10. Nexys 4 Uygulamaları

Bu bölümde VHDL dili ile geliştirilen tasarımların Nexys 4 kartı üzerinde gerçeklenmesi ve tasarlanan devrenin davranışı anlatılmıştır.

10.1. D İki Durumlusu

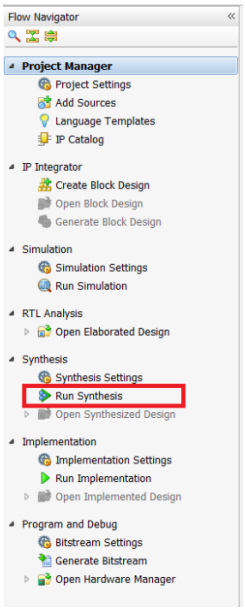
Aşağıda yetkilendirme girişi ve eş zamanlı olmayan resetli d mandali tasarımın yapıldığı **d_mandali.vhd** VHDL kodu verilmiştir. Tasarımda **in_rst** giriş portu değerinin **'1'** olması durumunda diğer giriş portlarının durumu fark etmeksizin **r_cikis** sinyalien **'0'** değeri atanmaktadır. **r_cikis** sinyalinin **'0'** değerini almasıyla **out_cikis** çıkış portuna **'0'** ve **out_cikis_degil** çıkış portna **'1'** değerleri atanmaktadır. **in_rst** giriş portunun diğer durumlarında ise **in_clk** giriş portunun yükselen kenarının meydana gelmesi beklenmektedir. Yükselen kenarın meydan gelmesi ile birlikte **in_en** giriş portu değerinin **'1'** olması durumunda **in_giriş** giriş portu değeri **r_cikis** sinyaline atandır. **in_en** giriş portunun diğer durumlarında ise **r_cikis** sinyali bir önceki durumunu korumaktadır.

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

entity d_mandali is
  Port (
    in_clk : in std_logic;
    in_rst : in std_logic;
    in_en : in std_logic;
    in_giris : in std_logic;
    out_cikis : out std_logic;
    out_cikis_degil : out std_logic
  );
end d_mandali;

architecture Behavioral of d_mandali is

  signal r_cikis : std_logic := '0';

begin

  process(in_clk, in_rst, in_en, in_giris)
  begin
    if in_rst = '1' then
      r_cikis <= '0';
```

```
    elsif rising_edge(in_clk) then
      if in_en = '1' then
        r_cikis <= in_giris;
      end if;
    end if;
  end process;

  out_cikis <= r_cikis;
  out_cikis_degil <= not r_cikis;

end Behavioral;
```

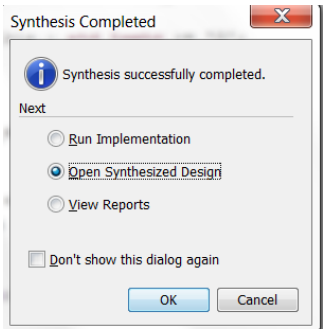
10.1.1. Sentezleme ve Port Bağlantılarının Yapılması

Tasarlanan **d_mandali** varlığının Nexsy 4 kartı ile bağlantılarının yapılabilemesi için öncelikle sentez işlemini yapılması gerekmektedir. Şekil 10-1'den de görüleceği üzere sntezleme işlemi **Synthesis** sekmesi altında bulunan **Run Synthesis** seçilerek başlatılır.

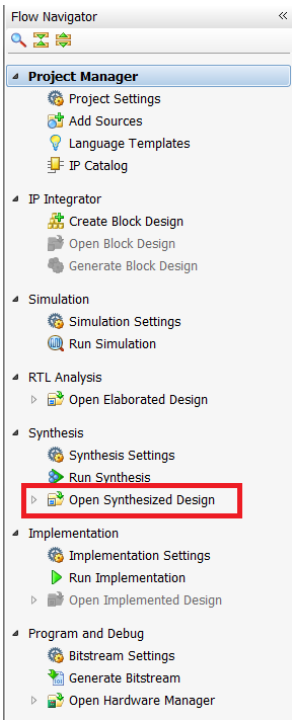


Şekil 10-1 Sentezleme işleminin başlatılması

Sentezleme işleminin bitiminde Şekil 10-2'de gösterilen pencere açılmaktadır. Açılan pencereden **Open Synthesis Design** seçilir ve **OK** tuşuna basılarak sentez sonrası port bağlantısı, hata ayıklama bağlantıları vb. gibi işlemler yapılabilmektedir. **Open Sythesis Design** aynı zamanda sentez işleminin sonucunda **Flow Navigator** penceresinde aktif olmaktadır (Şekil 10-3).

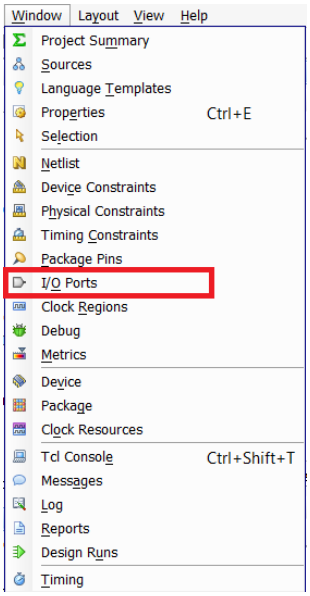


Şekil 10-2 Sentezleme işleminin tamamlanması ve sentez tasarımlarının açılması

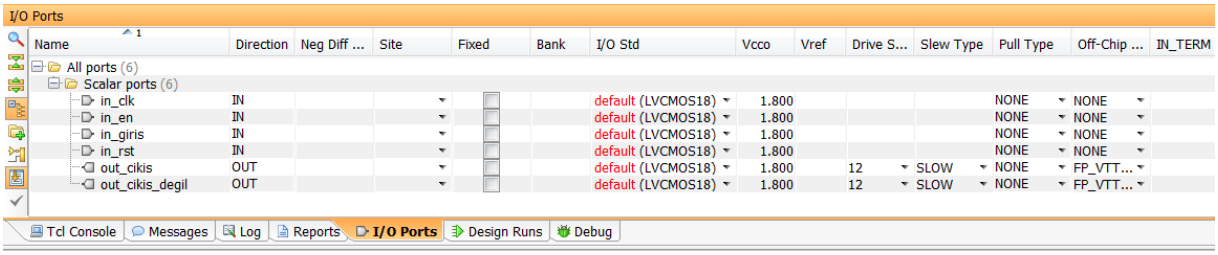


Şekil 10-3 Sentez tasarımlarının açılması

Sentez tasarımlarının açılmasından sonra **`Window`** sekmesi seçilir. **`Window`** sekmesi altında **`I/O Ports`** seçilir (Şekil 10-4). İşlemin ardından **`d_latch`** varlığının Nexys 4 kartı ile bağlantısının yapılması için **`I/O Ports`** penceresi açılacaktır. Açılan pencerede **`d_latch`** varlığın ait portlar görülecektir (Şekil 10-5).



Şekil 10-4 I/O Ports Pencersinin açılması - 1



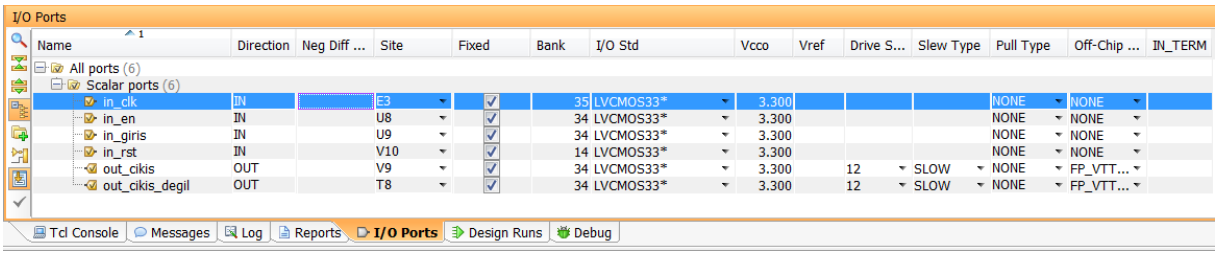
Şekil 10-5 I/O Ports Pencersinin açılması - 2

d_latch varlığın da daha önceden de bahsedildiği gibi 2 adet giriş portu ve 2 adet çıkış portu mevcuttur. Varlığın portlarının Nexys 4 kartı ile bağlantılası Tablo 10-1'deki şekilde yapılmalıdır.

Tablo 10-1 Yetki girişli D mandalı bağlantıları

Port Adı	Nexys 4	Konum	I/O Standart
in_clk	**-**	**E3**	LVCMOS33
in_en	**SW1**	**U8**	LVCMOS33
in_rst	**BTND**	**V10**	LVCMOS33
in_giris	**SW0**	**U9**	LVCMOS33
out_cikis	**LD1**	**V9**	LVCMOS33
out_cikis_degil	**LD0**	**T8**	LVCMOS33

Tablo 10-1'de verilen konum bilgileri **I/O Ports** pencersinde **Site** sekmesinde seçlirler. **I/O Std** sekmesinde ise Tablo 10-1'de verilen standart değerleri seçilerek (Şekil 10-6) kaydedilir.



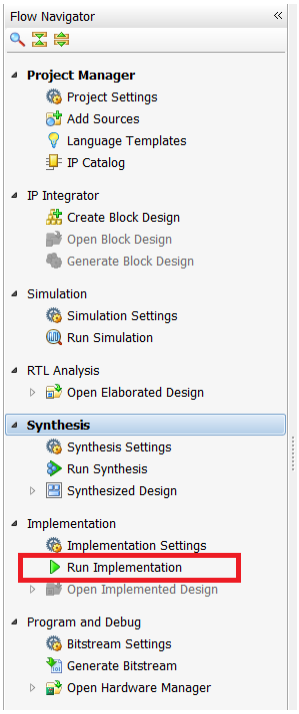
Şekil 10-6 Port bağlantılarının ayarlanması

Port tanımlama işlemlerinin yapılmasından sonra kaydedilen **xdc** uzantılı dosya aşağıdaki gibi olacaktır.

```
set_property PACKAGE_PIN E3 [get_ports in_clk]
set_property IOSTANDARD LVCMOS33 [get_ports in_clk]

set_property PACKAGE_PIN U8 [get_ports in_en]
set_property IOSTANDARD LVCMOS33 [get_ports in_en]

set_property PACKAGE_PIN U9 [get_ports in_giris]
set_property IOSTANDARD LVCMOS33 [get_ports in_giris]

set_property PACKAGE_PIN V10 [get_ports in_rst]
set_property IOSTANDARD LVCMOS33 [get_ports in_rst]

set_property PACKAGE_PIN V9 [get_ports out_cikis]
set_property IOSTANDARD LVCMOS33 [get_ports out_cikis]

set_property PACKAGE_PIN T8 [get_ports out_cikis_degil]
set_property IOSTANDARD LVCMOS33 [get_ports out_cikis_degil]
```

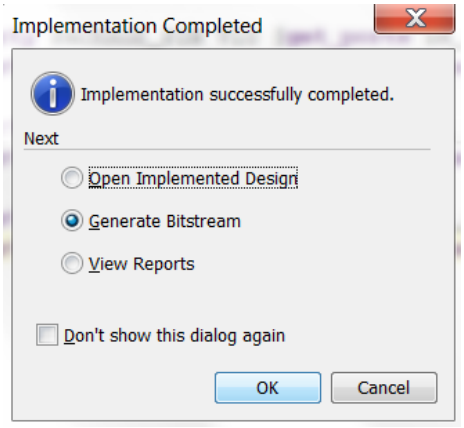
10.1.2. Bit Dosyasının Oluşturulması

Port tanımlama işlemlerinin tamamlanmasından sonra gerçekleme işlemi Implementation Sekmesi altında **Run Implmentation** seçilerek başlatılır (Şekil 10-7).

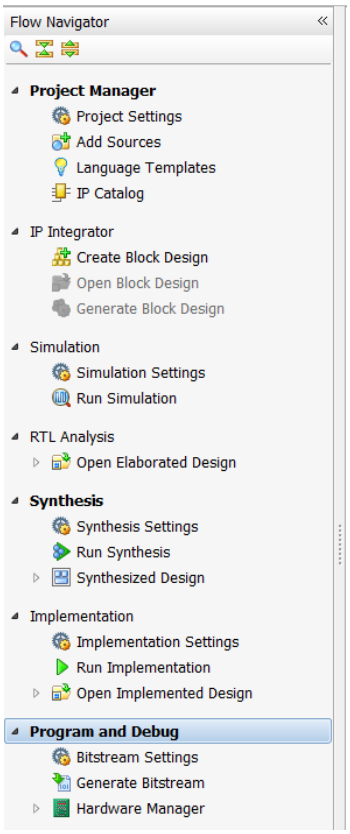


Şekil 10-7 Gerçekleme işleminin başlatılması

Gerçekleme işleminin bitiminde Şekil 10-8'de gösterilen pencere açılmaktadır. Açılan pencereden **`Generate Bitstream`** seçilir ve **`OK`** tuşuna basılarak bit dosyasının oluşturulma işlemi başlatılmaktadır. Bit dosyası oluşturma işlemi aynı zamanda **`Program and Debug`** sekmesi altında **`Generate Bitstream`** seçilerekte yapılabilmektedir. **`Generate Bitstream`** gerçekleme işleminin sonucunda **`Flow Navigator`** penceresinde aktif olmaktadır (Şekil 10-9).



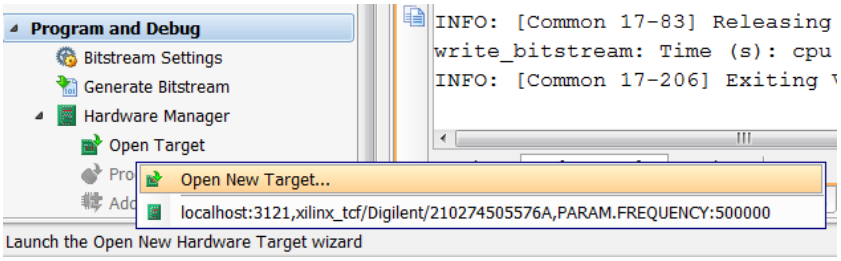
Şekil 10-8 Gerçekleme işleminin tamamlanması



Şekil 10-9 Bit dosyası oluşturma işleminin başlatılması

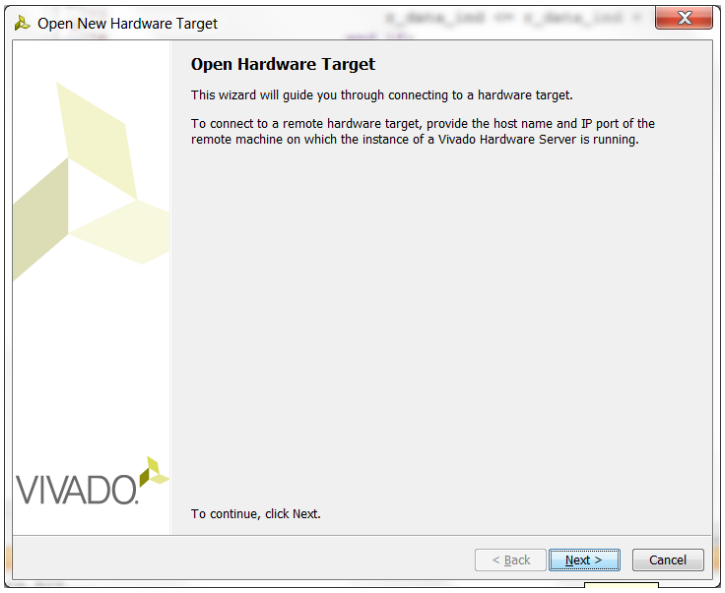
10.1.3. Bit Dosyasının Yüklenmesi

Bit uzantılı dosyanın oluşturulmasından sonra FPGA'ya gerçeklenen devrenin yüklenmesi aşaması başlamaktadır. Şekil 10-10'dan da görüleceği üzere, **`Program and Debug`** sekemsi altında bulunan **`Hardware Manager`** sekmesinde **`Open Target`** seçilir. Eğer daha önce kullandığınız FPGA ile bağlantı sağlamış iseniz o bağlantıyı seçerek işleme devam edebilirsiniz. Daha önce çalışma yapılmadıysa **`Open New Target`** seçilir.



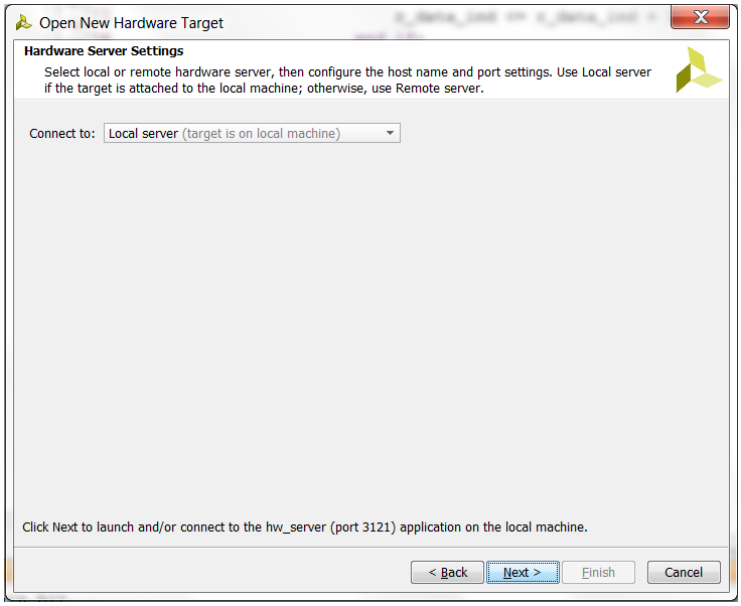
Şekil 10-10 Kullanılacak FPGA ile bağlantı salanması - 1

Açılan pencerede **`Next`** butonu seçilir (Şekil 10-11).



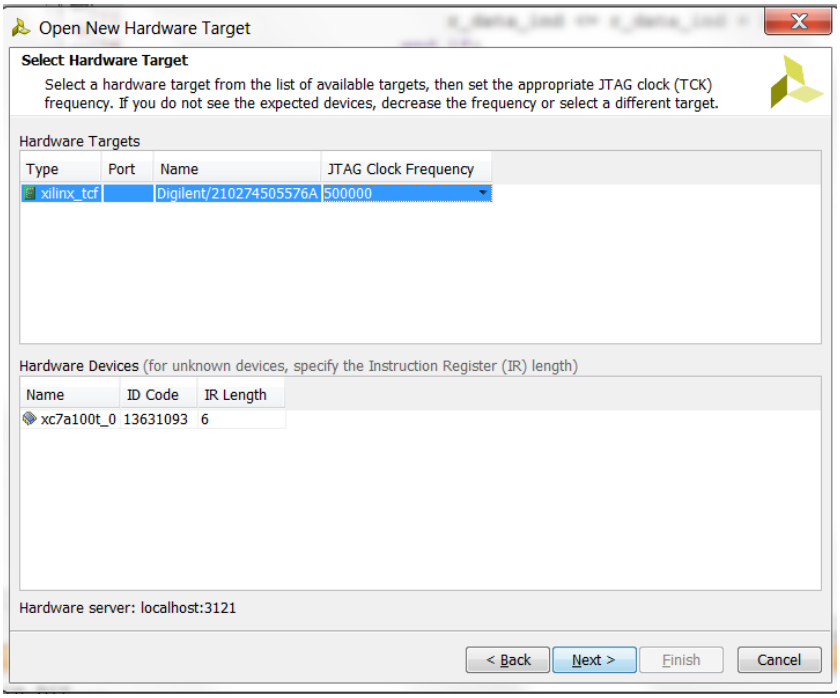
Şekil 10-11 Kullanılacak FPGA ile bağlantı salanması -2

Açılan pencerede **`Next`** butonu seçilir (Şekil 10-12).



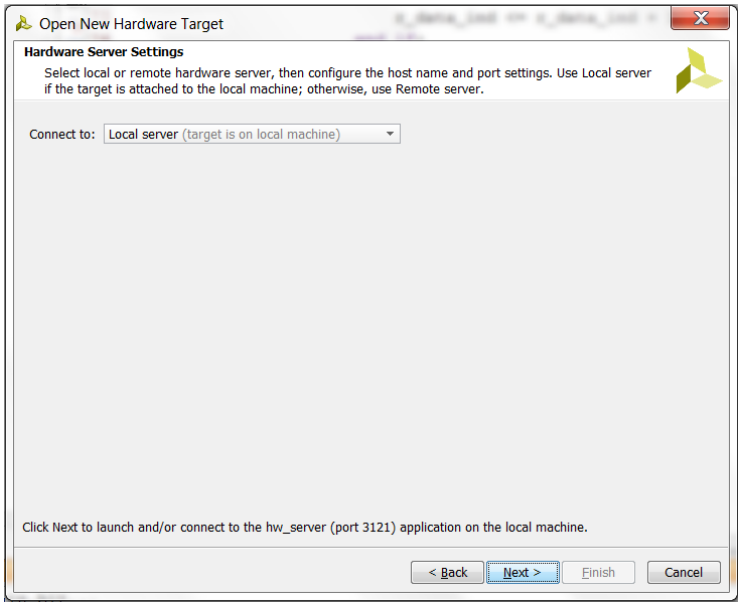
Şekil 10-12 Kullanılacak FPGA ile bağlantı salanması -3

Şekil 10-13'den de görüleceği üzere **`Hardware Device`** kısmında Nexys 4 kart üzerinde bulunan FPGA tanımlıdır. **`Hardware Target`** sekmesinde bulunan **`JTAG Clock Frequency`** kısmında düşük hız seçilmelidir. 500 KHz yeterli bir hız olmaktadır.



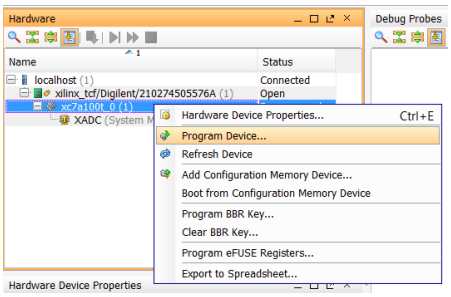
Şekil 10-13 Kullanılacak FPGA ile bağlantı salanması -4

Açılan pencerede **`Finish`** butonu seçilir (Şekil 10-14).



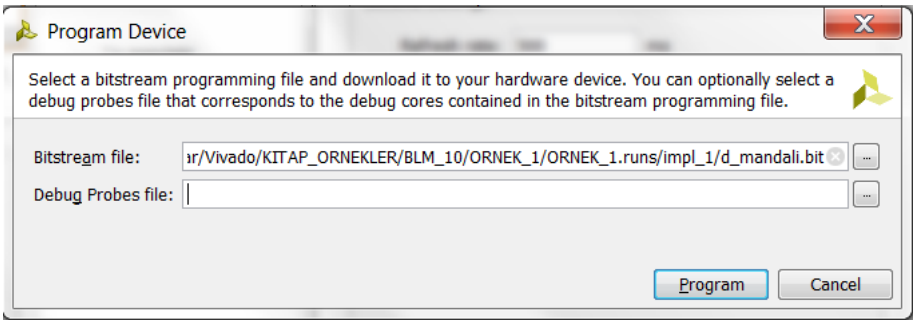
Şekil 10-14 Kullanılacak FPGA ile bağlantı salanması -5

Bağlantı işleminin Vivado ekranında açılan **`Hardware`** penceresinden tanımlı FPGA üzerine tıklanarak **`Program Device`** seçilir (Şekil 10-15).



Şekil 10-15 Devrenin FPGA'ya yüklenmesi - 1

Açılan **Program Device** penceresinde **Program** butonuna basılarak FPGA programlama işlemi başlatılır (Şekil 10-16).



Şekil 10-16 Devrenin FPGA'ya yüklenmesi - 2

10.1.4. Tasarımın Test Edilmesi

Tasarımın test edilmesi işleminda aşağıdaki adımlar gerçeklenir:

- **in_en** giriş portunun bağlı bulunduğu anahtar **'1'** konumuna, **in_giris** portunun bağlı bulunduğu anahtar **'0'** konumuna getirilir. Bu durumda **out_cikis_degil** çıkış portunun bağlı bulunduğu led yanacaktır.
- **in_en** giriş portunun bağlı bulunduğu anahtar **'0'** konumuna getirilir. Bu durumda **out_cikis_degil** çıkış portunun bağlı bulunduğu led yanacaktır.
- **in_giris** portunun bağlı bulunduğu anahtar **'1'** konumuna getirilir. Bu durumda **out_cikis_degil** çıkış portunun bağlı bulunduğu led yanacaktır.
- **in_en** giriş portunun bağlı bulunduğu anahtar **'1'** konumuna getirilir. Bu durumda **out_cikis** çıkış portunun bağlı bulunduğu led yanacaktır.
- **in_rst** giriş portunun bağlı bulunduğu tuşa basılı tutulunur. Bu durumda **out_cikis_degil** çıkış portunun bağlı bulunduğu led yanacaktır.
- **in_rst** giriş portunun bağlı bulunduğu tuşa bassma işlemi bırakılır. Bu durumda **out_cikis** çıkış portunun bağlı bulunduğu led yanacaktır.

10.2. Led Yakma Uygulaması

Led yakma uygulamasında Bölüm 9.8'de verilen **saat_frekans_bolucu.vhd** VHDL kodu kullanılmıştır. Tasarımda 100 MHz olan sistem frekansı, **saat_frekans_bolucu1_map** etiketi ile tanımlanmış alt devre ile 1 Hz'e indirilmektedir. Bu işlem için generic olarak tanımlanan N parametresi değeri 100000000 olarak ayarlanmıştır. Her iki alt modülün çalışma frekanslarının farklı olması nedeniyle elde edilen 1 Hz'lik sinyal 63-

70. satırlarda tanımlı **process** içerisinde saat darbesi domain geçiş işlemine tabi tutulmaktadır. Geçiş işleminin sağlanması ile elde edilen yeni saat darbesi işareti, **saat_frekans_bolucu2_map** etiketi ile tanımlanmış alt devreye saat darbesi girişi olarak verilmektedir. **saat_frekans_bolucu2_map** etiketi ile tanımlanmış alt devre ile 1/2 Hz,1/4 Hz, 1/8 Hz, 1/16 Hz ve **generic** olarak ayarlanan 1/32 Hz frekanslarında saat darbeleri elde edilmiştir.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity led_yakma is
5.   Port (
6.     in_clk : in std_logic;
7.     in_rst : in std_logic;
8.     out_clk_1Hz : out std_logic;
9.     out_clk_1_2Hz : out std_logic;
10.    out_clk_1_4Hz : out std_logic;
11.    out_clk_1_8Hz : out std_logic;
12.    out_clk_1_16Hz : out std_logic;
13.    out_clk_1_NHz : out std_logic
14.  );
15.end led_yakma;
16.
17.architecture Behavioral of led_yakma is
18.
19.  component saat_frekans_bolucu
20.  generic(
21.    N : integer := 16
22.  );
23.  Port (
24.    in_clk : in std_logic;
25.    in_rst : in std_logic;
26.    out_clk_2 : out std_logic;
27.    out_clk_4 : out std_logic;
28.    out_clk_8 : out std_logic;
29.    out_clk_16 : out std_logic;
30.    out_clk_N : out std_logic
31.  );
32.end component;
33.
34.  signal r_clk_1Hz_d : std_logic := '0';
35.  signal r_clk_1Hz : std_logic_vector(3 downto 0) := (others => '0');
```

```
36.   signal r_clk_1_2Hz : std_logic := '0';
37.   signal r_clk_1_4Hz : std_logic := '0';
38.   signal r_clk_1_8Hz : std_logic := '0';
39.   signal r_clk_1_16Hz : std_logic := '0';
40.   signal r_clk_1_NHz : std_logic := '0';
41.
42.begin
43.
44.   out_clk_1Hz <= r_clk_1Hz(3);
45.   out_clk_1_2Hz <= r_clk_1_2Hz;
46.   out_clk_1_4Hz <= r_clk_1_4Hz;
47.   out_clk_1_8Hz <= r_clk_1_8Hz;
48.   out_clk_1_16Hz <= r_clk_1_16Hz;
49.   out_clk_1_NHz <= r_clk_1_NHz;
50.
51.   saat_frekans_bolucu1_map : saat_frekans_bolucu
52.   generic map( N => 100000000  )
53.   port map (
54.     in_clk => in_clk,
55.     in_rst => in_rst,
56.     out_clk_2 => open,
57.     out_clk_4 => open,
58.     out_clk_8 => open,
59.     out_clk_16 => open,
60.     out_clk_N => r_clk_1Hz_d
61.   );
62.
63.   process(in_clk, in_rst)
64.   begin
65.     if in_rst = '1' then
66.       r_clk_1Hz <= (others => '0');
67.     elsif rising_edge(in_clk) then
68.       r_clk_1Hz <= r_clk_1Hz(2 downto 0) & r_clk_1Hz_d;
69.     end if;
70.   end processS;
71.
72.   saat_frekans_bolucu2_map : saat_frekans_bolucu
73.   generic map( N => 32  )
74.   port map (
75.     in_clk => r_clk_1Hz(3),
```

```
76.     in_rst => in_rst,
77.     out_clk_2 => r_clk_1_2Hz,
78.     out_clk_4 => r_clk_1_4Hz,
79.     out_clk_8 => r_clk_1_8Hz,
80.     out_clk_16 => r_clk_1_16Hz,
81.     out_clk_N => r_clk_1_NHz
82.   );
83.
84.end Behavioral;
```

Sistemde elde edilen yeni saat darbesi sinyallerinin Nexys 4 kartında görülebilmesi için devre sentezlendikten sonra Tablo 10-2'de verilen konum bilgileri ile bağlantıların yapılması gerekmektedir.

Tablo 10-2 Led yakma varlığına ilişkin port bağlantıları

Port Adı	Nexys 4	Konum	I/O Standart
in_clk	**-**	**E3**	LVCMOS33
in_rst	**BTND**	**V10**	LVCMOS33
out_clk_1Hz	**LD0**	**T8**	LVCMOS33
out_clk_1_2Hz	**LD1**	**V9**	LVCMOS33
out_clk_1_4Hz	**LD2**	**R8**	LVCMOS33
out_clk_1_8Hz	**LD3**	**T6**	LVCMOS33
out_clk_1_16Hz	**LD4**	**T5**	LVCMOS33
out_clk_1_NHz	**LD5**	**T4**	LVCMOS33

Tablo 10-2'de verilen konum bilgileri ile yapılan bağlantılardan sonra kaydedilen **xdc** uzantılı dosya aşağıdaki gibi olacaktır.

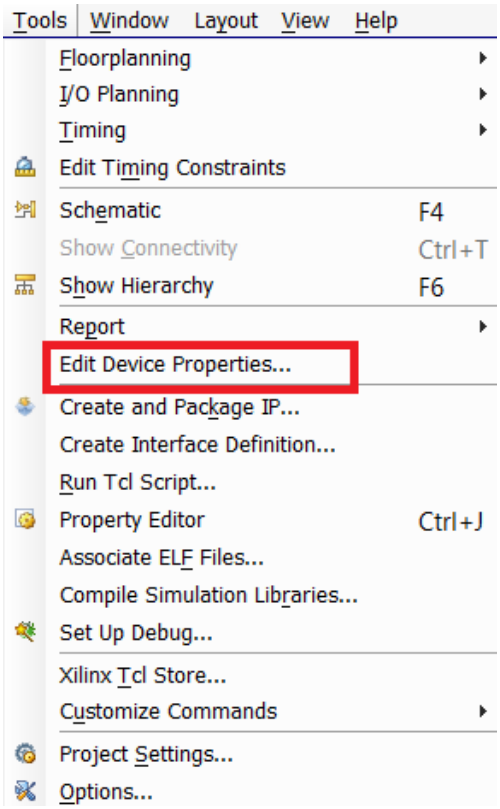
```
set_property PACKAGE_PIN E3 [get_ports in_clk]
set_property IOSTANDARD LVCMOS33 [get_ports in_clk]
set_property PACKAGE_PIN V10 [get_ports in_rst]
set_property IOSTANDARD LVCMOS33 [get_ports in_rst]
set_property PACKAGE_PIN T8 [get_ports out_clk_1Hz]
set_property IOSTANDARD LVCMOS33 [get_ports out_clk_1Hz]

set_property PACKAGE_PIN V9 [get_ports out_clk_1_2Hz]
set_property IOSTANDARD LVCMOS33 [get_ports out_clk_1_2Hz]
set_property PACKAGE_PIN R8 [get_ports out_clk_1_4Hz]
set_property IOSTANDARD LVCMOS33 [get_ports out_clk_1_8Hz]
set_property PACKAGE_PIN T6 [get_ports out_clk_1_8Hz]
set_property IOSTANDARD LVCMOS33 [get_ports out_clk_1_4Hz]
set_property PACKAGE_PIN T5 [get_ports out_clk_1_16Hz]
set_property IOSTANDARD LVCMOS33 [get_ports out_clk_1_16Hz]
```

```
set_property PACKAGE_PIN T4 [get_ports out_clk_1_NHz]
set_property IOSTANDARD LVCMOS33 [get_ports out_clk_1_NHz]
```

10.2.1. Flash Dosyası Oluşturma

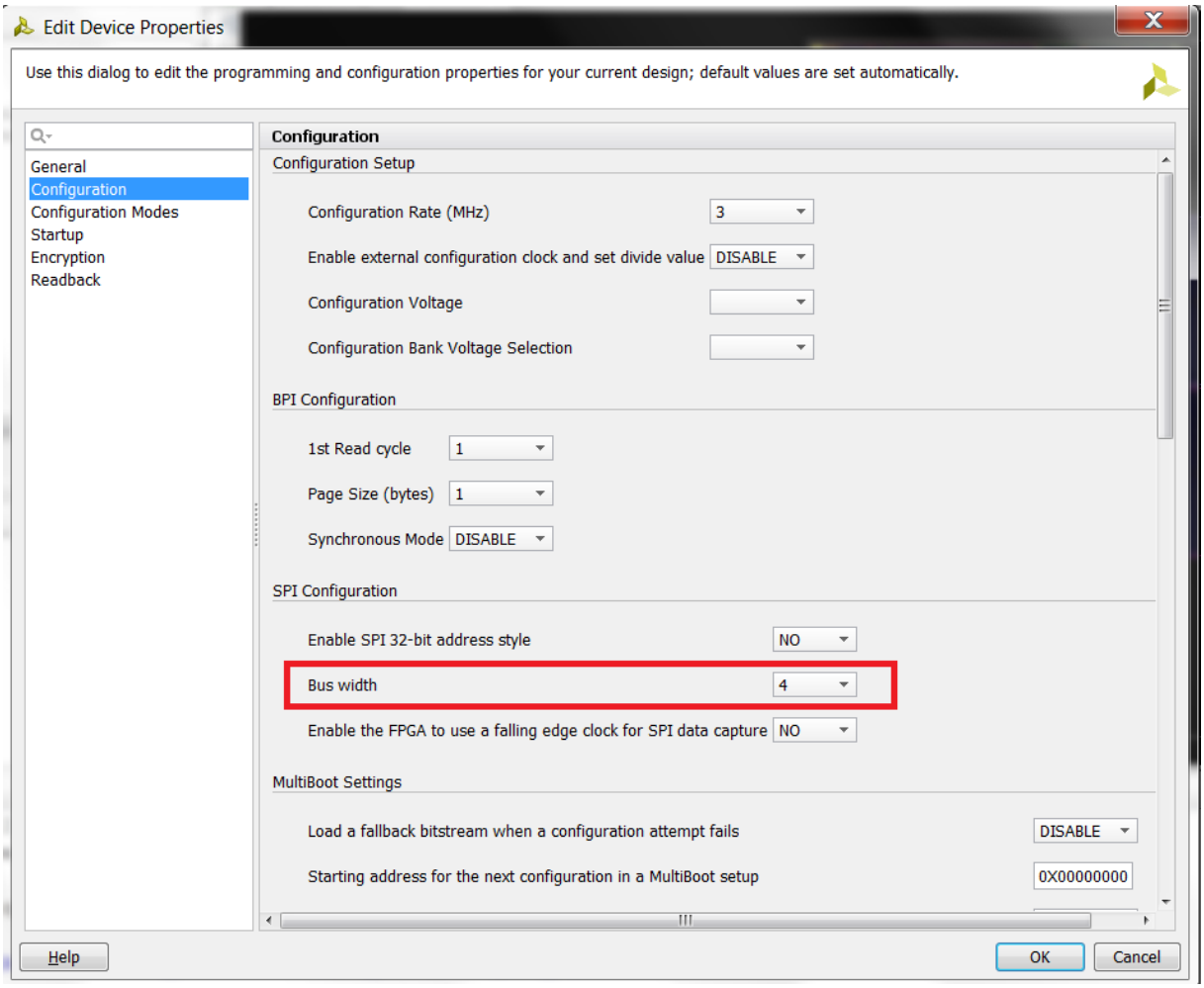
Neyxs 4 kartı üzerinde bulunan flash 4x SPI veri yoluna ve 128 Mb hafızaya sahiptir. Parça numarası **S25FL128S**'dir. Flash ayarlamalarını kendi projemiz içerisinde de yapmamız gerekmektedir.

Dosya oluşturma işleminden önce oluşturulan bit uznatılı dosya ile flashın veri yolu ayarları aynı olmalıdır. Ayarlama işlemi sentezleme ve bağlantı işlemlerinin bitimiyle **Tools** sekmesinden **Edit Device Properties** seçilir (Şekil 10-17).



Şekil 10-17 Flash ayarlarının yapılması - 1

Açılan pencereden **Configuration** seçilir ve **Bus width** 4 olarak ayarlanarak **OK** tuşuna basılır (Şekil 10-18).



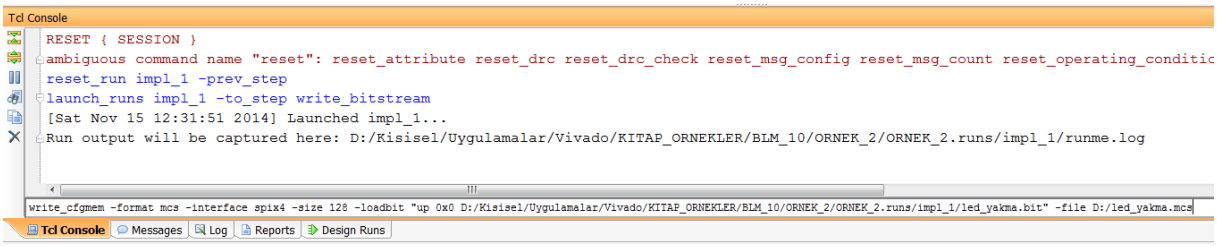
Şekil 10-18 Flash ayarlarının yapılması - 2

Bu işlemlerin ardından bit uzantılı dosya oluşturulur. Oluşturulan bit uznatılı dosyanın flasha yazmak için oluşturulacak dosyaya çevirme işlemi aşağıda verilen kod dizini TCL Console bölümüne yazılmasıyla başlatılır (Şekil 10-19).

```
write_cfgmem -format mcs -interface spix4 -size 128 -loadbit "up 0x0
VERI_YOLU_BIT " -file VERI_YOLU_MCS
```

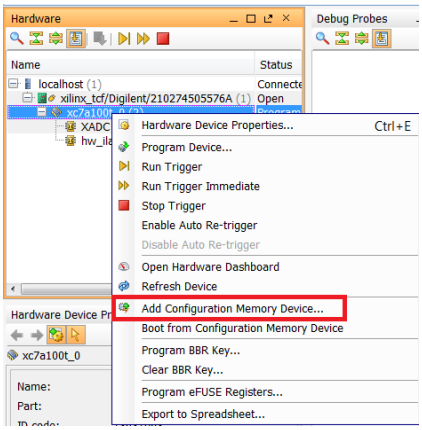
Örnek 10.2 için oluşturulan **led_yakma.bit** uzantılı dosyadan flasha yazmak için **led_yakma.mcs** uznatılı dosya elde etmek için aşağıdaki kod satırı TCL Console kısmına yazılır.

```
write_cfgmem -format mcs -interface spix4 -size 128 -loadbit "up 0x0
D:/Kisisel/Uygulamalar/Vivado/KITAP_ORNEKLER/BLM_10/ORNEK_2/ORNEK_2.runs/im
pl_1/led_yakma.bit" -file D:/led_yakma.mcs
```



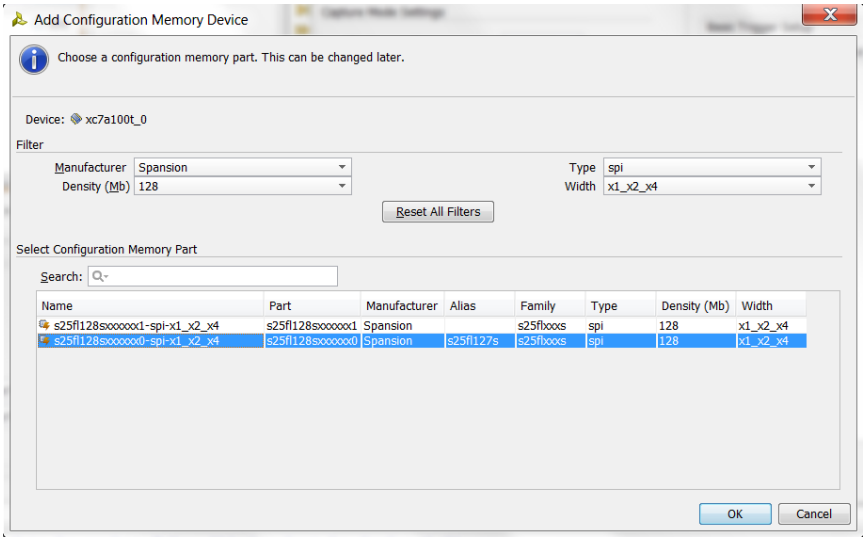
Şekil 10-19 led_yakma.mcs dosyasının oluşturulması

Bu işlemlerin ardından Nexys 4 kartımızda bulunan FPGA ile bağlantının yapılmalıdır. İşlemlerin ardından **`Hardware`** penceresinden tanımlı FPGA üzerine tıklanarak **`Add Configuration Memory Device`** seçilir (Şekil 10-20).



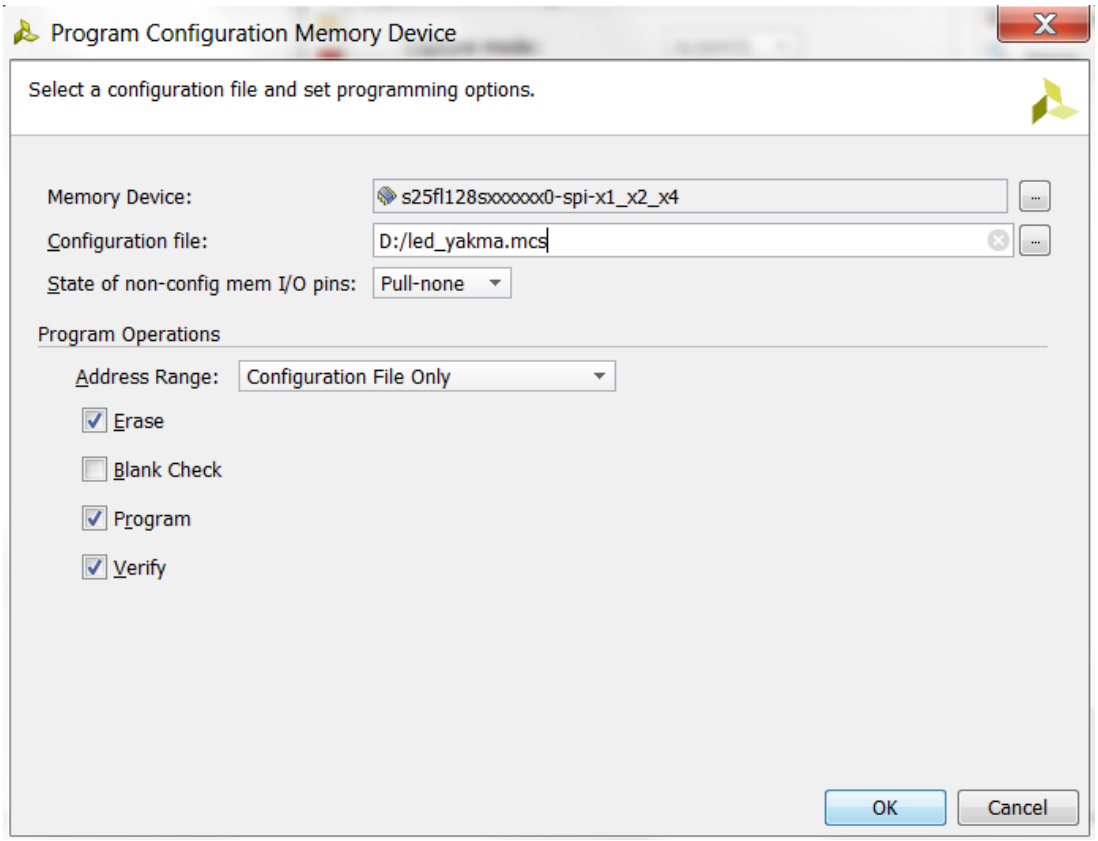
Şekil 10-20 Flash dosyasının yüklenmesi - 1

Açılan pencereden kartımızın üzerinde bulunan Flash seçilir ve OK tuşuna basılır (Şekil 10-21).



Şekil 10-21 Flash dosyasının yüklenmesi - 2

Daha sonra açılan pencereden **`Configuration file`** kısmına oluşturduğumuz **`led_yakma.mcs`** dosyası tanımlanır ve **`OK`** tuşuna basılarak flash programlaam işlemi başlamaktadır (Şekil 10-22).



Şekil 10-22 Flash dosyasının yüklenmesi - 3

Programlama işleminin bitmesiyle artık kartımızı her açtığımızda **led_yakma.vhd** VHDL kodu ile gerçekleştirdiğimiz devre aktif olacaktır.

10.3. Vivado ile Hata Ayıklama (Debug) Uygulaması

Aşğaıda verilen display.vhd VHDL kod ile tanımlı 4 bitlik giriş portunun aldığı değeri 7 Segment Displaylerde gösterilmektedir. Aynı zamanda bu kod ile Vivado programında sinyal değişlenlerinin aldığı değerleri kontrol edilebilmektedir. 26-33. satırlarda tanımlı nitelikler (**attribitude**) ile kod FPGA içerisinde çalışırken tanımlı sinyallerin aldığı değerleri görebilmemiz sağlanacaktır. 26-27. satırlarda tanımlı nitelikle kod içerisinde kesinlikle bulunması gerekmektedir. 29-30 ve 32-33. satırlarda tanımlı ifadelerde ise **r_giris** ve **r_cikis** sinyallerinin aldığı değerlerin izlenceği tanımlanmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use IEEE.STD_LOGIC_UNSIGNED.ALL;
4.
5. entity display is
6.     Port (
7.         in_clk : in std_logic;
```

```
        in_rst : in std_logic;
        in_giris : in std_logic_vector(3 downto 0);
        out_disp_sec : out std_logic_vector(7 downto 0);
        out_cikis : out std_logic_vector(7 downto 0)
    );
end display;

architecture Behavioral of display is

    type t_display_ekran is array (0 to 15) of std_logic_vector(7 downto 0);
    constant DISP_EKRAN : t_display_ekran := ("10000001", "11001111", "10010010",
"10000110",    "11001100",    "10100100",    "10100000",    "10001111", "10000000",
"10000100",    "10001000",    "11100000",    "10110001",    "11000010", "10110000",
"10111000");

    signal r_giris : std_logic_vector(3 downto 0) := (others => '0');
    signal r_cikis : std_logic_vector(7 downto 0) := (others => '0');

    attribute syn_keep : string;
    attribute mark_debug : string;

    attribute syn_keep of r_giris : signal is "true";
    attribute mark_debug of r_giris : signal is "true";

    attribute syn_keep of r_cikis : signal is "true";
    attribute mark_debug of r_cikis : signal is "true";

begin

    out_disp_sec <= "00000000";
    out_cikis <= r_cikis;

    process(in_clk, in_rst, in_giris)
    begin
        if in_rst = '1' then
            r_giris <= (others => '0');
            r_cikis <= (others => '0');
        elsif rising_edge(in_clk) then
```

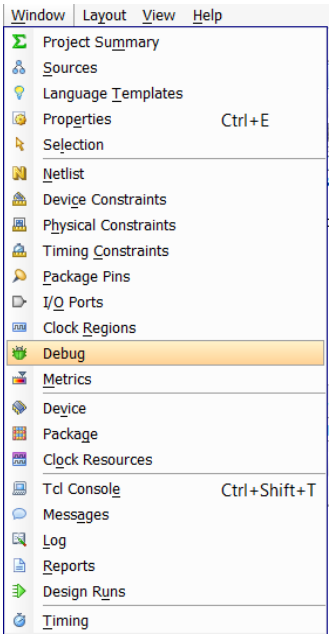
```
46.             r_giris <= in_giris;
47.             r_cikis <= DISP_EKRAN(conv_integer(r_giris));
48.         end if;
49.     end process;
50.end Behavioral;
```

Uygulamaya ilişkin pin atama işlemlerinin yapıldığı varsayımı ile anlatıma devam edilecektir. 7 Segment Display modülünün Nexys 4 kartında çalışabilmesi için gerekli port konumları ve port standarları Tablo 10-3'de verilmiştir.

Tablo 10-3 Display modülünün Nexys 4 kartında bağlantıları

Port	Konum	I/O Standart
in_clk	E3	LVCMOS33
in_rst	V10	LVCMOS33
in_giris(0)	U9	LVCMOS33
in_giris(1)	U8	LVCMOS33
in_giris(2)	R7	LVCMOS33
in_giris(3)	R6	LVCMOS33
out_disp_sec(0)	N6	LVCMOS33
out_disp_sec(1)	M6	LVCMOS33
out_disp_sec(2)	M3	LVCMOS33
out_disp_sec(3)	N5	LVCMOS33
out_disp_sec(4)	N2	LVCMOS33
out_disp_sec(5)	N4	LVCMOS33
out_disp_sec(6)	L1	LVCMOS33
out_disp_sec(7)	M1	LVCMOS33
out_cikis(0)	L6	LVCMOS33
out_cikis(1)	M2	LVCMOS33
out_cikis(2)	K3	LVCMOS33
out_cikis(3)	L4	LVCMOS33
out_cikis(4)	L5	LVCMOS33
out_cikis(5)	N1	LVCMOS33
out_cikis(6)	L3	LVCMOS33
out_cikis(7)	M4	LVCMOS33

Sentezleme ve port tanımlama işlemleri yapıldıktan sonra **Window** sekmesi altında bulunan **Debug** seçilir (Şekil 10-23).



Şekil 10-23 Hata ayıklama işlemleri – 1

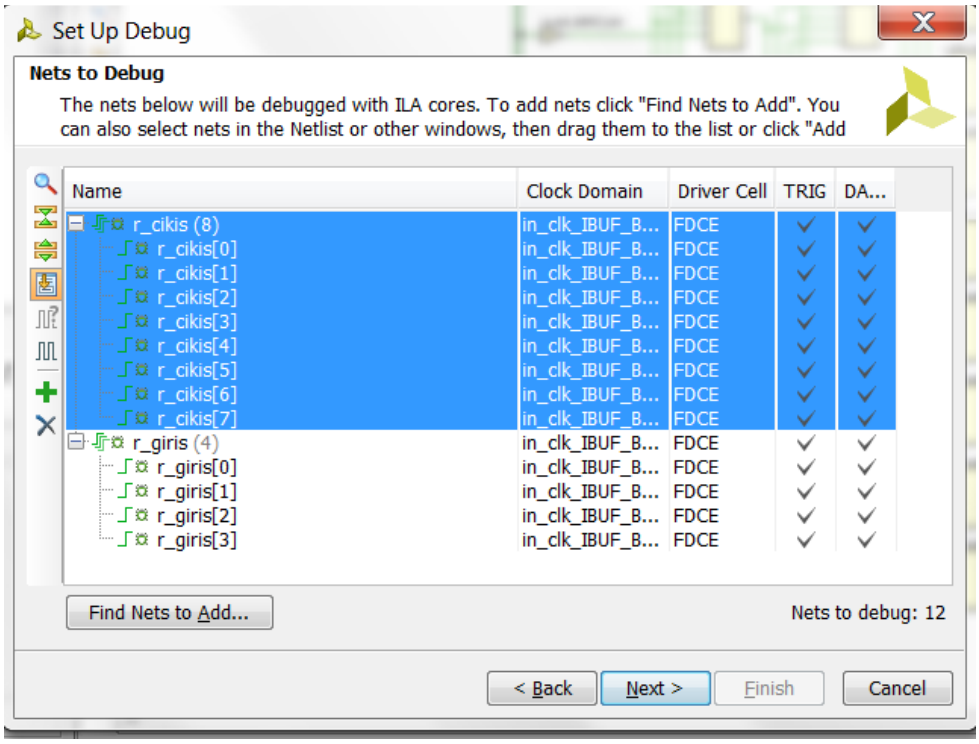
Açılan **Debug** penceresinde bulunan **Set Up Debug** pencersi seçilir (Şekil 10-24).

Şekil 10-24 Hata ayıklama işlemleri - 2

Açılan pencerede **Next** butonuna basılır (Şekil 10-25).

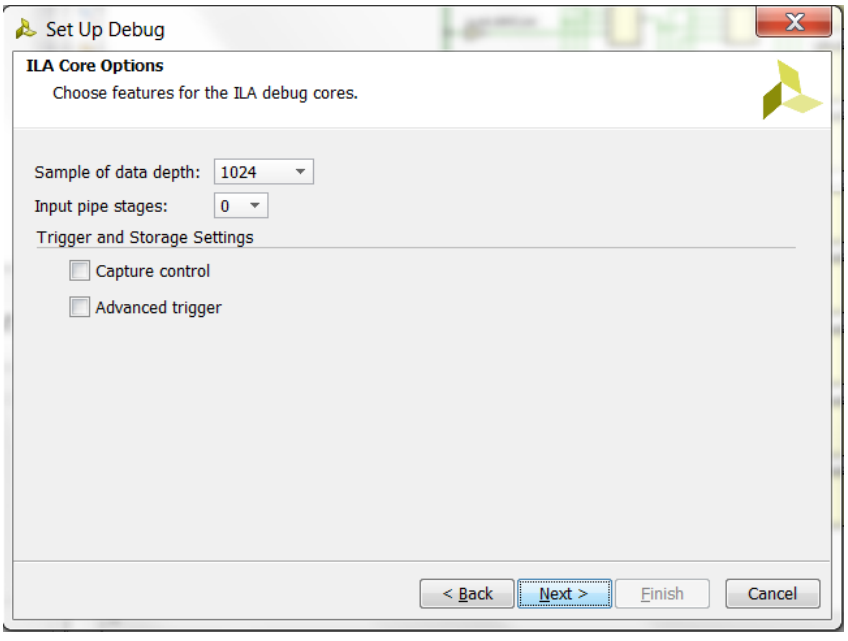
Şekil 10-25 Hata ayıklama işlemleri - 3

Niteliklerde tanımlı **`r_cikis`** ve **`r_giris`** sinyallerinin listelendiği görülmektedir. **`Next`** butonuna basılarak bir sonraki aşamaya geçilir (Şekil 10-26).



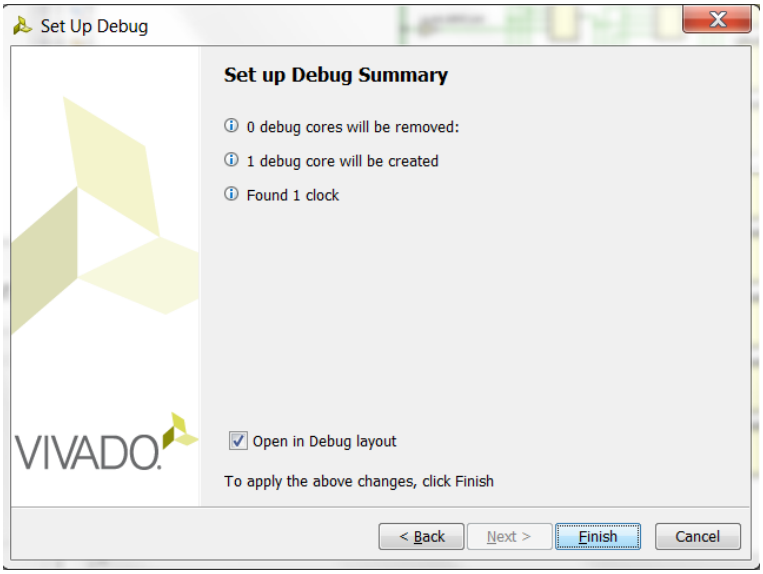
Şekil 10-26 Hata ayıklama işlemleri - 4

`Next` butonuna basılarak bir sonraki aşamaya geçilir (Şekil 10-27Şekil 10-26).



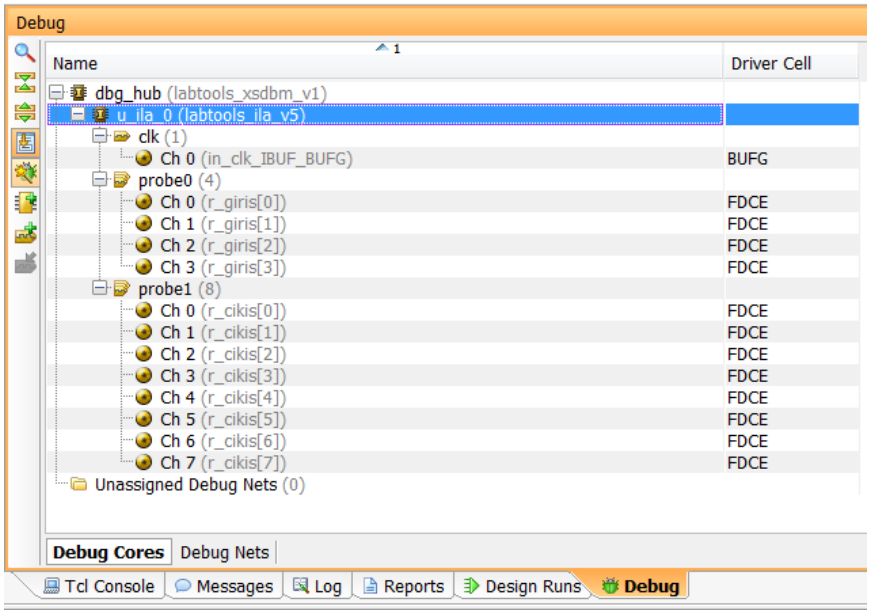
Şekil 10-27 Hata ayıklama işlemleri - 5

Finish butonuna basılarak işlem sonlandırılır (Şekil 10-28).



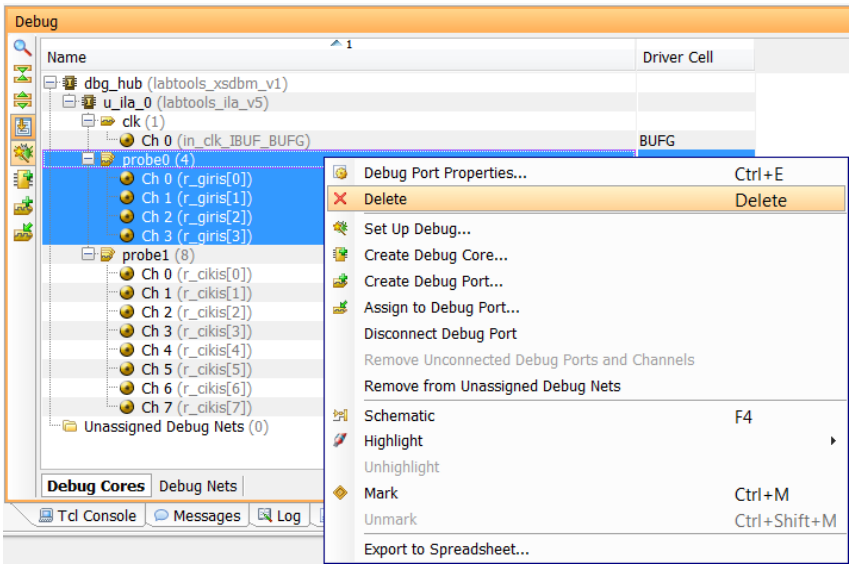
Şekil 10-28 Hata ayıklama işlemleri - 6

İşlemin bitirilmesinin ardından **Debug** penceresinde bağlantılar görülmektedir (Şekil 10-29).

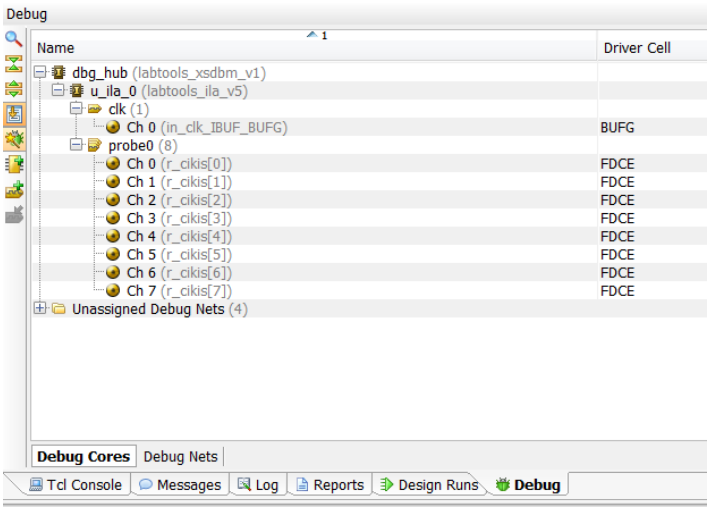


Şekil 10-29 Hata ayıklama işlemleri - 7

Var olan bağlantıyı silmek için bağlantı üzerine sağ tıklanır ve **Delete** seçilir. Daha sonra açılan pencereden Yes seçilirerek bağlantı silinir (Şekil 10-30, Şekil 10-31).

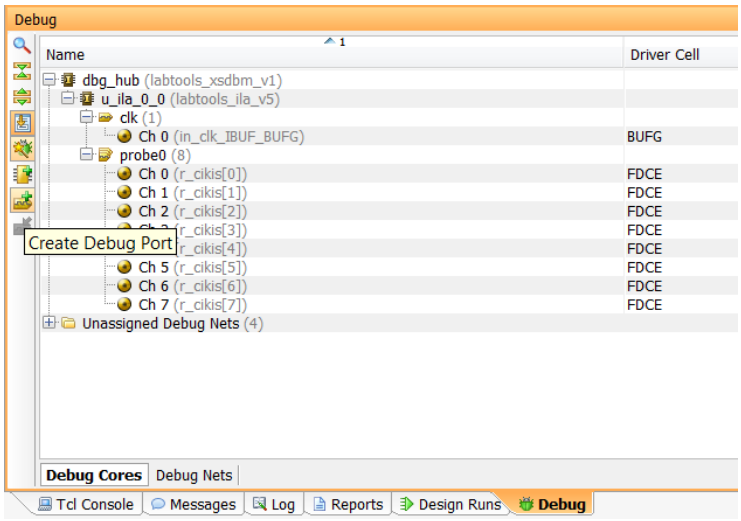


Şekil 10-30 Hata ayıklama işlemleri - 8



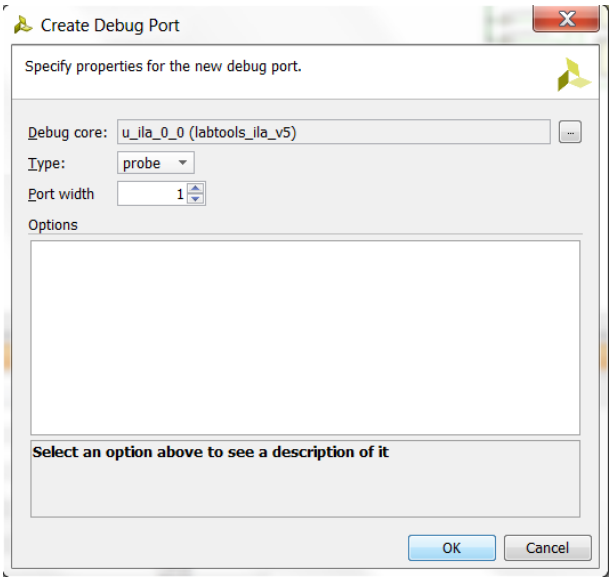
Şekil 10-31 Hata ayıklama işlemleri - 9

Yeni bağlantı eklemek için **`Debug`** penceresinde **`Create Debug Port`** seçilir (Şekil 10-32).



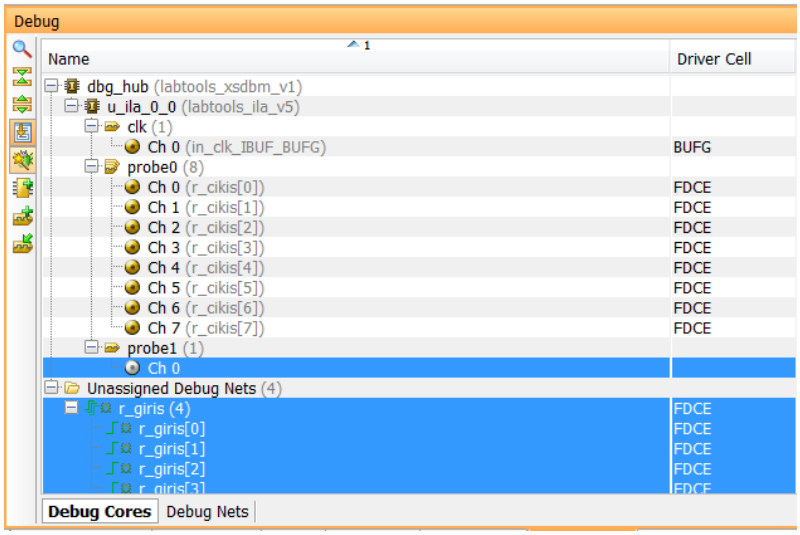
Şekil 10-32 Hata ayıklama işlemleri - 10

Açılan pencerede **Ok** butonuna basılır (Şekil 10-33).



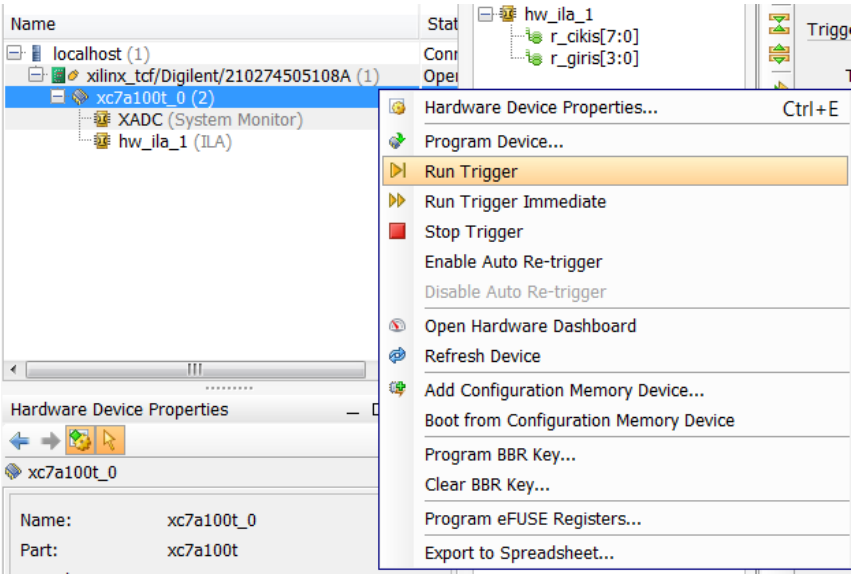
Şekil 10-33 Hata ayıklama işlemleri - 11

Oluşan yeni bağlantı noktasına boşta olan sinyal sürüklenerek bırakılırak yeni bağlantı oluşturulur.



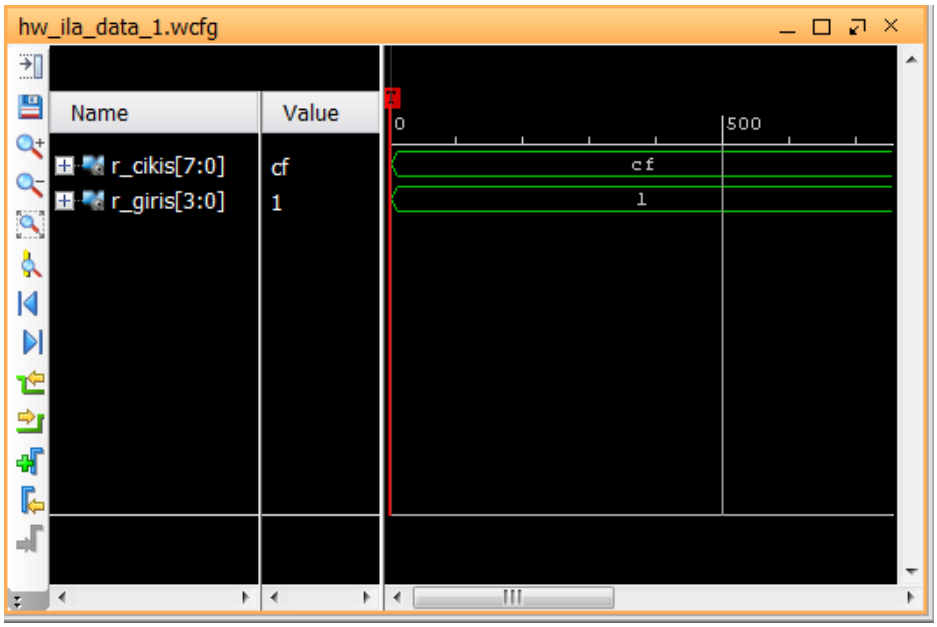
Şekil 10-34 Hata ayıklama işlemleri - 12

Bağlantı işlemlerinin tamamlanmasından sonra tüm işlemler kaydedilir. Analtıma **`Implementation`** ve Generate Bitstream aşamalarının tamamlanıp kodun FPGA'ya yüklendiği varsayımı yapılarak devam edilecektir. Yükelme işlemini bitiminden Name penceresinde **xc7a100t_0** sekmesine sağ tıklanırak Run Trigger seçilir (Şekil 10-35).



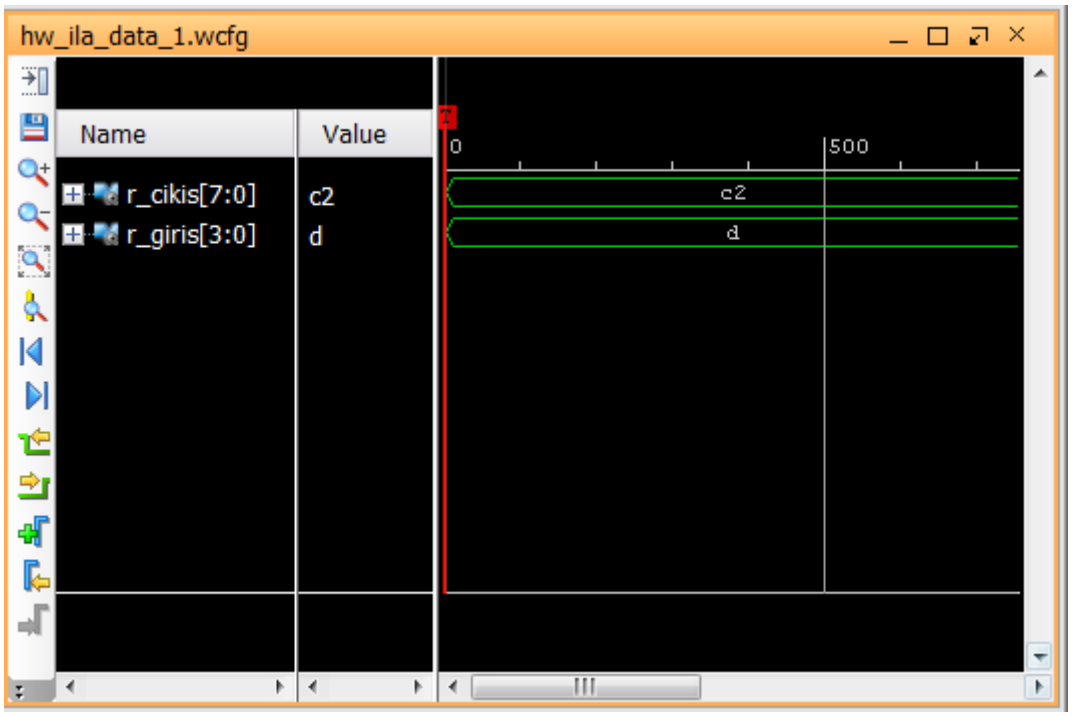
Şekil 10-35 Hata ayıklama işlemleri - 13

Anahtarlarla giriş portuna **`"0001"`** değeri gönderilmiştir. Şekil 10-36'den de görüleceği üzere **`r_giris`** sinyali 1 hex değerini ve **`r_cikis`** sinyali de **`DISP_EKRAN(1)`** değerini almıştırlar.



Şekil 10-36 Hata ayıklama işlemleri - 14

Anahtarlarla giriş portuna **"1101"** değeri gönderilmiştir. Şekil 10-37'den de görüleceği üzere **r_giris** sinyali d hex değerini ve **r_cikis** sinyali de **DISP_EKRAN(14)** değerini almıştırlar.



Şekil 10-37 Hata ayıklama işlemleri - 15

10.4. Karakter Kaydırma Uygulaması

Aşğaıda verilen **karakter_kaydirma.vhd** VHDL kod ile tanımlı 4 bitlik giriş portunun aldığı değerleri kullanıcının belirlediği zaman aralıklarında RAM'a yazlmkatadır ve RAM dataları sola kayma işlemi yapmaktadır. Bu şekilde displayler üzerinde kullanıcının belirlediği zaman aralığında 0-F aralığında tanımlı karakterlerin sola doğru kaydığu görülecektir.50-63. satırlarda tanımlı process işleminde kullanıcının belirlediği zaman diliminin dolması beklenmekte ve zaman dolması ile beraber 33-43. satırları arasında tanımlı fonsiyon çağırılarak RAM dataları sola kaydırılmakta ve en sağına giriş datası yazılmaktadır. 65-91. satırlar arasınd tanımlı process ile displaya yazılacak RAM verisi belirlenme işlemi yaılmaktadır. 93-106. Satırlarda tanımlı process ile display sürme işlemi yapılmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use IEEE.STD_LOGIC_UNSIGNED.ALL;
4.
5. entity karakter_kaydirma is
6.   Port (
7.     in_clk : in std_logic;
8.     in_rst : in std_logic;
9.     in_giris_data : in std_logic_vector(3 downto 0);
10.    out_disp_sec : out std_logic_vector(7 downto 0);
11.    out_cikis : out std_logic_vector(7 downto 0)
12.  );
13.end karakter_kaydirma;
14.
15.architecture Behavioral of karakter_kaydirma is
16.
17.  type t_display_ekran is array (0 to 15) of std_logic_vector(7 downto
   0);
18.  constant DISP_EKRAN : t_display_ekran := ("10000001", "11001111",
   "10010010",
19.  "10000110",   "11001100",   "10100100",   "10100000",   "10001111",
   "10000000",
20.  "10000100",   "10001000",   "11100000",   "10110001",   "11000010",
   "10110000",
21.  "10111000");
22.
23.  type t_RAM_data is array (0 to 7) of std_logic_vector(3 downto 0);
24.  signal r_RAM_data : t_RAM_data := (others => (others => '0'));
25.
26.  constant BEKLEME : integer := 3;
27.
28.  signal r_disp_sec : std_logic_vector(7 downto 0) := "11111110";
29.  signal r_cikis : std_logic_vector(7 downto 0) := (others => '0');
30.  signal r_sayac_clk : integer := 0;
31.  signal r_sayac_disp : integer := 0;
32.
33.  function f_kaydir(in_giris : std_logic_vector(3 downto 0); r_RAM_data
   : t_RAM_data )
34.  return t_RAM_data is
35.    variable v_RAM_data : t_RAM_data;
36.  begin
37.    v_RAM_data := r_RAM_data;
```

```
38.     for n_i in 6 downto 0 loop
39.       v_RAM_data(n_i + 1) := v_RAM_data(n_i);
40.     end loop;
41.     v_RAM_data(0) := in_giris;
42.     return v_RAM_data;
43.   end f_kaydir;
44.
45.begin
46.
47.   out_disp_sec <= r_disp_sec;
48.   out_cikis <= r_cikis;
49.
50.   process(in_clk, in_rst, in_giris_data)
51.   begin
52.     if in_rst = '1' then
53.       r_RAM_data <= (others => (others => '0'));
54.       r_sayac_clk <= 0;
55.     elsif rising_edge(in_clk) then
56.       if r_sayac_clk = BEKLEME * 100000000 - 1 then
57.         r_sayac_clk <= 0;
58.         r_RAM_data <= f_kaydir(in_giris_data, r_RAM_data);
59.       else
60.         r_sayac_clk <= r_sayac_clk + 1;
61.       end if;
62.     end if;
63.   end process;
64.
65.   process(in_clk, in_rst, r_disp_sec)
66.   begin
67.     if in_rst = '1' then
68.       r_cikis <= "00000000";
69.     elsif rising_edge(in_clk) then
70.       case r_disp_sec is
71.         when "11111110" =>
72.           r_cikis <= DISP_EKRAN(conv_integer(r_RAM_data(0)));
73.         when "11111101" =>
74.           r_cikis <= DISP_EKRAN(conv_integer(r_RAM_data(1)));
75.         when "11111011" =>
76.           r_cikis <= DISP_EKRAN(conv_integer(r_RAM_data(2)));
77.         when "11110111" =>
78.           r_cikis <= DISP_EKRAN(conv_integer(r_RAM_data(3)));
79.         when "11101111" =>
80.           r_cikis <= DISP_EKRAN(conv_integer(r_RAM_data(4)));
81.         when "11011111" =>
82.           r_cikis <= DISP_EKRAN(conv_integer(r_RAM_data(5)));
83.         when "10111111" =>
84.           r_cikis <= DISP_EKRAN(conv_integer(r_RAM_data(6)));
85.         when "01111111" =>
86.           r_cikis <= DISP_EKRAN(conv_integer(r_RAM_data(7)));
```

```
87.         when others =>
88.           r_cikis <= "00000000";
89.       end case;
90.     end if;
91.   end process;
92.
93.   process(in_clk, in_rst)
94.   begin
95.     if in_rst = '1' then
96.       r_disp_sec <= "11111110";
97.       r_sayac_disp <= 0;
98.     elsif rising_edge(in_clk) then
99.       if r_sayac_disp = 10000 then
100.        r_sayac_disp <= 0;
101.        r_disp_sec <= r_disp_sec(6 downto 0) & r_disp_sec(7);
102.      else
103.        r_sayac_disp <= r_sayac_disp + 1;
104.      end if;
105.      end if;
106.      end process;
107.
108.      end Behavioral;
```

display modülünün modülünün Nexys 4 kartında çalışabilmesi için gerekli port konumları ve port standarları Tablo 10-3'de verilmiştir.

Tablo 10-4 Display modülünün Nexys 4 kartında bağlantıları

Port	Konum	I/O Standart
in_clk	E3	LVCMOS33
in_rst	V10	LVCMOS33
in_giris_data(0)	U9	LVCMOS33
in_giris_data(1)	U8	LVCMOS33
in_giris_data(2)	R7	LVCMOS33
in_giris_data(3)	R6	LVCMOS33
out_disp_sec(0)	N6	LVCMOS33
out_disp_sec(1)	M6	LVCMOS33
out_disp_sec(2)	M3	LVCMOS33
out_disp_sec(3)	N5	LVCMOS33
out_disp_sec(4)	N2	LVCMOS33
out_disp_sec(5)	N4	LVCMOS33
out_disp_sec(6)	L1	LVCMOS33
out_disp_sec(7)	M1	LVCMOS33
out_cikis(0)	L6	LVCMOS33
out_cikis(1)	M2	LVCMOS33
out_cikis(2)	K3	LVCMOS33
out_cikis(3)	L4	LVCMOS33
out_cikis(4)	L5	LVCMOS33
out_cikis(5)	N1	LVCMOS33
out_cikis(6)	L3	LVCMOS33
out_cikis(7)	M4	LVCMOS33

10.5. UART Protokolü Kullanarak Data Kontrolü

UART (Universal asynchronous receiver/transmitter – Evrensel eşzamanlı olmaya alıcı/verici) paralel ve seri formlar arasında data çevrim işlemi yapan bilgisayar donanım parçasından biridir. UART'lar RS-232, RS-485 gibi yaygın iletişim standartları ile birlikte kullanılır.

Aşağıda VHDL dilinde UART modülü ile data alma ve gönderme işlemlerinin gerçekleştirildiği örnekler verilmiştir.

Örnek 10.1.1: Aşağıda UART protokoklü kullanarak data gönderim işlemninin yapıldığı **UART_tx.vhd** VHDL kodu erilmiştir. **UART_tx** varlığımıza ilişkin generic bildirimleri 7-10. satırlarda yapılmıştır. generic bildirimi içerisinde yapılan değerler kullanılarak 23. satırda her bir bit değeri için gerekli saat darbesi sayısı hesaplanmaktadır. Port bildirim işlemleri 11-18 satırları arasında yapılmaktıdır. **UART_tx** varlığımız 1 bitlik başla biti, 1 bitlik bitir biti ve 8 bitlik data gönderecek şekilde tasarlanmıştır. **t_UART_tx** tipinde tanımlı **r_UART_tx** sinyali başlangıç durumunda **BOSTA** durumundadır ve **in_tx_basla** giriş portu değerinin **'1'** olmasını beklemektedir. **in_tx_basla** giriş portu değerinin **'1'** olması ile birlikte, **r_data** sinyaline **in_txt_data** giriş portu değeri atanmaktadır ve **r_UART_tx** sinyali **BASLA** durumuna dallanır. **BASLA** durumunda başla biti gönderim işlemi yapılmaktadır. Yani **CLK_BIT** sabitinin değeri kadar saat darbesinde bu durum içerisinde beklenmekte ve **out_tx_cikis** portuna **'1'** değeri gönderilmektedir. **CLK_BIT** sayısı kadar saat darbesi beklendikten sonra **r_UART_tx** sinyali **GONDER** durumuna dallanır. Bu durumda **r_data** sinyalinin en anlamsız bitinden en anlamlı bitine doğru tüm datalar **CLK_BIT** sayısı kadar saat darbesi süresi ile gönderilmektedir. Yani bu durum içerisinde 8 x **CLK_BIT** saat darbesi kadar beklenmektedir. Tüm bitlerin gönderim işleminden sonra **r_UART_tx** sinyali **BITIR** durumuna dallanır. Bu durum içerisinde bitir biti gönderim işlemi yapılmaktadır. Yani **CLK_BIT** sabitinin değeri kadar saat darbesinde bu durum içerisinde beklenmekte ve **out_tx_cikis** portuna **'1'** değeri gönderilmektedir. **CLK_BIT** sayısı kadar saat darbesi beklendikten sonra **r_UART_tx** sinyali **TAMAM** durumuna dallanır. **TAMAM** durumunda **r_tx_tamam** sinyali **'1'** değerini alarak data gönderim işleminin bittiği bildirilmektedir ve **r_UART_tx** sinyali **BOSTA** durumuna dallanır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use IEEE.STD_LOGIC_SIGNED.ALL;
4. use IEEE.STD_LOGIC_ARITH.ALL;
5.
6. entity UART_tx is
7.   Generic (
8.     CLK_FREKANS : integer := 100000000; -- 100 MHz
9.     BOUDRATE : integer := 115200
10.  );
11.  Port(
12.    in_clk : in std_logic;
13.    in_rst : in std_logic;
14.    in_tx_basla : in std_logic;
15.    in_tx_data : in std_logic_vector(7 downto 0);
16.    out_tx : out std_logic;
17.    out_tx_tamam : out std_logic
```

```
18.   );
19. end UART_tx;
20.
21. architecture Behavioral of UART_tx is
22.
23.   constant CLK_BIT : integer := CLK_FREKANS / BOUDRATE + 1;
24.
25.   type t_UART_tx is (BOSTA, BASLA, GONDER, BITIR, TAMAM);
26.   signal r_UART_tx : t_UART_tx := BOSTA;
27.   signal r_clk_sayac : integer range 0 to CLK_BIT - 1 := 0;
28.   signal r_data_ind : integer range 0 to 7 := 0;
29.   signal r_data   : std_logic_vector(7 downto 0) := (others => '0');
30.   signal r_tx : std_logic := '1';
31.   signal r_tx_tamam : std_logic := '0';
32.
33. begin
34.
35.   out_tx <= r_tx;
36.   out_tx_tamam <= r_tx_tamam;
37.
38.   process(in_clk, in_rst)
39.   begin
40.     if in_rst = '1' then
41.       r_UART_tx <= BOSTA;
42.       r_clk_sayac <= 0;
43.       r_data_ind <= 0;
44.       r_data <= (others => '0');
45.       r_tx <= '1';
46.       r_tx_tamam <= '0';
47.
48.     elsif rising_edge(in_clk) then
49.       r_tx_tamam <= '0';
50.       case r_UART_tx is
51.         when BOSTA =>
52.           r_tx <= '1';
53.           r_clk_sayac <= 0;
54.           r_data_ind <= 0;
55.           if in_tx_basla = '1' then
56.             r_data <= in_tx_data;
57.             r_UART_tx <= BASLA;
```

```
58.           end if;
59.
60.         when BASLA =>
61.           r_tx <= '0';
62.           if r_clk_sayac = CLK_BIT - 1 then
63.             r_clk_sayac <= 0;
64.             r_UART_tx <= GONDER;
65.           else
66.             r_clk_sayac <= r_clk_sayac + 1;
67.           end if;
68.
69.         when GONDER =>
70.           r_tx  <= r_data(r_data_ind);
71.           if r_clk_sayac = CLK_BIT - 1 then
72.             r_clk_sayac <= 0;
73.             if r_data_ind = 7 then
74.               r_data_ind <= 0;
75.               r_UART_tx <= BITIR;
76.             else
77.               r_data_ind <= r_data_ind + 1;
78.             end if;
79.           else
80.             r_clk_sayac <= r_clk_sayac + 1;
81.           end if;
82.
83.         when BITIR =>
84.           r_tx  <= '1';
85.           if r_clk_sayac = CLK_BIT - 1 then
86.             r_clk_sayac <= 0;
87.             r_UART_tx <= TAMAM;
88.           else
89.             r_clk_sayac <= r_clk_sayac + 1;
90.           end if;
91.
92.         when TAMAM =>
93.           r_tx  <= '1';
94.           r_tx_tamam <= '1';
95.           r_UART_tx <= BOSTA;
96.
97.         when others => NULL;
```

```
98.       end case;
99.     end if;
100.      end process;
101.
102.      end Behavioral;
```

Örnek 10.1.2: Aşağıda UART protokoklü kullanarak data alım işlemininin yapıldığı **UART_rx.vhd** VHDL kodu erilmiştir. **UART_rx** varlığımıza ilişkin generic bildirimleri 7-10. satırlarda yapılmıştır. generic bildirimi içerisinde yapılan değerler kullanılarak 22. satırda her bir bit değeri için gerekli saat darbesi sayısı hesaplanmaktadır. Port bildirim işlemleri 11-17 satırları arasında yapılmaktıdır. **UART_rx** varlığımız 8 bitlik data alacak şekilde tasarlanmıştır. 48. satırda saat darbelerinin farklı olmasından dolayı domain eşleştirme işlemi yapılmaktadır. **t_UART_rx** tipinde tanımlı **r_UART_rx** sinyali başlangıç durumunda **BOSTA** durumundadır ve **r_rx_cnt** sinyalin ilk 2 bitinin değerinin **"10"** olması beklemektedir. **r_rx_cnt** sinyalin ilk 2 bitinin değerinin **"10"** olması ile **r_UART_rx** sinyali **BASLA** durumuna dallanır. **BASLA** durumunda **CLK_BIT** sabitinin değerinin yarısı kadar beklenmektedir. **CLK_BIT/2** sayısı kadar saat darbesi beklendikten sonra **r_UART_rx** sinyali **DATA_AL** durumuna dallanır. Bu durumda **CLK_BIT** sayısı kadar saat darbesi süresi kadar beklendikten sonra **r_data** sinyalinin en anlamsız bitinden en anlamlı bitine doğru datalar yazılmaktadır. Yani bu durum içerisinde 8 x **CLK_BIT** saat darbesi kadar beklenmektedir. Tüm bitlerin alınma işleminden sonra **r_UART_rx** sinyali **BITIR** durumuna dallanır. Bu durum içerisinde bitir **CLK_BIT** sabitinin değeri kadar saat darbesi beklendikten sonra **r_UART_rx** sinyali **TAMAM** durumuna dallanır. **TAMAM** durumunda **r_rx_tamam** sinyali **'1'** değerini alarak data alım işleminin bittiği bildirilmektedir ve **r_UART_rx** sinyali **BOSTA** durumuna dallanır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3. use IEEE.STD_LOGIC_SIGNED.ALL;
4. use IEEE.STD_LOGIC_ARITH.ALL;
5.
6. entity UART_rx is
7.   Generic (
8.     CLK_FREKANS : integer := 100000000;
9.     BOUDRATE : integer := 115200
10.  );
11.  Port(
12.    in_clk : in std_logic;
13.    in_rst : in std_logic;
14.    in_rx : in std_logic;
15.    out_rx_data : out std_logic_vector(7 downto 0);
16.    out_rx_tamam : out std_logic
17.  );
18.end UART_rx;
19.
20.architecture Behavioral of UART_rx is
```

```
21.
22.  constant CLK_BIT : integer := CLK_FREKANS / BOUDRATE + 1;
23.
24.  type t_UART_rx is (BOSTA, BASLA, DATA_AL, BITIR, TAMAM);
25.  signal r_UART_rx : t_UART_rx := BOSTA;
26.  signal r_clk_sayac : integer range 0 to CLK_BIT - 1 := 0;
27.  signal r_data_ind : integer range 0 to 7 := 0;
28.  signal r_data   : std_logic_vector(7 downto 0) := (others => '0');
29.  signal r_rx_tamam : std_logic := '0';
30.  signal r_rx_cnt : std_logic_vector(2 downto 0) := (others => '0');
31.
32.begin
33.
34.  out_rx_data <= r_data;
35.  out_rx_tamam <= r_rx_tamam;
36.
37.  process(in_clk)
38.  begin
39.    if in_rst = '1' then
40.      r_UART_rx <= BOSTA;
41.      r_clk_sayac <= 0;
42.      r_data_ind <= 0;
43.      r_data <= (others => '0');
44.      r_rx_cnt <= (others => '0');
45.      r_rx_tamam <= '0';
46.
47.    elsif rising_edge(in_clk) then
48.      r_rx_cnt <= r_rx_cnt(1 downto 0) & in_rx;
49.      r_rx_tamam <= '0';
50.
51.      case r_UART_rx is
52.        when BOSTA =>
53.          if r_rx_cnt(2 downto 1) = "10" then
54.            r_UART_rx <= BASLA;
55.          end if;
56.
57.        when BASLA =>
58.          if r_clk_sayac = (CLK_BIT - 1) / 2 then
59.            r_clk_sayac <= 0;
60.            r_UART_rx <= DATA_AL;
```

```
61.           else
62.             r_clk_sayac <= r_clk_sayac + 1;
63.           end if;
64.
65.         when DATA_AL =>
66.           r_data(r_data_ind) <= r_rx_cnt(2);
67.           if r_clk_sayac = CLK_BIT - 1 then
68.             r_clk_sayac <= 0;
69.             if r_data_ind = 7 then
70.               r_data_ind <= 0;
71.               r_UART_rx <= BITIR;
72.             else
73.               r_data_ind <= r_data_ind + 1;
74.             end if;
75.           else
76.             r_clk_sayac <= r_clk_sayac + 1;
77.           end if;
78.
79.         when BITIR =>
80.           if r_clk_sayac = CLK_BIT - 1 then
81.             r_clk_sayac <= 0;
82.             r_UART_rx <= TAMAM;
83.           else
84.             r_clk_sayac <= r_clk_sayac + 1;
85.           end if;
86.
87.         when TAMAM =>
88.           r_rx_tamam <= '1';
89.           r_UART_rx <= BOSTA;
90.         when others => NULL;
91.       end case;
92.     end if;
93.   end process;
94.end Behavioral;
```

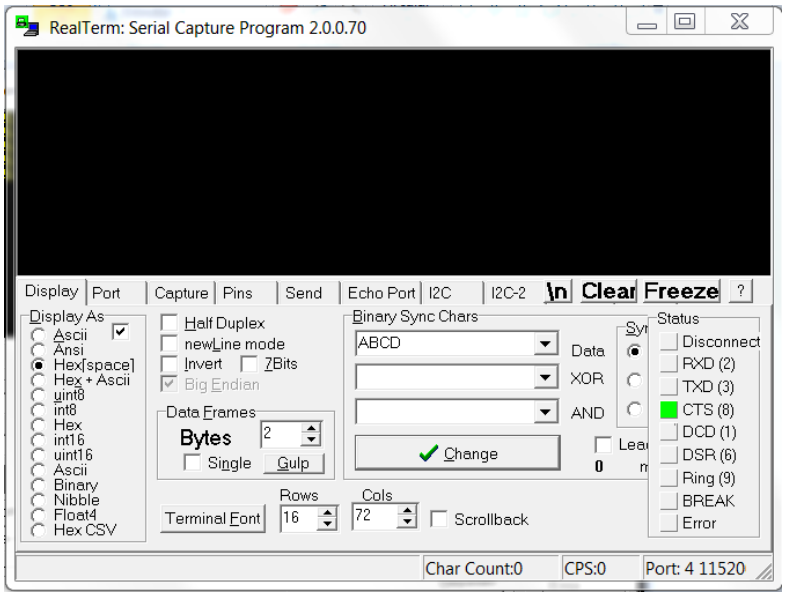
Örnek 10.1.3: Aşağıda UART protokoklü kullanarak data alım-gönderim işlemininin yapıldığı **UART_main.vhd** VHDL kodu erilmiştir. **UART_main** varlığımıza ilişkin port bildirim işlemleri 12-18 satırları arasında yapılmaktıdır. **UART_rx** varlığımız UART_rx alt devresinden aldığı dataları UART_tx alt devresi ile göndermek üzere tasarlanmıştır. UART_tx alt devresi component tanımlama işlemleri 15-28. satırlar arasında, bağlantı işlemleri ise 97-109. satırlar arasında yapılmaktadır. UART_rx alt devresi component tanımlama işlemleri 30-42. satırlar arasında, bağlantı işlemleri ise 84-95. satırlar arasında yapılmaktadır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.ALL;
3.
4. entity UART_main is
5.   Port (
6.     in_clk : in std_logic;
7.     in_rst : in std_logic;
8.     in_rx : in std_logic;
9.     out_tx : out std_logic
10.  );
11.end UART_main;
12.
13.architecture Behavioral of UART_main is
14.
15.  component UART_tx
16.  Generic (
17.   CLK_FREKANS : integer := 100000000;
18.   BOUDRATE : integer := 115200
19.  );
20.  Port(
21.    in_clk : in std_logic;
22.    in_rst : in std_logic;
23.    in_tx_basla : in std_logic;
24.    in_tx_data : in std_logic_vector(7 downto 0);
25.    out_tx : out std_logic;
26.    out_tx_tamam : out std_logic
27.  );
28.  end component;
29.
30.  component UART_rx
31.  Generic (
32.    CLK_FREKANS : integer := 100000000;
33.    BOUDRATE : integer := 115200
34.  );
35.  Port(
36.    in_clk : in std_logic;
37.    in_rst : in std_logic;
38.    in_rx : in std_logic;
39.    out_rx_data : out std_logic_vector(7 downto 0);
```

```
40.     out_rx_tamam : out std_logic
41.   );
42.   end component;
43.
44.   type t_Data_Cntrl is (BOSTA, DATA_AL, DATA_GONDER);
45.   signal r_Data_Cntrl : t_Data_Cntrl := BOSTA;
46.   signal r_tx_basla : std_logic := '0';
47.   signal r_tx_tamam : std_logic := '0';
48.   signal r_rx_tamam : std_logic := '0';
49.   signal r_data : std_logic_vector(7 downto 0);
50.   signal r_rx_data : std_logic_vector(7 downto 0);
51.   signal r_tx_data : std_logic_vector(7 downto 0);
52.
53.begin
54.
55.   process(in_clk, in_rst)
56.   begin
57.     if in_rst = '1' then
58.       r_Data_Cntrl <= BOSTA;
59.       r_data <= (others => '0');
60.
61.     elsif rising_edge(in_clk) then
62.       r_tx_basla <= '0';
63.
64.       case r_Data_Cntrl is
65.         when BOSTA =>
66.           r_Data_Cntrl <= DATA_AL;
67.
68.         when DATA_AL =>
69.           if r_rx_tamam = '1' then
70.             r_tx_data <= r_rx_data;
71.             r_tx_basla <= '1';
72.           end if;
73.
74.         when DATA_GONDER =>
75.           if r_tx_tamam = '1' then
76.             r_Data_Cntrl <= BOSTA;
77.           end if;
78.
79.         when others => NULL;
```

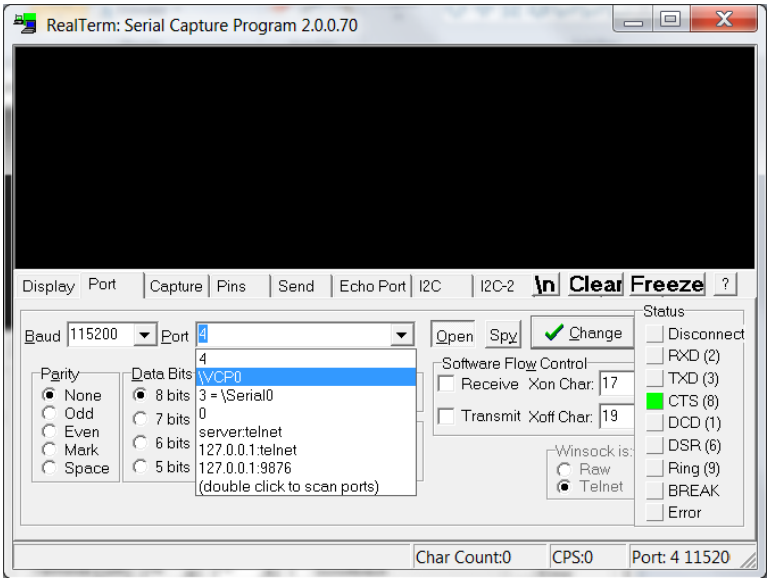
```
80.       end case;
81.     end if;
82.   end process;
83.
84.   UART_rx_map : UART_rx
85.   Generic map(
86.     CLK_FREKANS => 100000000, --100 MHz
87.     BOUDRATE => 115200
88.   )
89.   Port map(
90.     in_clk => in_clk,
91.     in_rst => in_rst,
92.     in_rx => in_rx,
93.     out_rx_data => r_rx_data,
94.     out_rx_tamam => r_rx_tamam
95.   );
96.
97.   UART_tx_map : UART_tx
98.   Generic map(
99.     CLK_FREKANS => 100000000, --100 MHz
100.          BOUDRATE => 115200
101.       )
102.       Port map(
103.         in_clk => in_clk,
104.         in_rst => in_rst,
105.         in_tx_basla => r_tx_basla,
106.         in_tx_data => r_tx_data,
107.         out_tx => out_tx,
108.         out_tx_tamam => r_tx_tamam
109.       );
110.
111.     end Behavioral;
```

UART_main modülümüzün test işlemleri için **Realterm** programı kullanılmıştır. **Realterm** programında **Display** sekmesinde **Display As**'de **Hex[space]** seçilerek ekranda gösterilecek karakter formatı seçilmektedir (Şekil 10-38).



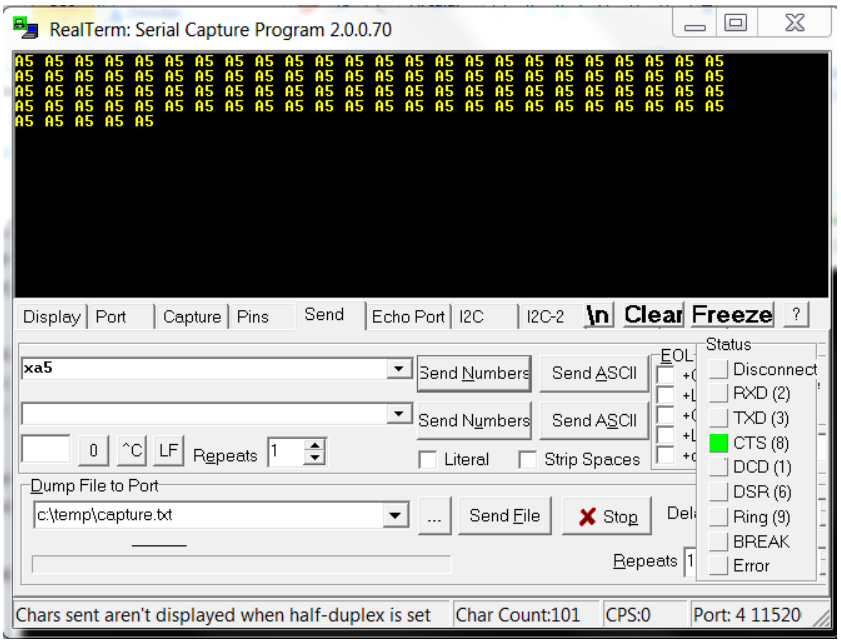
Şekil 10-38 UART_main modülü test işlemleri - 1

`Port` sekmesinde Baud sekmesinde 115200, Port sekmesinde ise \VCP0 seçilmektedir. Daha sonra Change butonuna basılarak değişiklikler yapılmaktadır (Şekil 10-39).



Şekil 10-39 UART_main modülü test işlemleri - 2

`Send` sekmesinde UART protokülü ile FPGA'ya göndermek istediğimiz datayı hex sayı formatında yazarak **`Send Numbers`** butonuna basılır. **`UART_main`** modülü aldığı datayı tekrar göndermek üzere tasarlandığından dolayı ekranda gönderilen data görülmektedir (Şekil 10-40).



Şekil 10-40 UART_main modülü test işlemleri - 3