_ANODE }]; #IO_L1N_T0_34 Sch=jc_n[2]
#set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33     } [get_ports { jc[4] }]; #IO_L8P_T1_34 Sch=jc_p[3]
#set_property -dict { PACKAGE_PIN Y14   IOSTANDARD LVCMOS33     } [get_ports { jc[5] }]; #IO_L8N_T1_34 Sch=jc_n[3]
#set_property -dict { PACKAGE_PIN T12   IOSTANDARD LVCMOS33     } [get_ports { jc[6] }]; #IO_L2P_T0_34 Sch=jc_p[4]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33     } [get_ports { jc[7] }]; #IO_L2N_T0_34 Sch=jc_n[4]


##Pmod Header JD
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33     } [get_ports { in_UART_RX }]; #IO_L5P_T0_34 Sch=jd_p[1]
```

Şekil 6. 1: Pin konfigürasyonu

Şekil 6. 1' de gösterilen konfigürasyonun yapılmasının ardından bitstream oluşturulup kart programlanmış ve tasarımın çalıştığı gözlemlenmştir. Projenin son hali Şekil 6.2' de gösterilmiştir.



Şekil 6. 2: Proje çıktısı

SONUÇLAR VE ÖNERİLER

Tasarlanan modüllerin simülasyonları gerçekleştirilip gerekli çalışırlılık testleri uygulanmıştır. Uygulama VHDL kodu ile gerçekleştirilmiştir ama kullanılan ZYBO Z7-20 kartında UART IP'si de mevcuttur. Bu projede kullanılmamıştır fakat ilerleyen projelerde kullanabilmek amacıyla Xilinx University Programı kapsamındaki laboratuvar dökümanları ile IP Core kullanımını öğrenmek mümkündür.

KAYNAKLAR

[1] http://www.fpganedir.com/ornek/rs232/receive_mode.php

[2] http://www.fpganedir.com/ornek/seven_segment/vhdl_kod.php

[3] https://reference.digilentinc.com/_media/reference/pmod/pmodssd/pmodssd_sch.pdf

[4] http://www.alldatasheet.com/datasheet-pdf/pdf/27894/TI/SN74HC14DR.html

[5]https://www.snapeda.com/parts/ERA-8AED3011V/Panasonic/view-part/?ref=search&t=ERA-8AED3011V

[6]https://www.snapeda.com/parts/SN74HC14DR/Texas%20Instruments/view-part/?ref=search&t=SN74HC14%20SMD