# Nexys 4 Uygulamaları

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

## 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**;

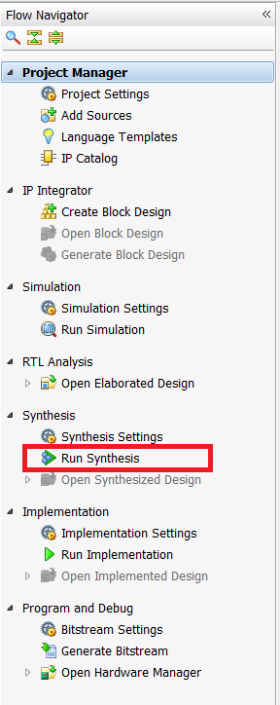
out\_cikis <= r\_cikis;

out\_cikis\_degil <= **not** r\_cikis;

**end** Behavioral;

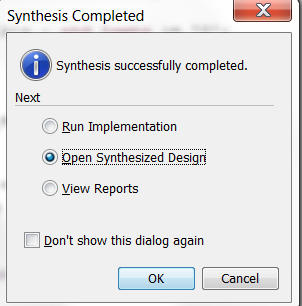
### Sentezleme ve Port Bağlantılarının Yapılması

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.

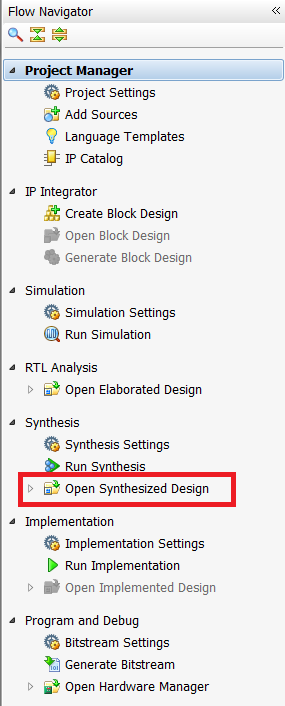


Şekil 10‑1 Sentezleme işleminin başlatılması

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).

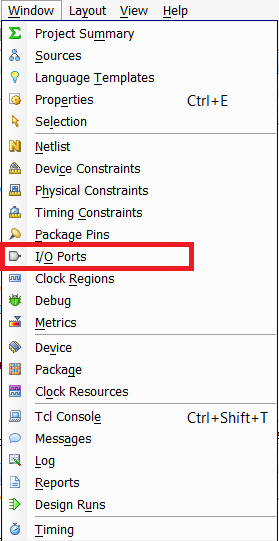


Şekil 10‑2 Sentezleme işleminin tamamlanması ve sentez tasarımlarının açılması

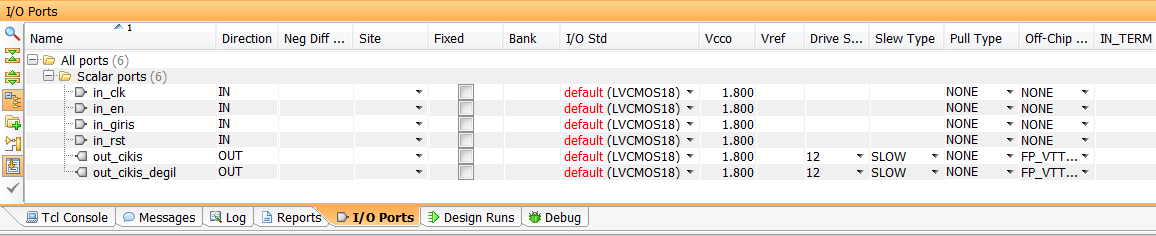


Şekil 10‑3 Sentez tasarımlarının açılması

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



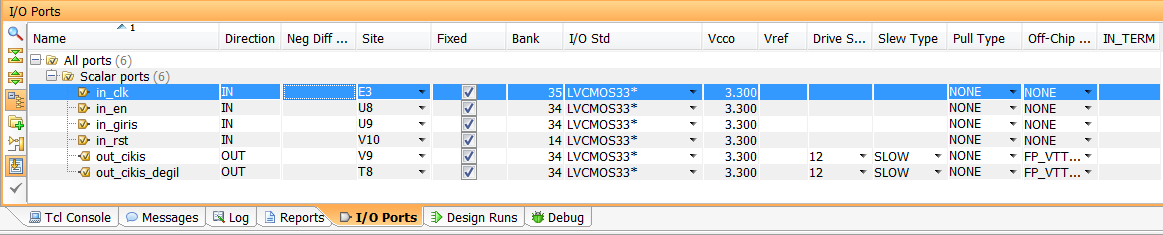
Şekil 10‑5 I/O Ports Pencersinin açılması - 2

**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]

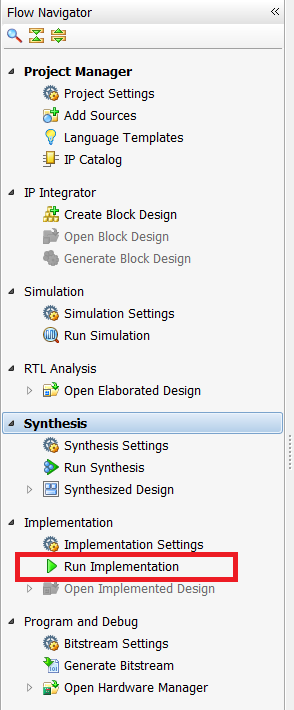
set\_property IOSTANDARD LVCMOS33 [get\_ports out\_cikis]

set\_property PACKAGE\_PIN T8 [get\_ports out\_cikis\_degil]

set\_property IOSTANDARD LVCMOS33 [get\_ports out\_cikis\_degil]

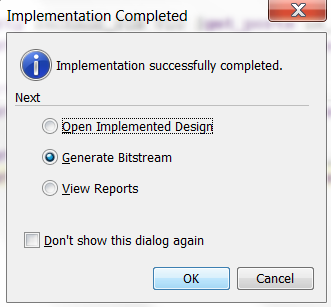
### Bit Dosyasının Oluşturulması

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

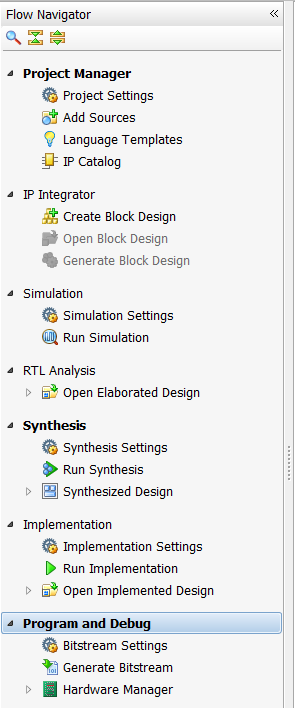


Şekil 10‑7 Gerçekleme işleminin başlatılması

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).



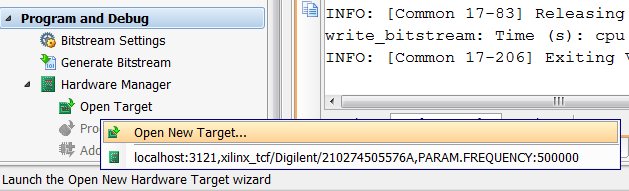
Şekil 10‑8 Gerçekleme işleminin tamamlanması



Şekil 10‑9 Bit dosyası oluşturma işleminin başlatılması

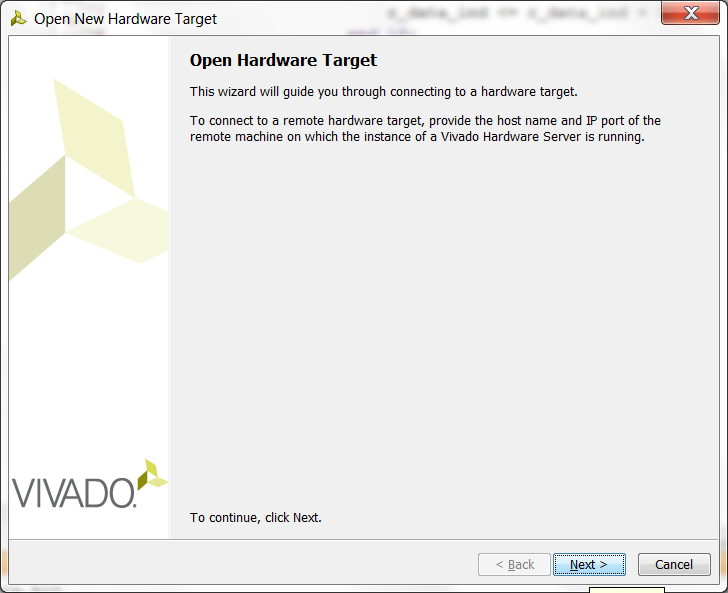
### Bit Dosyasının Yüklenmesi

Bit uzantılı dosyanın oluşuturlmasından 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.



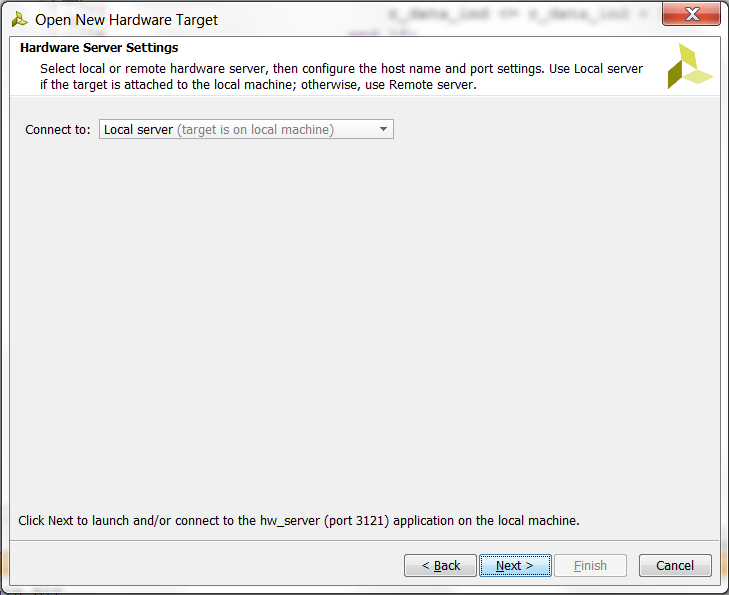
Şekil 10‑10 Kullanılacak FPGA ile bağlantı salanması - 1

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



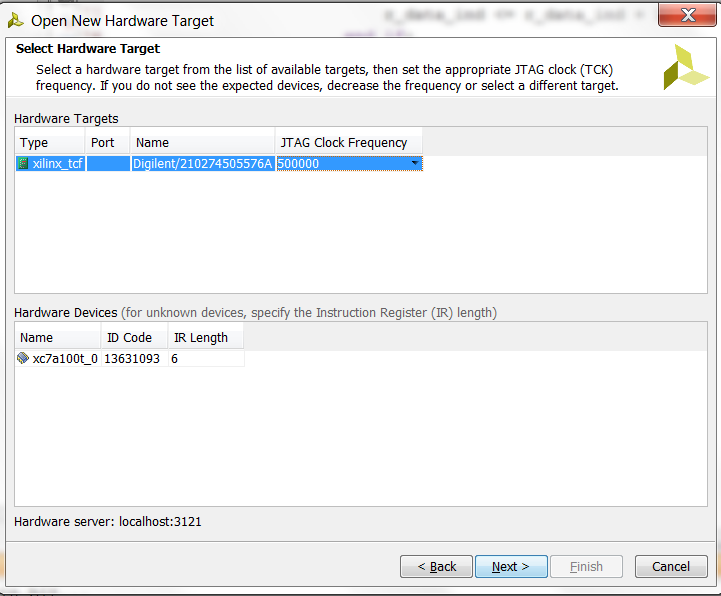
Şekil 10‑11 Kullanılacak FPGA ile bağlantı salanması -2

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



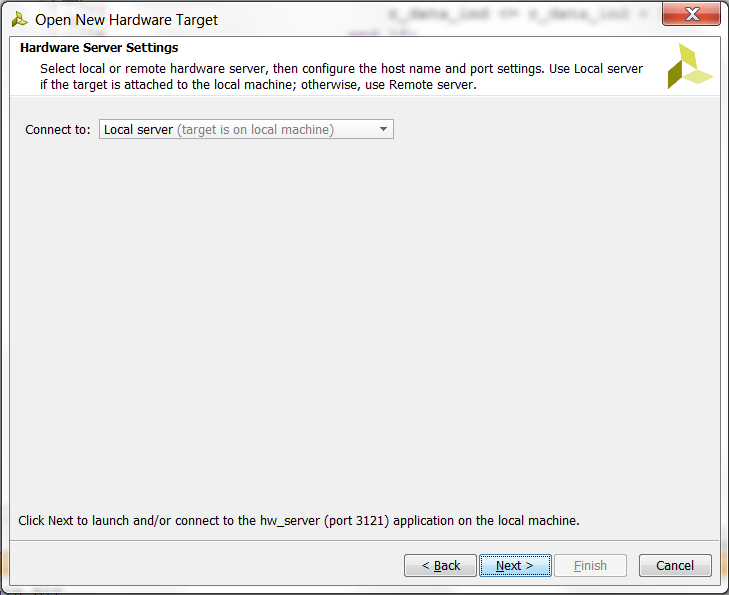
Şekil 10‑12 Kullanılacak FPGA ile bağlantı salanması -3

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.



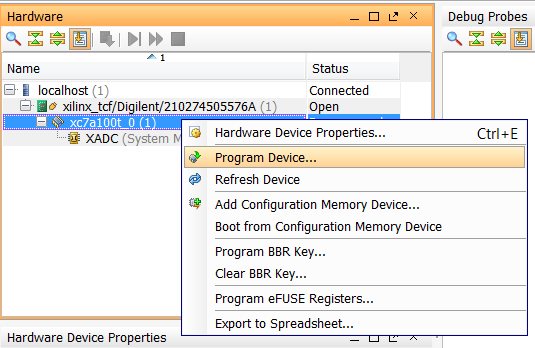
Şekil 10‑13 Kullanılacak FPGA ile bağlantı salanması -4

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



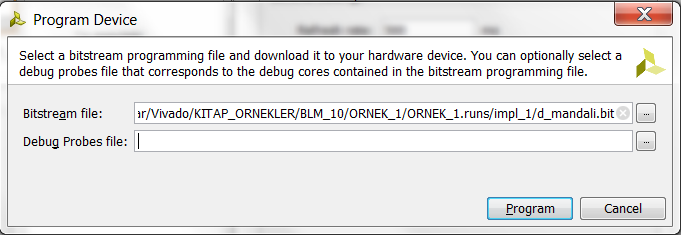
Şekil 10‑14 Kullanılacak FPGA ile bağlantı salanması -5

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

### 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.

## 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. **entity** led\_yakma **is**
4. **Port** (
5. in\_clk : **in** std\_logic;
6. in\_rst : **in** std\_logic;
7. out\_clk\_1Hz : **out** std\_logic;
8. out\_clk\_1\_2Hz : **out** std\_logic;
9. out\_clk\_1\_4Hz : **out** std\_logic;
10. out\_clk\_1\_8Hz : **out** std\_logic;
11. out\_clk\_1\_16Hz : **out** std\_logic;
12. out\_clk\_1\_NHz : **out** std\_logic
13. );
14. **end** led\_yakma;
15. **architecture** Behavioral **of** led\_yakma **is**
16. **component** saat\_frekans\_bolucu
17. **generic**(
18. N : integer := 16
19. );
20. **Port** (
21. in\_clk : **in** std\_logic;
22. in\_rst : **in** std\_logic;
23. out\_clk\_2 : **out** std\_logic;
24. out\_clk\_4 : **out** std\_logic;
25. out\_clk\_8 : **out** std\_logic;
26. out\_clk\_16 : **out** std\_logic;
27. out\_clk\_N : **out** std\_logic
28. );
29. **end** **component**;
30. **signal** r\_clk\_1Hz\_d : std\_logic := '0';
31. **signal** r\_clk\_1Hz : std\_logic\_vector(**3** downto **0**) := (**others** => '0');
32. **signal** r\_clk\_1\_2Hz : std\_logic := '0';
33. **signal** r\_clk\_1\_4Hz : std\_logic := '0';
34. **signal** r\_clk\_1\_8Hz : std\_logic := '0';
35. **signal** r\_clk\_1\_16Hz : std\_logic := '0';
36. **signal** r\_clk\_1\_NHz : std\_logic := '0';
37. **begin**
38. out\_clk\_1Hz <= r\_clk\_1Hz(3);
39. out\_clk\_1\_2Hz <= r\_clk\_1\_2Hz;
40. out\_clk\_1\_4Hz <= r\_clk\_1\_4Hz;
41. out\_clk\_1\_8Hz <= r\_clk\_1\_8Hz;
42. out\_clk\_1\_16Hz <= r\_clk\_1\_16Hz;
43. out\_clk\_1\_NHz <= r\_clk\_1\_NHz;
44. saat\_frekans\_bolucu1\_map : saat\_frekans\_bolucu
45. **generic** **map**( N => 100000000 )
46. **port** **map** (
47. in\_clk => in\_clk,
48. in\_rst => in\_rst,
49. out\_clk\_2 => **open**,
50. out\_clk\_4 => **open**,
51. out\_clk\_8 => **open**,
52. out\_clk\_16 => **open**,
53. out\_clk\_N => r\_clk\_1Hz\_d
54. );
55. **process**(in\_clk, in\_rst)
56. **begin**
57. **if** in\_rst = '1' **then**
58. r\_clk\_1Hz <= (**others** => '0');
59. **elsif** rising\_edge(in\_clk) **then**
60. r\_clk\_1Hz <= r\_clk\_1Hz(**2** downto **0**) & r\_clk\_1Hz\_d;
61. **end** **if**;
62. **end** **procesS**;
63. saat\_frekans\_bolucu2\_map : saat\_frekans\_bolucu
64. **generic** **map**( N => 32 )
65. **port** **map** (
66. in\_clk => r\_clk\_1Hz(3),
67. in\_rst => in\_rst,
68. out\_clk\_2 => r\_clk\_1\_2Hz,
69. out\_clk\_4 => r\_clk\_1\_4Hz,
70. out\_clk\_8 => r\_clk\_1\_8Hz,
71. out\_clk\_16 => r\_clk\_1\_16Hz,
72. out\_clk\_N => r\_clk\_1\_NHz
73. );
74. **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]

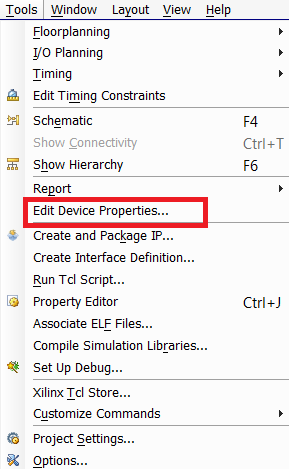
set\_property PACKAGE\_PIN T4 [get\_ports out\_clk\_1\_NHz]

set\_property IOSTANDARD LVCMOS33 [get\_ports out\_clk\_1\_NHz]

### Flash Dosyası Oluşturma

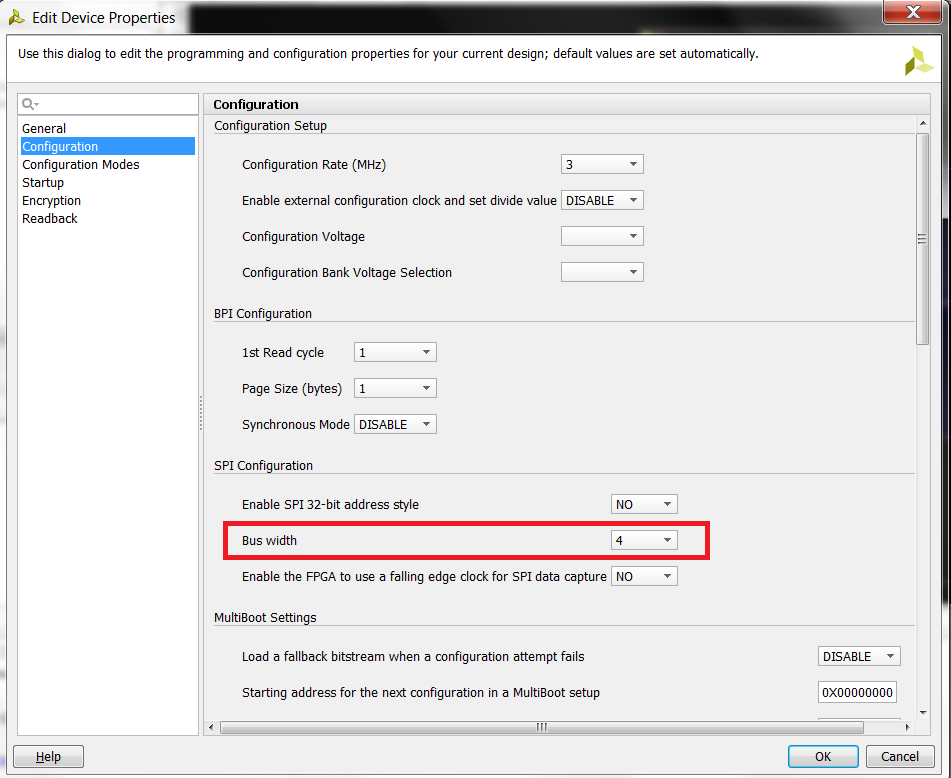
Neyxs 4 kartı üzerinde bulunan flash 4x SPI veri yoluna ve 128 Mb hafızaya sahiptir. Parça numarası **S25FL128S**’dir. Flash ayarlamalarını kendi projemiz içerisinde de yapmamız gerekmektedir.

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).



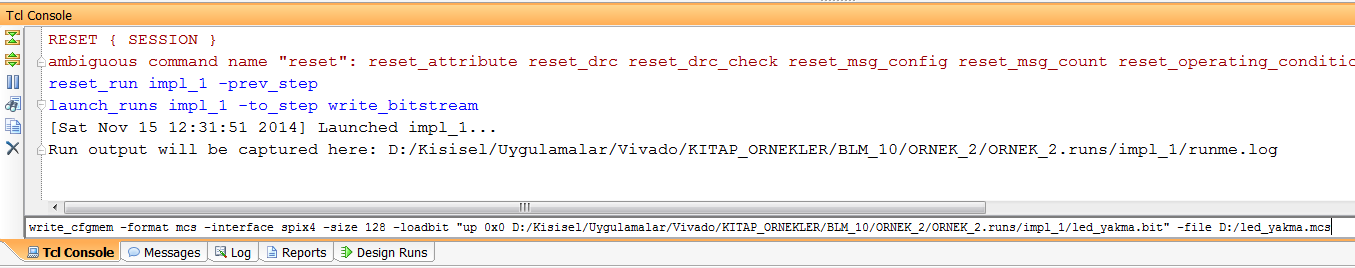
Şekil 10‑18 Flash ayarlarının yapılması - 2

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

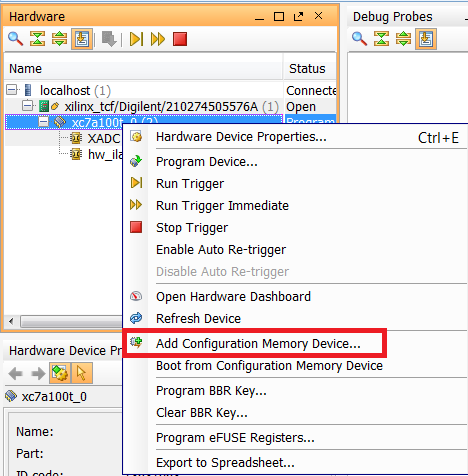
Örnek 10.2 için oluşturulan **led\_yakma.bit** uzantılı dosyadan flasha yazmak için **led\_yakma.mcs** uznatılı dosya elde etmek için aşağıdaki kod satırı TCL Console kısmına yazılır.

**write\_cfgmem** **-format** mcs **-interface** spix4 **-size** 128 **-loadbit** "up 0x0 D:/Kisisel/Uygulamalar/Vivado/KITAP\_ORNEKLER/BLM\_10/ORNEK\_2/ORNEK\_2.runs/impl\_1/led\_yakma.bit" **-file** D:/led\_yakma.mcs



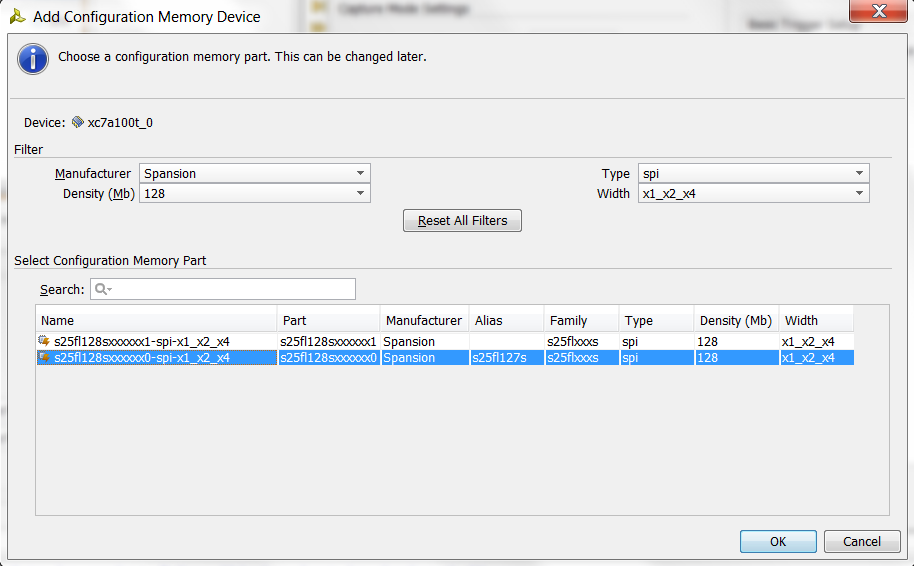
Şekil 10‑19 led\_yakma.mcs dosyasının oluşturulması

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).



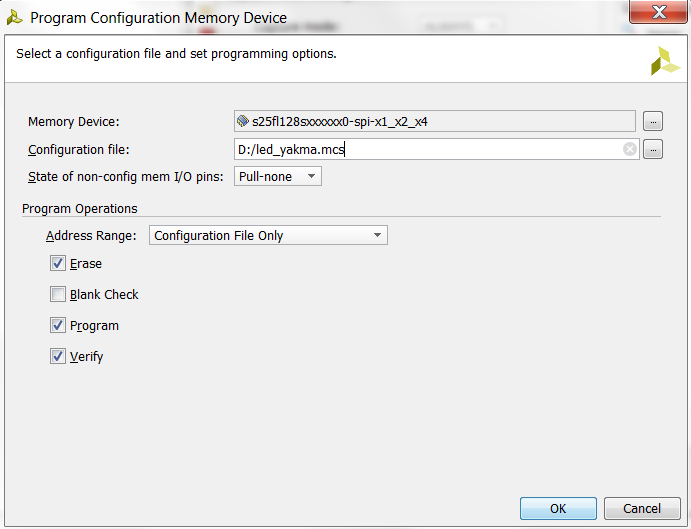
Şekil 10‑20 Flash dosyasının yüklenmesi - 1

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.

## 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.

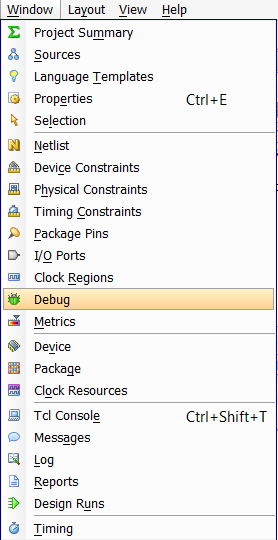
1. library IEEE;
2. use IEEE.STD\_LOGIC\_1164.ALL;
3. use IEEE.STD\_LOGIC\_UNSIGNED.ALL;
4. entity display is
5. Port (
6. in\_clk : in std\_logic;
7. in\_rst : in std\_logic;
8. in\_giris : in std\_logic\_vector(3 downto 0);
9. out\_disp\_sec : out std\_logic\_vector(7 downto 0);
10. out\_cikis : out std\_logic\_vector(7 downto 0)
11. );
12. end display;
13. architecture Behavioral of display is
14. type t\_display\_ekran is array (0 to 15) of std\_logic\_vector(7 downto 0);
15. constant DISP\_EKRAN : t\_display\_ekran := ("10000001", "11001111", "10010010",
16. "10000110", "11001100", "10100100", "10100000", "10001111", "10000000",
17. "10000100", "10001000", "11100000", "10110001", "11000010", "10110000",
18. "10111000");
19. signal r\_giris : std\_logic\_vector(3 downto 0) := (others => '0');
20. signal r\_cikis : std\_logic\_vector(7 downto 0) := (others => '0');
22. attribute syn\_keep : string;
23. attribute mark\_debug : string;
24. attribute syn\_keep of r\_giris : signal is "true";
25. attribute mark\_debug of r\_giris : signal is "true";
26. attribute syn\_keep of r\_cikis : signal is "true";
27. attribute mark\_debug of r\_cikis : signal is "true";
28. begin
30. out\_disp\_sec <= "00000000";
31. out\_cikis <= r\_cikis;
33. process(in\_clk, in\_rst, in\_giris)
34. begin
35. if in\_rst = '1' then
36. r\_giris <= (others => '0');
37. r\_cikis <= (others => '0');
38. elsif rising\_edge(in\_clk) then
39. r\_giris <= in\_giris;
40. r\_cikis <= DISP\_EKRAN(conv\_integer(r\_giris));
41. end if;
42. end process;
43. 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

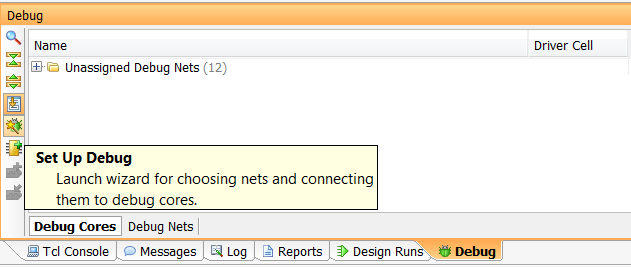
|  |  |  |
| --- | --- | --- |
| Port | Konum | I/O Standart |
| in\_clk | E3 | LVCMOS33 |
| in\_rst | V10 | LVCMOS33 |
| in\_giris(0) | U9 | LVCMOS33 |
| in\_giris(1) | U8 | LVCMOS33 |
| in\_giris(2) | R7 | LVCMOS33 |
| in\_giris(3) | R6 | LVCMOS33 |
| out\_disp\_sec(0) | N6 | LVCMOS33 |
| out\_disp\_sec(1) | 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).



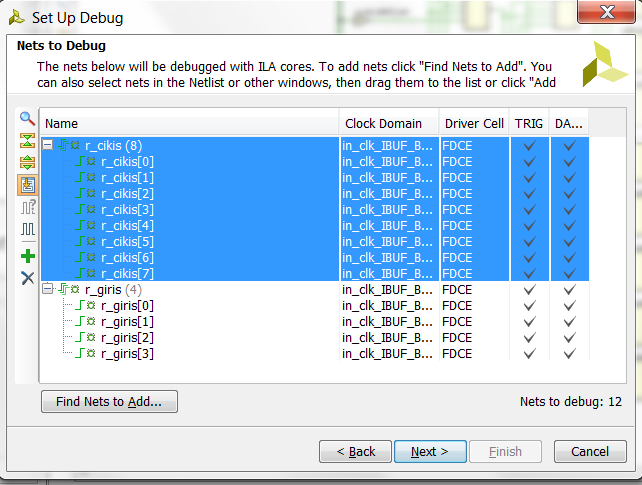
Şekil 10‑24 Hata ayıklama işlemleri - 2

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



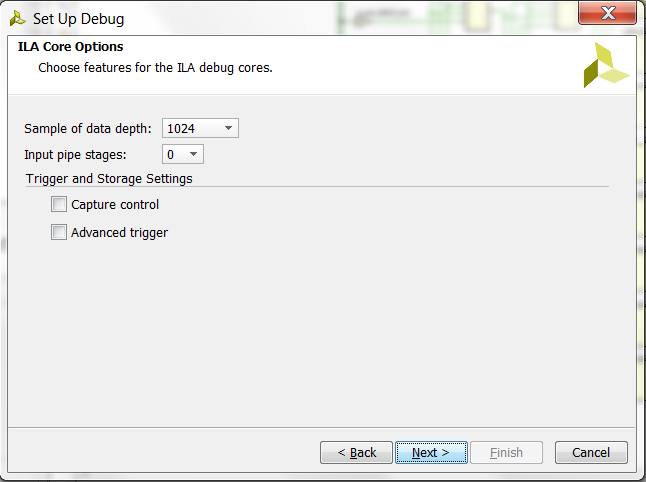
Şekil 10‑25 Hata ayıklama işlemleri - 3

Niteliklerde tanımlı **r\_cikis** ve **r\_giris** sinyallerinin listelendiği görülmektedir. **Next** butonuna basılarak bir sonraki aşamaya geçilir (Şekil 10‑26).



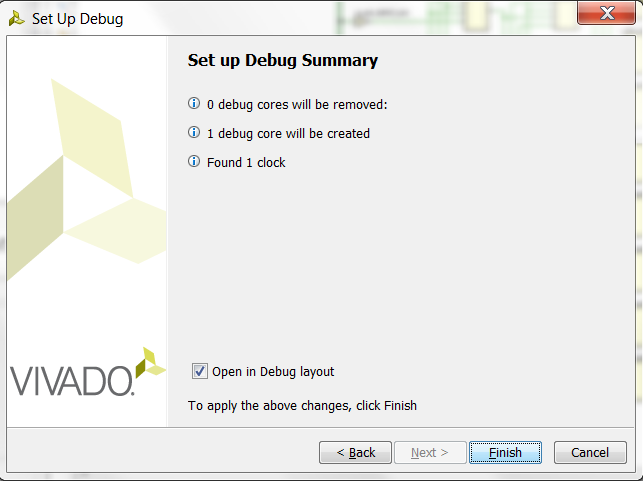
Şekil 10‑26 Hata ayıklama işlemleri - 4

**Next** butonuna basılarak bir sonraki aşamaya geçilir (Şekil 10‑27Şekil 10‑26).



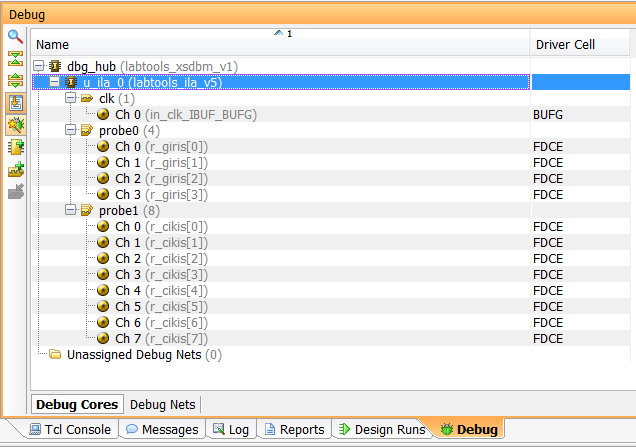
Şekil 10‑27 Hata ayıklama işlemleri - 5

**Finish** butonuna basılarak işlem sonlandırılır (Şekil 10‑28).



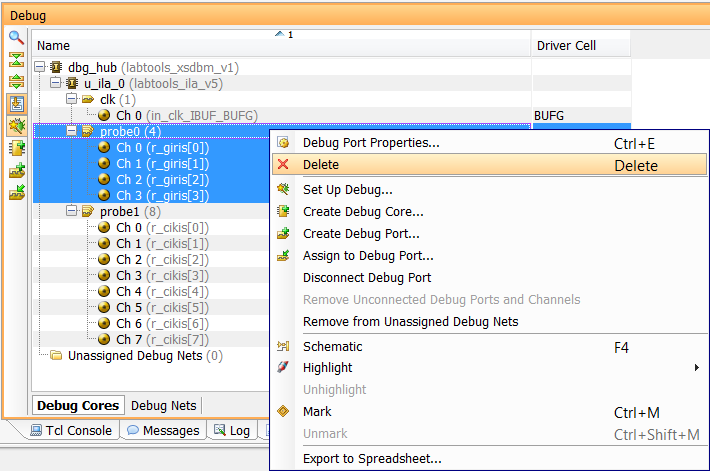
Şekil 10‑28 Hata ayıklama işlemleri - 6

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

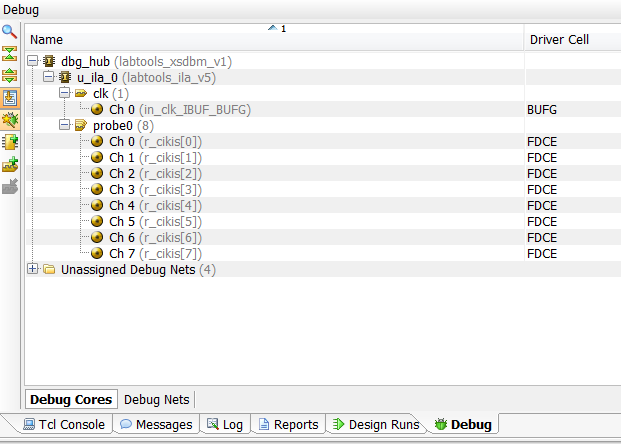


Şekil 10‑29 Hata ayıklama işlemleri - 7

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).

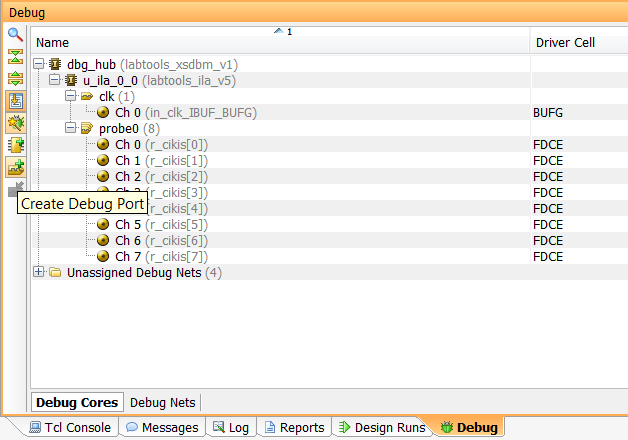


Şekil 10‑30 Hata ayıklama işlemleri - 8



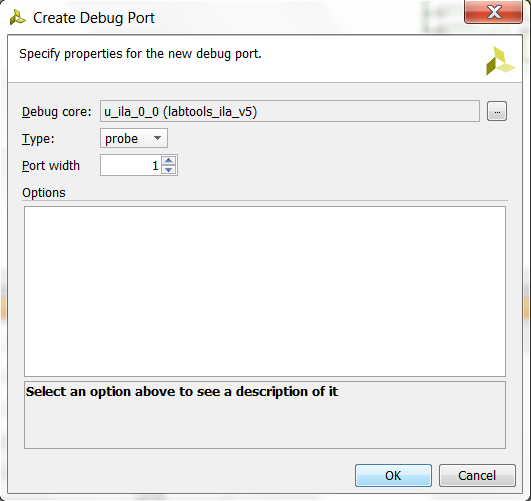
Şekil 10‑31 Hata ayıklama işlemleri - 9

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



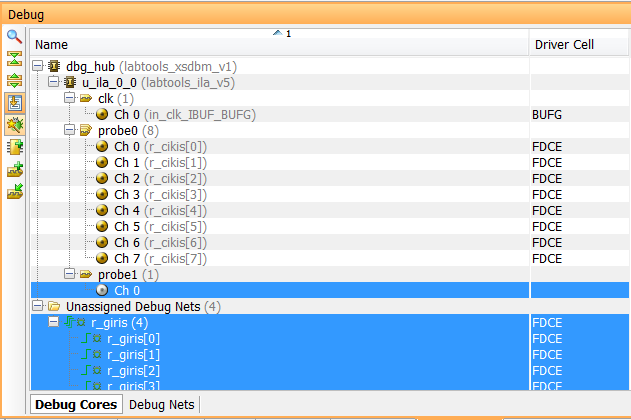
Şekil 10‑32 Hata ayıklama işlemleri - 10

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



Şekil 10‑33 Hata ayıklama işlemleri - 11

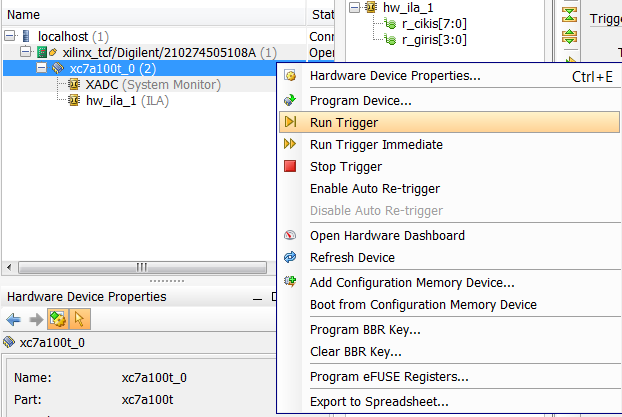
Oluşan yeni bağlantı noktasına boşta olan sinyal sürüklenerek bırakılırak yeni bağlantı oluşturulur.



Şekil 10‑34 Hata ayıklama işlemleri - 12

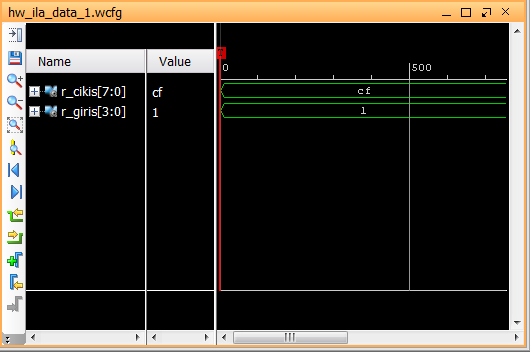
Bağlantı işlemlerinin tamamlanmasından sonra tüm işlemler kaydedilir. Analtıma **Implementation** ve Generate Bitstream aşamalarının tamamlanıp kodun FPGA’ya yüklendiği varsayımı yapılarak devam edilecektir.

Yükelme işlemini bitiminden Name penceresinde **xc7a100t\_0** sekmesine sağ tıklanırak Run Trigger seçilir (Şekil 10‑35).



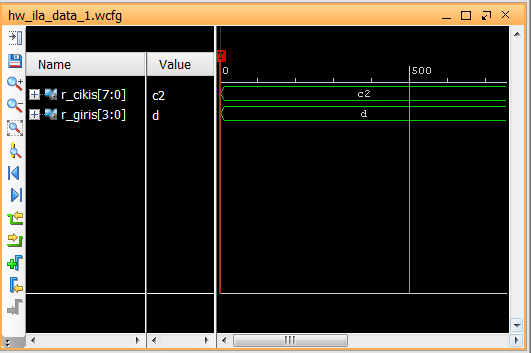
Şekil 10‑35 Hata ayıklama işlemleri - 13

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

## 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. **entity** karakter\_kaydirma **is**
5. **Port** (
6. in\_clk : **in** std\_logic;
7. in\_rst : **in** std\_logic;
8. in\_giris\_data : **in** std\_logic\_vector(**3** downto **0**);
9. out\_disp\_sec : **out** std\_logic\_vector(**7** downto **0**);
10. out\_cikis : **out** std\_logic\_vector(**7** downto **0**)
11. );
12. **end** karakter\_kaydirma;
13. **architecture** Behavioral **of** karakter\_kaydirma **is**
14. **type** t\_display\_ekran **is** **array** (**0** to **15**) **of** std\_logic\_vector(**7** downto **0**);
15. **constant** DISP\_EKRAN : t\_display\_ekran := ("10000001", "11001111", "10010010",
16. "10000110", "11001100", "10100100", "10100000", "10001111", "10000000",
17. "10000100", "10001000", "11100000", "10110001", "11000010", "10110000",
18. "10111000");
19. **type** t\_RAM\_data **is** **array** (**0** to **7**) **of** std\_logic\_vector(**3** downto **0**);
20. **signal** r\_RAM\_data : t\_RAM\_data := (**others** => (**others** => '0'));
21. **constant** BEKLEME : integer := 3;
22. **signal** r\_disp\_sec : std\_logic\_vector(**7** downto **0**) := "11111110";
23. **signal** r\_cikis : std\_logic\_vector(**7** downto **0**) := (others => '0');
24. **signal** r\_sayac\_clk : integer := 0;
25. **signal** r\_sayac\_disp : integer := 0;
26. **function** f\_kaydir(in\_giris : std\_logic\_vector(**3** downto **0**); r\_RAM\_data : t\_RAM\_data )
27. **return** t\_RAM\_data **is**
28. **variable** v\_RAM\_data : t\_RAM\_data;
29. **begin**
30. v\_RAM\_data := r\_RAM\_data;
31. **for** n\_i **in** **6** downto **0** **loop**
32. v\_RAM\_data(n\_i + 1) := v\_RAM\_data(n\_i);
33. **end** **loop**;
34. v\_RAM\_data(0) := in\_giris;
35. **return** v\_RAM\_data;
36. **end** f\_kaydir;
37. **begin**
38. out\_disp\_sec <= r\_disp\_sec;
39. out\_cikis <= r\_cikis;
40. **process**(in\_clk, in\_rst, in\_giris\_data)
41. **begin**
42. **if** in\_rst = '1' **then**
43. r\_RAM\_data <= (**others** => (**others** => '0'));
44. r\_sayac\_clk <= 0;
45. **elsif** rising\_edge(in\_clk) **then**
46. **if** r\_sayac\_clk = BEKLEME \* 100000000 - 1 **then**
47. r\_sayac\_clk <= 0;
48. r\_RAM\_data <= f\_kaydir(in\_giris\_data, r\_RAM\_data);
49. **else**
50. r\_sayac\_clk <= r\_sayac\_clk + 1;
51. **end** **if**;
52. **end** **if**;
53. **end** **process**;
54. **process**(in\_clk, in\_rst, r\_disp\_sec)
55. **begin**
56. **if** in\_rst = '1' **then**
57. r\_cikis <= "00000000";
58. **elsif** rising\_edge(in\_clk) **then**
59. **case** r\_disp\_sec **is**
60. **when** "11111110" =>
61. r\_cikis <= DISP\_EKRAN(conv\_integer(r\_RAM\_data(0)));
62. **when** "11111101" =>
63. r\_cikis <= DISP\_EKRAN(conv\_integer(r\_RAM\_data(1)));
64. **when** "11111011" =>
65. r\_cikis <= DISP\_EKRAN(conv\_integer(r\_RAM\_data(2)));
66. **when** "11110111" =>
67. r\_cikis <= DISP\_EKRAN(conv\_integer(r\_RAM\_data(3)));
68. **when** "11101111" =>
69. r\_cikis <= DISP\_EKRAN(conv\_integer(r\_RAM\_data(4)));
70. **when** "11011111" =>
71. r\_cikis <= DISP\_EKRAN(conv\_integer(r\_RAM\_data(5)));
72. **when** "10111111" =>
73. r\_cikis <= DISP\_EKRAN(conv\_integer(r\_RAM\_data(6)));
74. **when** "01111111" =>
75. r\_cikis <= DISP\_EKRAN(conv\_integer(r\_RAM\_data(7)));
76. **when** **others** =>
77. r\_cikis <= "00000000";
78. **end** **case**;
79. **end** **if**;
80. **end** **process**;
81. **process**(in\_clk, in\_rst)
82. **begin**
83. **if** in\_rst = '1' **then**
84. r\_disp\_sec <= "11111110";
85. r\_sayac\_disp <= 0;
86. **elsif** rising\_edge(in\_clk) **then**
87. **if** r\_sayac\_disp = 10000 **then**
88. r\_sayac\_disp <= 0;
89. r\_disp\_sec <= r\_disp\_sec(**6** downto **0**) & r\_disp\_sec(7);
90. **else**
91. r\_sayac\_disp <= r\_sayac\_disp + 1;
92. **end** **if**;
93. **end** **if**;
94. **end** **process**;
95. **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 |

## 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. **entity** UART\_tx **is**
6. **Generic** (
7. CLK\_FREKANS : integer := 100000000; -- 100 MHz
8. BOUDRATE : integer := 115200
9. );
10. **Port**(
11. in\_clk : **in** std\_logic;
12. in\_rst : **in** std\_logic;
13. in\_tx\_basla : **in** std\_logic;
14. in\_tx\_data : **in** std\_logic\_vector(**7** downto **0**);
15. out\_tx : **out** std\_logic;
16. out\_tx\_tamam : **out** std\_logic
17. );
18. **end** UART\_tx;
19. **architecture** Behavioral **of** UART\_tx **is**
20. **constant** CLK\_BIT : integer := CLK\_FREKANS / BOUDRATE + 1;
21. **type** t\_UART\_tx **is** (BOSTA, BASLA, GONDER, BITIR, TAMAM);
22. **signal** r\_UART\_tx : t\_UART\_tx := BOSTA;
23. **signal** r\_clk\_sayac : integer range **0** to **CLK\_BIT - 1** := 0;
24. **signal** r\_data\_ind : integer range **0** to **7** := 0;
25. **signal** r\_data : std\_logic\_vector(**7** downto **0**) := (**others** => '0');
26. **signal** r\_tx : std\_logic := '1';
27. **signal** r\_tx\_tamam : std\_logic := '0';
28. **begin**
29. out\_tx <= r\_tx;
30. out\_tx\_tamam <= r\_tx\_tamam;
31. **process**(in\_clk, in\_rst)
32. **begin**
33. **if** in\_rst = '1' **then**
34. r\_UART\_tx <= BOSTA;
35. r\_clk\_sayac <= 0;
36. r\_data\_ind <= 0;
37. r\_data <= (**others** => '0');
38. r\_tx <= '1';
39. r\_tx\_tamam <= '0';
40. **elsif** rising\_edge(in\_clk) **then**
41. r\_tx\_tamam <= '0';
42. **case** r\_UART\_tx **is**
43. **when** BOSTA =>
44. r\_tx <= '1';
45. r\_clk\_sayac <= 0;
46. r\_data\_ind <= 0;
47. **if** in\_tx\_basla = '1' **then**
48. r\_data <= in\_tx\_data;
49. r\_UART\_tx <= BASLA;
50. **end** **if**;
51. **when** BASLA =>
52. r\_tx <= '0';
53. **if** r\_clk\_sayac = CLK\_BIT - 1 **then**
54. r\_clk\_sayac <= 0;
55. r\_UART\_tx <= GONDER;
56. **else**
57. r\_clk\_sayac <= r\_clk\_sayac + 1;
58. **end** **if**;
59. **when** GONDER =>
60. r\_tx <= r\_data(r\_data\_ind);
61. **if** r\_clk\_sayac = CLK\_BIT - 1 **then**
62. r\_clk\_sayac <= 0;
63. **if** r\_data\_ind = 7 **then**
64. r\_data\_ind <= 0;
65. r\_UART\_tx <= BITIR;
66. **else**
67. r\_data\_ind <= r\_data\_ind + 1;
68. **end** **if**;
69. **else**
70. r\_clk\_sayac <= r\_clk\_sayac + 1;
71. **end** **if**;
72. **when** BITIR =>
73. r\_tx <= '1';
74. **if** r\_clk\_sayac = CLK\_BIT - 1 **then**
75. r\_clk\_sayac <= 0;
76. r\_UART\_tx <= TAMAM;
77. **else**
78. r\_clk\_sayac <= r\_clk\_sayac + 1;
79. **end** **if**;
80. **when** TAMAM =>
81. r\_tx <= '1';
82. r\_tx\_tamam <= '1';
83. r\_UART\_tx <= BOSTA;
84. **when** **others** => **NULL**;
85. **end** **case**;
86. **end** **if;**
87. **end** **process**;
88. **end** Behavioral;

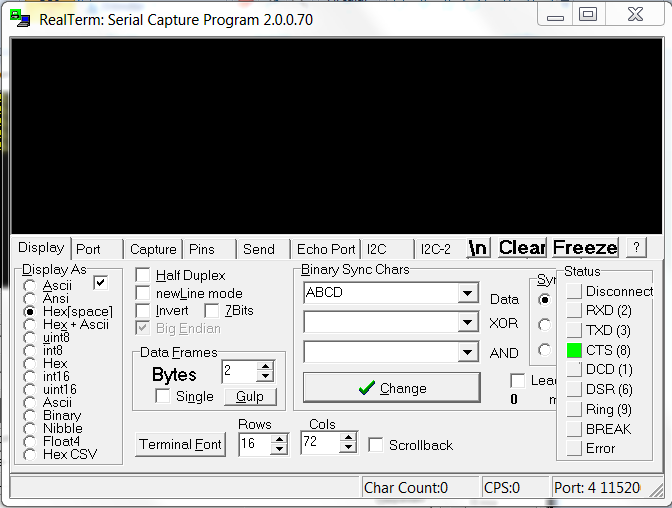
**Örnek 10.1.2**: Aşağıda UART protokoklü kullanarak data alım işlemininin yapıldığı **UART\_rx.vhd** VHDL kodu erilmiştir. **UART\_rx** varlığımıza ilişkin generic bildirimleri 7-10. satırlarda yapılmıştır. generic bildirimi içerisinde yapılan değerler kullanılarak 22. satırda her bir bit değeri için 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. **entity** UART\_rx is
6. **Generic** (
7. CLK\_FREKANS : integer := 100000000;
8. BOUDRATE : integer := 115200
9. );
10. **Port**(
11. in\_clk : **in** std\_logic;
12. in\_rst : **in** std\_logic;
13. in\_rx : **in** std\_logic;
14. out\_rx\_data : **out** std\_logic\_vector(**7** downto **0**);
15. out\_rx\_tamam : **out** std\_logic
16. );
17. **end** UART\_rx;
18. **architecture** Behavioral **of** UART\_rx **is**
19. **constant** CLK\_BIT : integer := CLK\_FREKANS / BOUDRATE + 1;
20. **type** t\_UART\_rx **is** (BOSTA, BASLA, DATA\_AL, BITIR, TAMAM);
21. **signal** r\_UART\_rx : t\_UART\_rx := BOSTA;
22. **signal** r\_clk\_sayac : integer range **0** to **CLK\_BIT - 1** := 0;
23. **signal** r\_data\_ind : integer range **0** to **7** := 0;
24. **signal** r\_data : std\_logic\_vector(**7** downto **0**) := (**others** => '0');
25. **signal** r\_rx\_tamam : std\_logic := '0';
26. **signal** r\_rx\_cnt : std\_logic\_vector(**2** downto **0**) := (**others** => '0');
27. **begin**
28. out\_rx\_data <= r\_data;
29. out\_rx\_tamam <= r\_rx\_tamam;
30. **process**(in\_clk)
31. **begin**
32. **if** in\_rst = '1' **then**
33. r\_UART\_rx <= BOSTA;
34. r\_clk\_sayac <= 0;
35. r\_data\_ind <= 0;
36. r\_data <= (**others** => '0');
37. r\_rx\_cnt <= (**others** => '0');
38. r\_rx\_tamam <= '0';
39. **elsif** rising\_edge(in\_clk) **then**
40. r\_rx\_cnt <= r\_rx\_cnt(**1** downto **0**) & in\_rx;
41. r\_rx\_tamam <= '0';
42. **case** r\_UART\_rx **is**
43. **when** BOSTA =>
44. **if** r\_rx\_cnt(**2** downto **1**) = "10" **then**
45. r\_UART\_rx <= BASLA;
46. **end** **if**;
47. **when** BASLA =>
48. **if** r\_clk\_sayac = (CLK\_BIT - 1) / 2 **then**
49. r\_clk\_sayac <= 0;
50. r\_UART\_rx <= DATA\_AL;
51. **else**
52. r\_clk\_sayac <= r\_clk\_sayac + 1;
53. **end** **if**;
54. **when** DATA\_AL =>
55. r\_data(r\_data\_ind) <= r\_rx\_cnt(2);
56. **if** r\_clk\_sayac = CLK\_BIT - 1 **then**
57. r\_clk\_sayac <= 0;
58. **if** r\_data\_ind = 7 **then**
59. r\_data\_ind <= 0;
60. r\_UART\_rx <= BITIR;
61. **else**
62. r\_data\_ind <= r\_data\_ind + 1;
63. **end if;**
64. **else**
65. r\_clk\_sayac <= r\_clk\_sayac + 1;
66. **end** **if**;
67. **when** BITIR =>
68. **if** r\_clk\_sayac = CLK\_BIT - 1 **then**
69. r\_clk\_sayac <= 0;
70. r\_UART\_rx <= TAMAM;
71. **else**
72. r\_clk\_sayac <= r\_clk\_sayac + 1;
73. **end** **if**;
74. **when** TAMAM =>
75. r\_rx\_tamam <= '1';
76. r\_UART\_rx <= BOSTA;
77. **when** **others** => **NULL**;
78. **end** **case**;
79. **end** **if**;
80. **end** **process**;
81. **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.

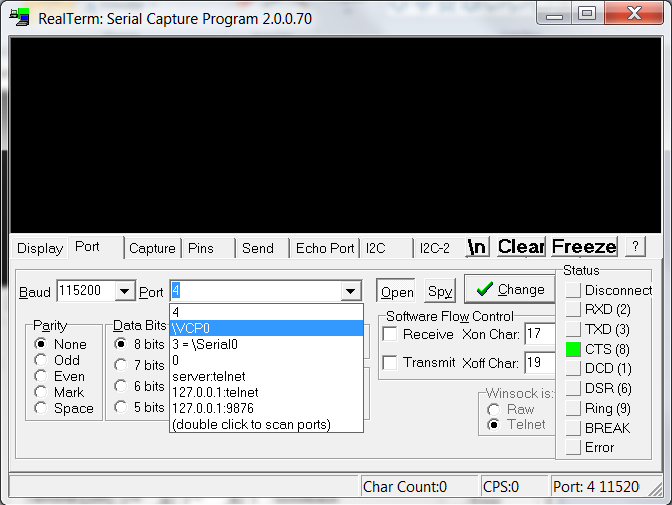
1. **library** IEEE;
2. **use** IEEE.STD\_LOGIC\_1164.ALL;
3. **entity** UART\_main **is**
4. **Port** (
5. in\_clk : **in** std\_logic;
6. in\_rst : **in** std\_logic;
7. in\_rx : **in** std\_logic;
8. out\_tx : **out** std\_logic
9. );
10. **end** UART\_main;
11. **architecture** Behavioral **of** UART\_main **is**
12. **component** UART\_tx
13. **Generic** (
14. CLK\_FREKANS : integer := 100000000;
15. BOUDRATE : integer := 115200
16. );
17. **Port**(
18. in\_clk : **in** std\_logic;
19. in\_rst : **in** std\_logic;
20. in\_tx\_basla : **in** std\_logic;
21. in\_tx\_data : **in** std\_logic\_vector(**7** downto **0**);
22. out\_tx : **out** std\_logic;
23. out\_tx\_tamam : **out** std\_logic
24. );
25. **end** **component**;
26. **component** UART\_rx
27. **Generic** (
28. CLK\_FREKANS : integer := 100000000;
29. BOUDRATE : integer := 115200
30. );
31. **Port**(
32. in\_clk : **in** std\_logic;
33. in\_rst : **in** std\_logic;
34. in\_rx : **in** std\_logic;
35. out\_rx\_data : **out** std\_logic\_vector(**7** downto **0**);
36. out\_rx\_tamam : **out** std\_logic
37. );
38. **end** **component**;
39. **type** t\_Data\_Cntrl **is** (BOSTA, DATA\_AL, DATA\_GONDER);
40. **signal** r\_Data\_Cntrl : t\_Data\_Cntrl := BOSTA;
41. **signal** r\_tx\_basla : std\_logic := '0';
42. **signal** r\_tx\_tamam : std\_logic := '0';
43. **signal** r\_rx\_tamam : std\_logic := '0';
44. **signal** r\_data : std\_logic\_vector(**7** downto **0**);
45. **signal** r\_rx\_data : std\_logic\_vector(**7** downto **0**);
46. **signal** r\_tx\_data : std\_logic\_vector(**7** downto **0**);
47. **begin**
48. **process**(in\_clk, in\_rst)
49. **begin**
50. **if** in\_rst = '1' **then**
51. r\_Data\_Cntrl <= BOSTA;
52. r\_data <= (**others** => '0');
53. **elsif** rising\_edge(in\_clk) **then**
54. r\_tx\_basla <= '0';
56. **case** r\_Data\_Cntrl **is**
57. **when** BOSTA =>
58. r\_Data\_Cntrl <= DATA\_AL;
59. **when** DATA\_AL =>
60. **if** r\_rx\_tamam = '1' **then**
61. r\_tx\_data <= r\_rx\_data;
62. r\_tx\_basla <= '1';
63. **end** **if**;
64. **when** DATA\_GONDER =>
65. **if** r\_tx\_tamam = '1' **then**
66. r\_Data\_Cntrl <= BOSTA;
67. **end** **if**;
68. **when** **others** => **NULL**;
69. **end** **case**;
70. **end** **if**;
71. **end** **process**;
72. UART\_rx\_map : UART\_rx
73. **Generic** **map**(
74. CLK\_FREKANS => 100000000, --100 MHz
75. BOUDRATE => 115200
76. )
77. **Port** **map**(
78. in\_clk => in\_clk,
79. in\_rst => in\_rst,
80. in\_rx => in\_rx,
81. out\_rx\_data => r\_rx\_data,
82. out\_rx\_tamam => r\_rx\_tamam
83. );
84. UART\_tx\_map : UART\_tx
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\_tx\_basla => r\_tx\_basla,
93. in\_tx\_data => r\_tx\_data,
94. out\_tx => out\_tx,
95. out\_tx\_tamam => r\_tx\_tamam
96. );
97. **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).



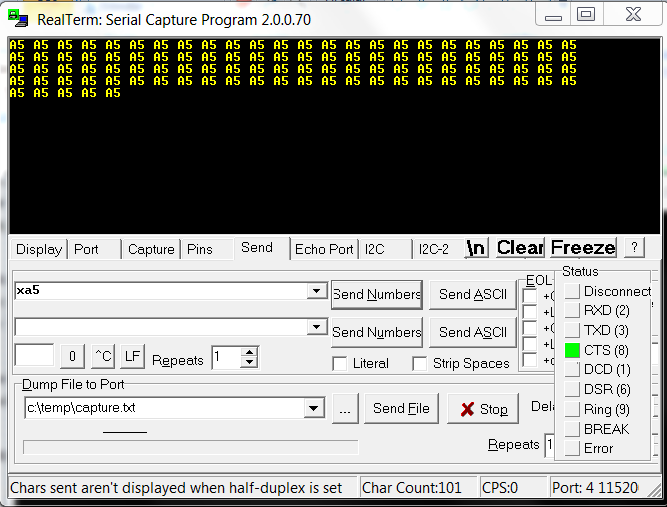
Şekil 10‑38 UART\_main modülü test işlemleri - 1

**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