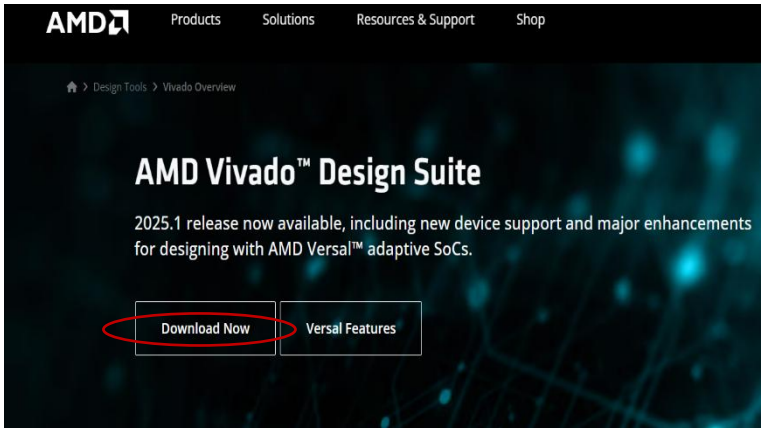
1.1 AMD Vivado İndirme ve Lisans Aktifleştirme

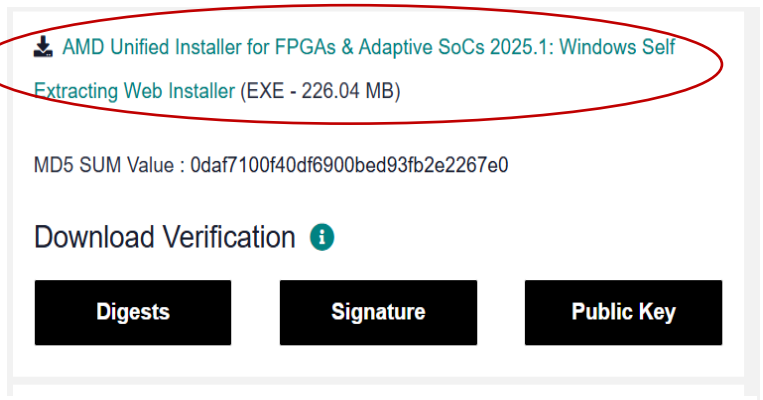
Amaç: Bu bölümde TRUE- ARTDAC7 kartını kullanabilmek için gerekli olan Xilinx Vivado yazılımını indirme ve lisans aktifleştirme hakkında bilgi verilmektedir.

AMD Vivado uygulamasını İndirmek için AMD Vivado Design Suite sitesine girilmelidir. Şekil 1.1.1'de gösterildiği gibi Download Now seçeneği seçilmelidir.



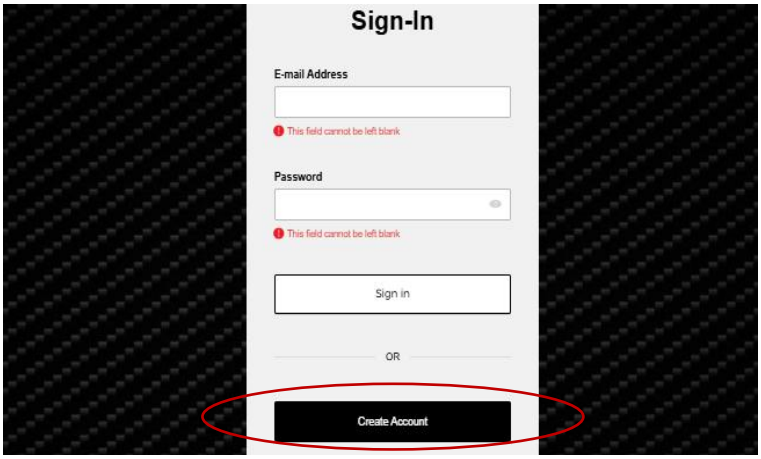
Şekil 1.1. 1 - AMD Vivado Design Suite

Sonrasında açılan sayfada kullanılan işletim sistemine özel olan yükleyicinin son versiyonu Vivado AMD Unified Installer for FPGAs & Adaptive SoCs 2025.1: Windows Self Extracting Web Installer sekmesinden indirilmelidir. Şekil 1.1.2'de gösterildiği gibi Windows işletim sistemi için olan yükleyici seçilmesi gereklidir.



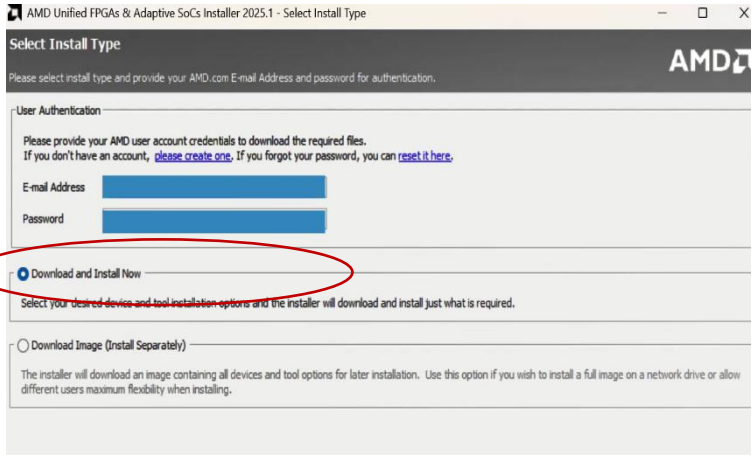
Şekil 1.1. 2 - Vivado Web Installer

Sonraki işlemlere AMD hesabı ile devam edilmektedir. Hesabınız varsa, kullanıcı bilgilerinizle giriş yapabilirsiniz, hesabınız yoksa Şekil 1.1.3'te gösterilen "Create your account" seçeneğine tıklayarak yeni bir hesap oluşturunuz.



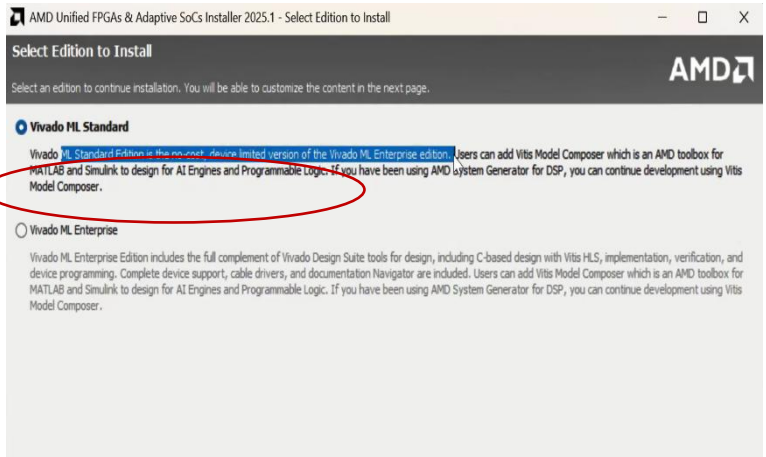
Şekil 1.1. 3 - AMD Kullanıcı Hesabı

Hesap işlemleri bittikten sonra Vivado kurulum dosyasını indirmeye başlar. İndirme ve kurulum için giriş yapmanız gerekmektedir. Ardından Download and Instal now seçeneği seçilip bir sonraki adıma geçebilirsiniz. Bir sonraki adımda Vivado'yu seçip ilerleyiniz.



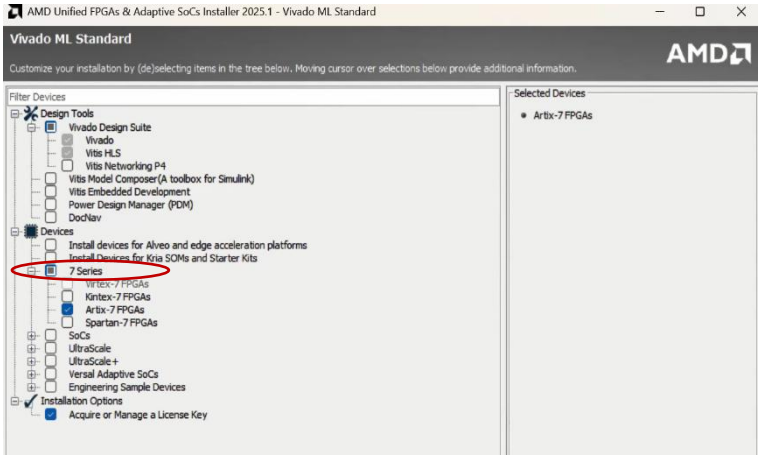
Şekil 1.1. 4 - Kullanıcı Girişi

Bir sonraki adımda Vivado ML Standart Sürümü, Vivado ML Kurumsal sürümünün ücretsiz, aygıt sınırlı versiyonu seçilmelidir.



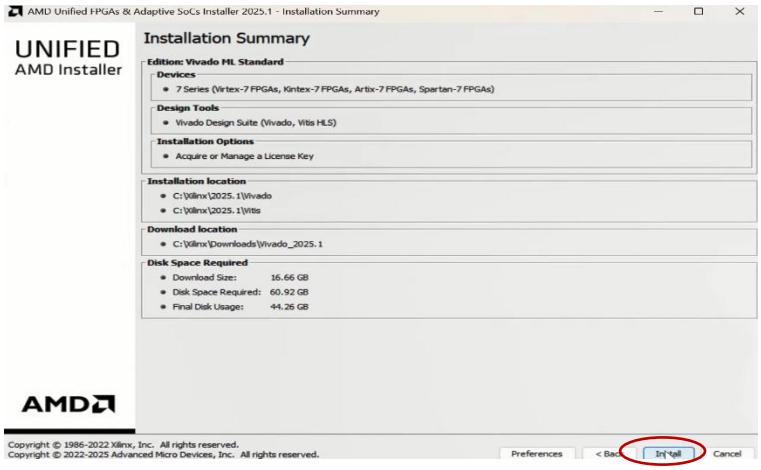
Şekil 1.1. 5 - Vivado Sürümü

Bir sonraki adımda Vivado aileleri seçilmelidir. Ardından bir sonraki adıma geçip lisans aktifleştirme işlemleri onaylanmalıdır.



Şekil 1.1. 6 - Vivado ML Standart

Son adım olarak kurulum işlemi için INSTAL seçeneği seçilmelidir.

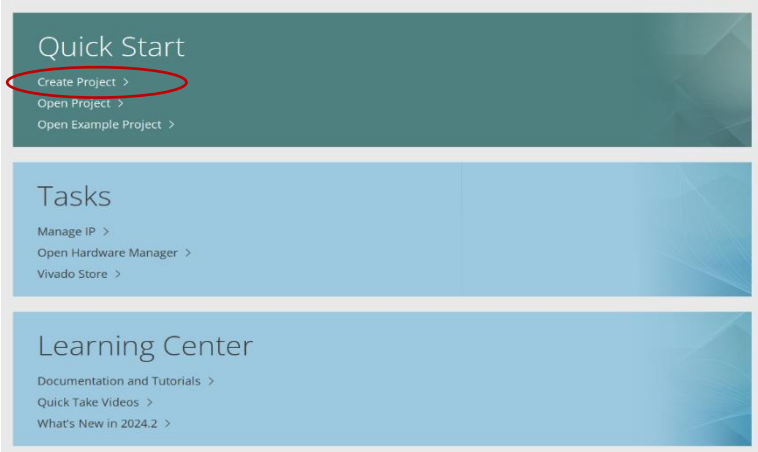


Şekil 1.1. 7 - İndirme Özeti

Bu noktadan sonra lisanslama ve kurulum işlemleri otomatik gerçekleşir ve işlem tamamlandıktan sonra AMD Vivado 2025.1 kullanılmaya hazır hale gelir.

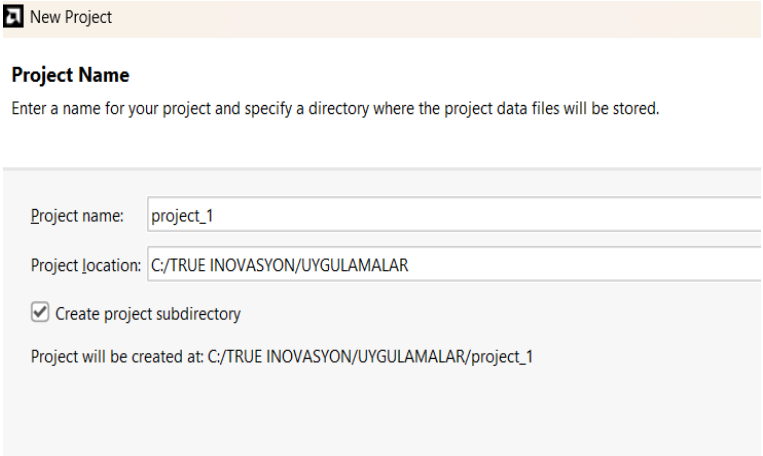
1.2 AMD Vivado Arayüz ve Proje oluşturma

Vivado Uygulaması açıldıktan sonra Şekil 1.2.1'de verilen arayüz görünmektedir. Yeni bir proje oluşturmak için Create New Project seçeneği seçilmelidir.



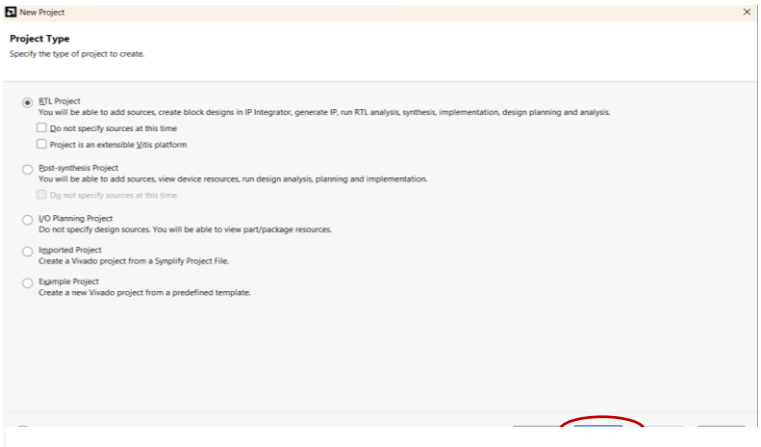
Şekil 1.2. 1 - Vivado Arayüzü

Bu adımda Projenin adı ve projenin lokasyonu girilir ve next butonuna basılır.



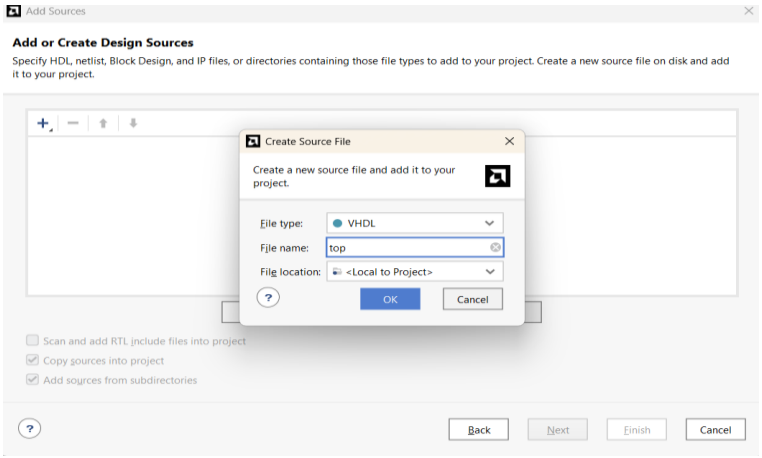
Şekil 1.2. 2 - Proje Adı ve Lokasyonu

Next butonuna basıldıktan sonra proje türü olarak RTL proje seçilir. Ve next butonuna basılır.



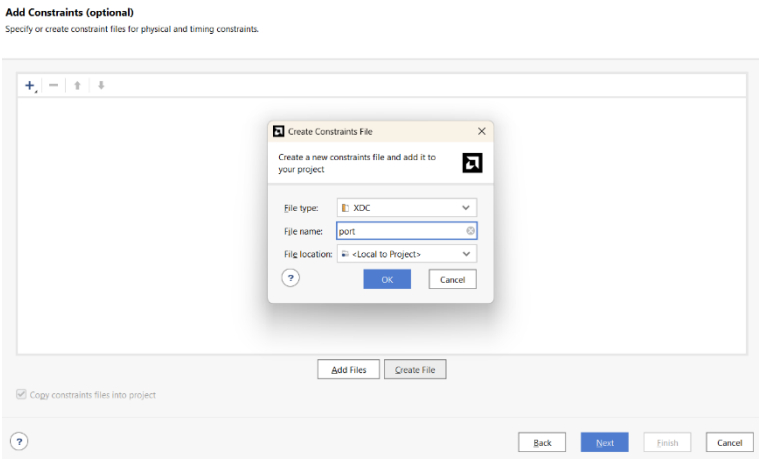
Şekil 1.2. 3 - Proje Türü

Bir sonraki adımda kaynak dosyası eklenmelidir ve hedef dil VHDL olarak seçilmelidir. Ardından next butonuna basılmalıdır.

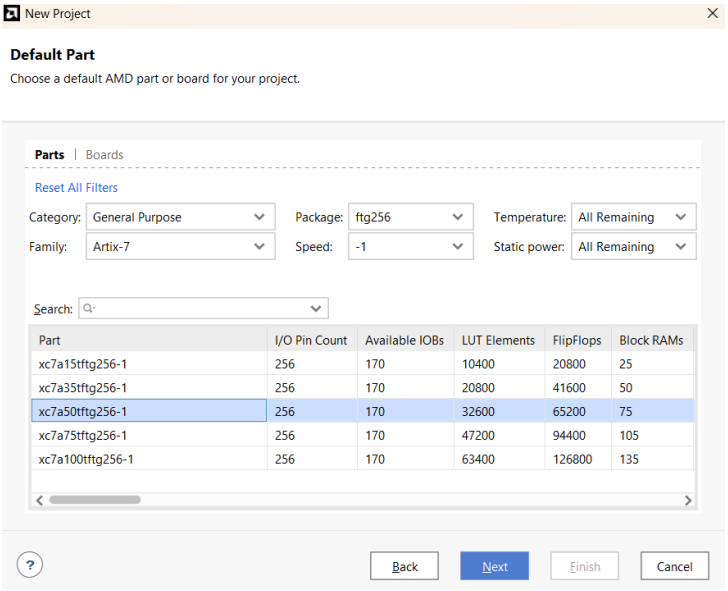


Şekil 1.2. 4 - Kaynak Dosya Oluşturma

Bir sonraki adımda port atamaları için bir dosya oluşturulmalıdır (Opsiyonel).

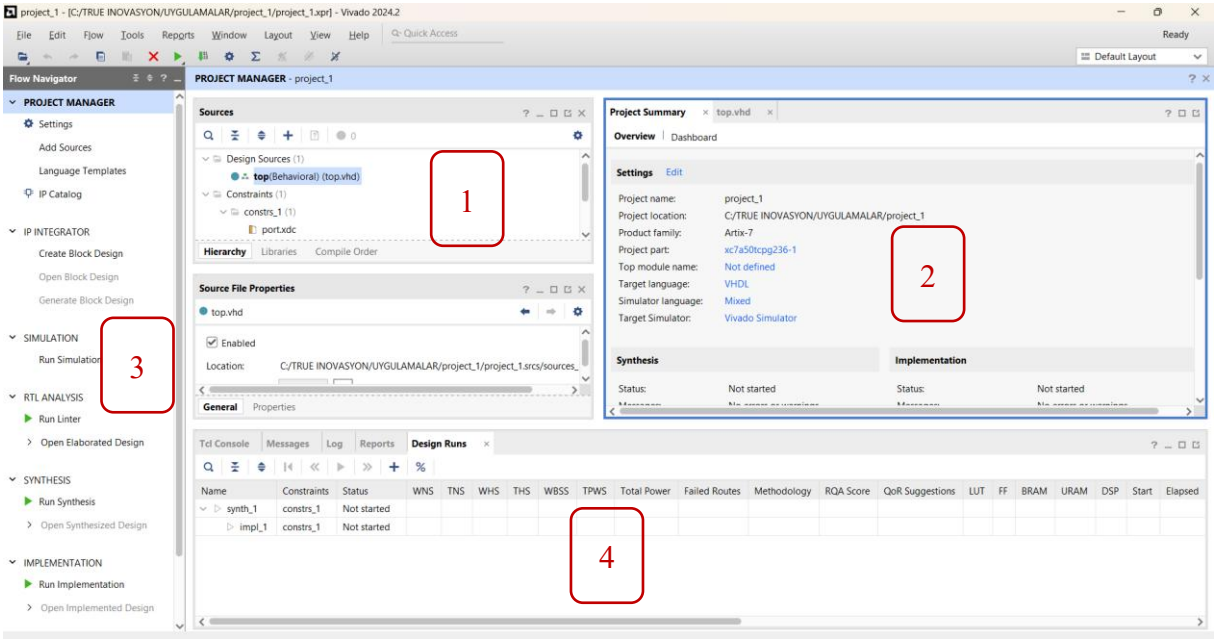


Bir sonraki adımda hedef donanım sayfası çıkar bu sayfada TRUE- ARTDAC7 için XC7A50T FTG256-1 seçeneği seçilmelidir. Bu seçenek şekil 1.2.6'da yapılan filtrelerle bulunabilir.



Şekil 1.2. 6 - TRUE- ARTDAC7 Donanım Seçimi

Bu adımdan sonra next butonuna ardından ise finish butonuna basılır ve Project Manager açılır. Açılan sayfa şekil 1.2.7'de gösterilmektedir.



Şekil 1.2. 7 - Proje Yönetici Arayüzü

1. **Sources:** Proje kaynaklarının göründüğü paneldir. Buradan kaynak ekleme ve çıkarma işlemlerinin yönetildiği paneldir.
2. **Editör:** HDL kaynak kodunun yazıldığı paneldir. Aynı zamanda proje RTL şematik dosyalarının gibi uygulamaların da yönetildiği ana paneldir.
3. **Project Manager:** Simülasyon'nun sentezlemenin implementation ve generate bitstream işlemlerin yönetildiği paneldir.
4. **Status:** Hata mesajlarının, TCL Console, Log,reports, Design Runs I/O port atamalarının yönetildiği paneldir.

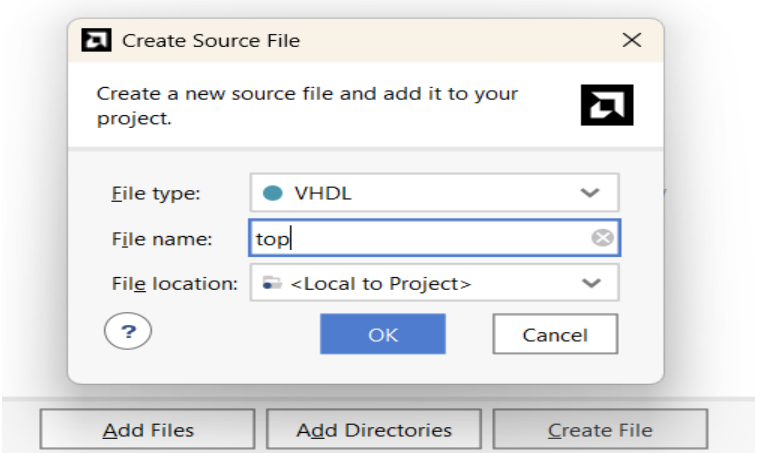
UYGULAMAR

2.1 Led Yakma

Amaç: Bu uygulamada TRUE- ARTDAC7 kartı üzerinde bulunan led'in sürekli aktif olması sağlanacaktır.

Led yakma uygulamasının yapılabilmesi için yeni proje açılması gerekmektedir.

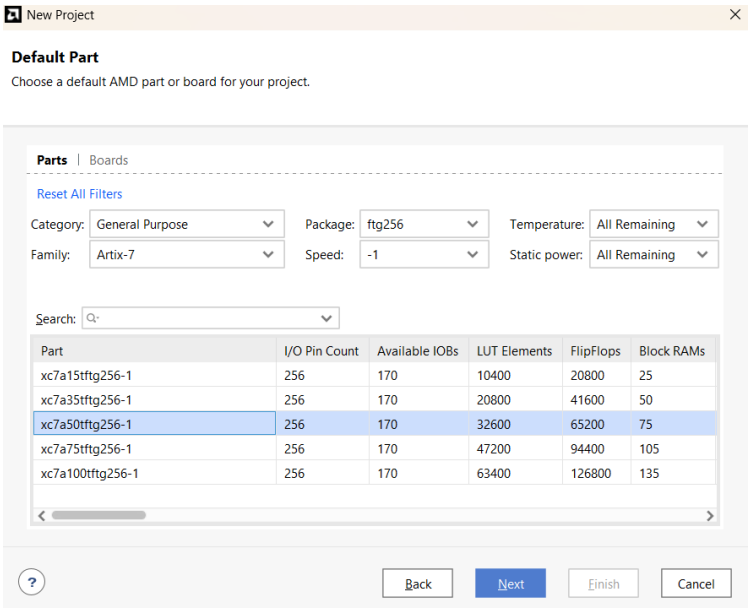
Açılan yeni projeye isim verildikten sonra şekil 2.1.1'de gösterildiği gibi hedef dili VHDL olarak belirtmek gerekir ardından yeni kaynak dosyasına bir isim verilmelidir.



Şekil 2.1. 1 – Kaynak Dosya Oluşturma

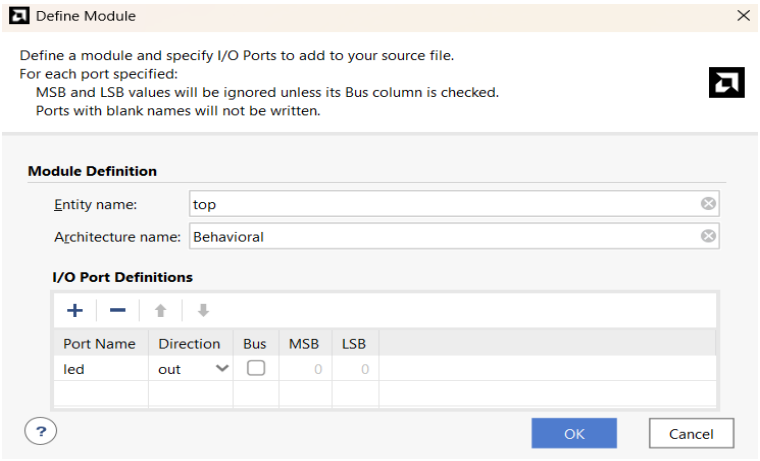
Kaynak dosyası oluşturulduktan sonra constraints dosyası oluşturulmadan devam edilmelidir. Bu dosya kaynak dosyası sentezlendikten sonra manuel olarak oluşturulacaktır.

TRUE-ARTDAC7 FPGA Geliştirme kartının üzerindeki FPGA çipi şekil 2.1.2'de gösterildiği gibi (xc7a50tftg256-1) seçilmelidir.



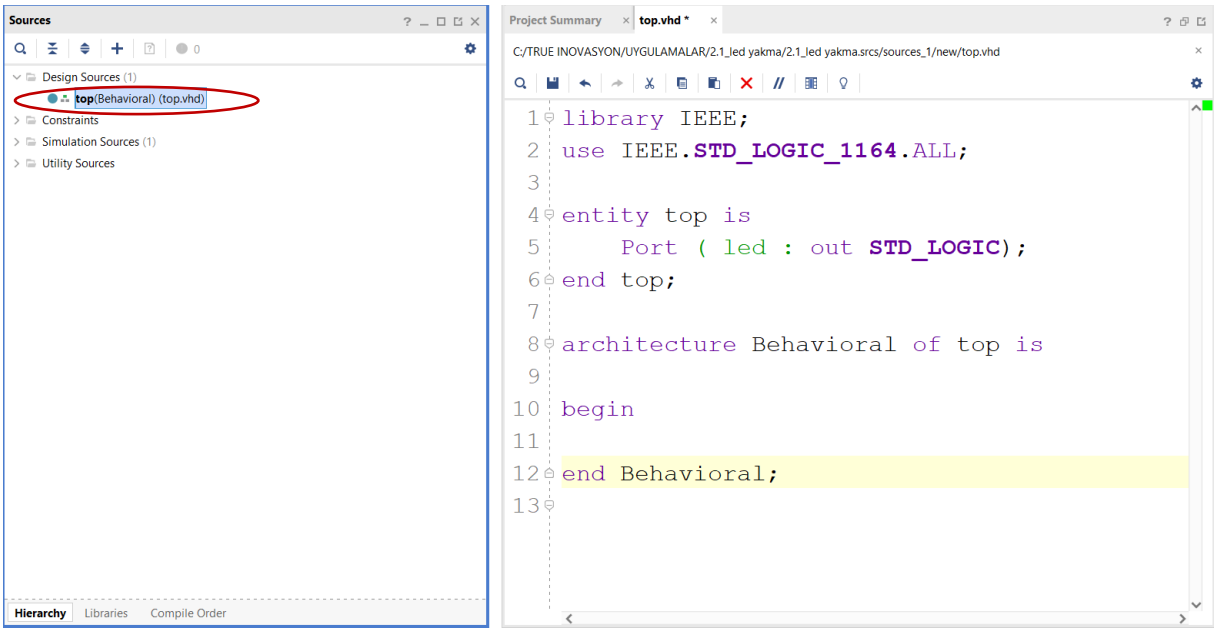
Şekil 2.1. 2- TRUE- ARTDAC7 Donanım Seçimi

Donanım seçildikten sonra Finish butonuna basarak proje oluşturulur ve şekil 2.1.3'te gösterildiği gibi led adından 1 bitlik out sinyali oluşturulur. Bu sayede hazır bir kalıp oluşturulur.



Şekil 2.1. 3 - Port Oluşturma

Açılan ana ekranda şekil 2.1.4' te belirtildiği gibi Design Sources altında oluşturduğumuz top kısmına iki sefer tıklanmasıyla beraber tasarlanan top modülümüz karşımıza çıkacaktır.



Şekil 2.1. 4 - Design Sources

Port kısmında bulunan sinyalin özellikleri şu şekildedir;

In: Sinyalin input olduğu belirtir.

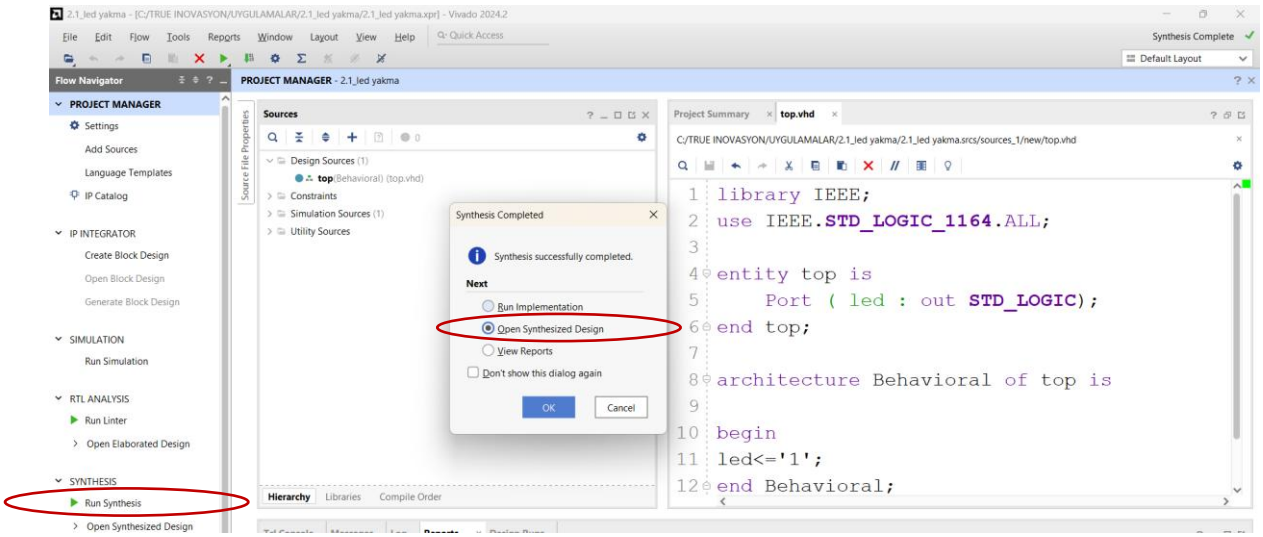
Std_logic: standart Logic'in kısaltılmasıdır. Bu sinyal yalnızca '0' ve '1' değerlerini alabilir.

Bu bilgiyle kodda görülen led: out std_logic; ifadesi led adında bir sinyalin output olduğu ve tek bitlik bir sinyal olduğu anlaşılmaktadır.

Hazır olarak oluşturulan entity kısmının ardından yapılması gereken mantık kısmını oluşturmaktır. Mantık kısmı VHDL dilinde begin ve architecture arasında tanımlanır, tasarlanması planlanan led yakma uygulaması için entity kısmında tanımlanan led adında sinyale '1' değerinin atanması gerekmektedir bu sayede led'in FPGA çipine bağlı olan pinine sürekli olarak enerji verilerek yanması sağlanacaktır.

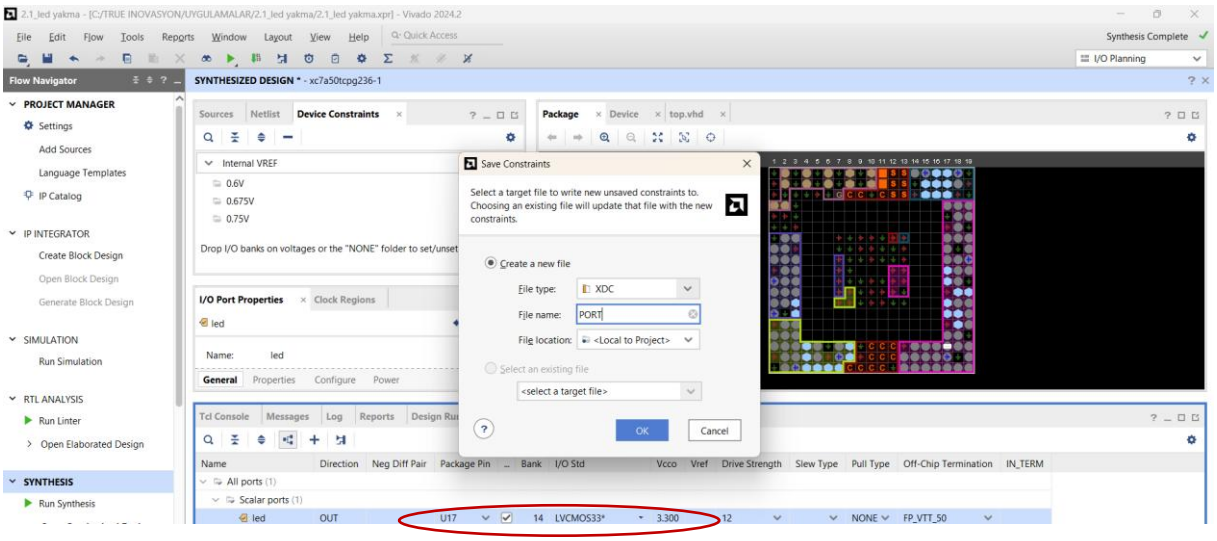
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity LED_Yakma is
    Port ( o_led : out  STD_LOGIC);
end LED_Yakma;
architecture Behavioral of LED_Yakma is
begin
led <= '1' ;
end Behavioral;
```

Kısıt dosyasının oluşturulması için tasarlanan yapının kaydedildikten sonra sentezlenmesi gerekmektedir şekil 2.1.5' te belirtilen Run synthesis seçeneği basılarak kodun sentezlenme işlemini başlatmak gerekir. Sentezleme işlemi tamamlandıktan sonra open synthesis seçeneği seçilerek bir sonraki adıma geçilmesi gerekir.



Şekil 2.1. 5 Run Synthesis

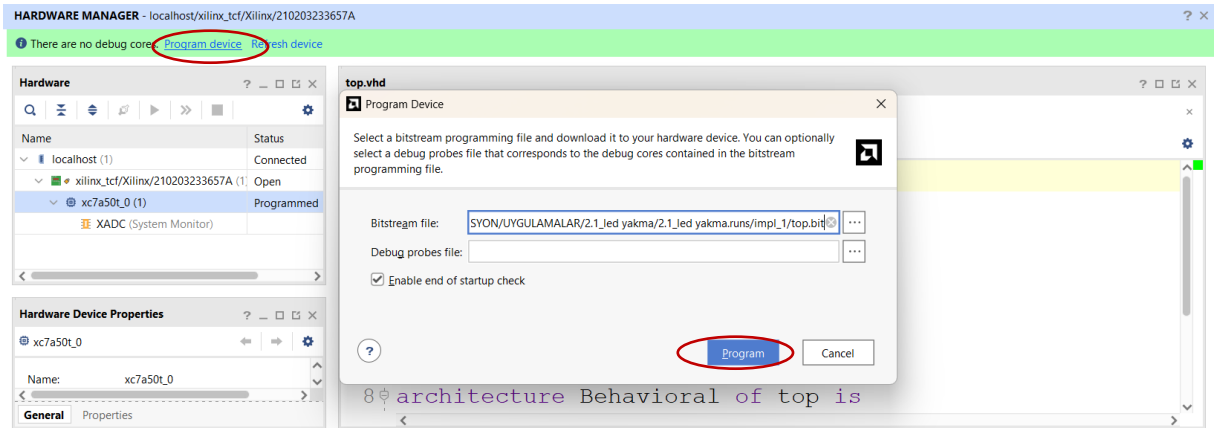
Açılan Synthesized Design arayüzünden sonra Window kısmından I/O ports seçeneği seçilip scalar ports kısmından pin ataması gerekmektedir. TRUE- ARTDAC7 kartında bulunan sağdan ilk led olan T2 pinine atama işlemi gerçekleştirilmesi gerekmektedir atama işlemi tamamlandıktan sonra bu işlemin kaydedilmesi gerekmektedir şekil 2.1.6' da belirtildiği gibi XDC dosyası isimlendirilip ok seçeneğinin seçilmesi gerekmektedir.



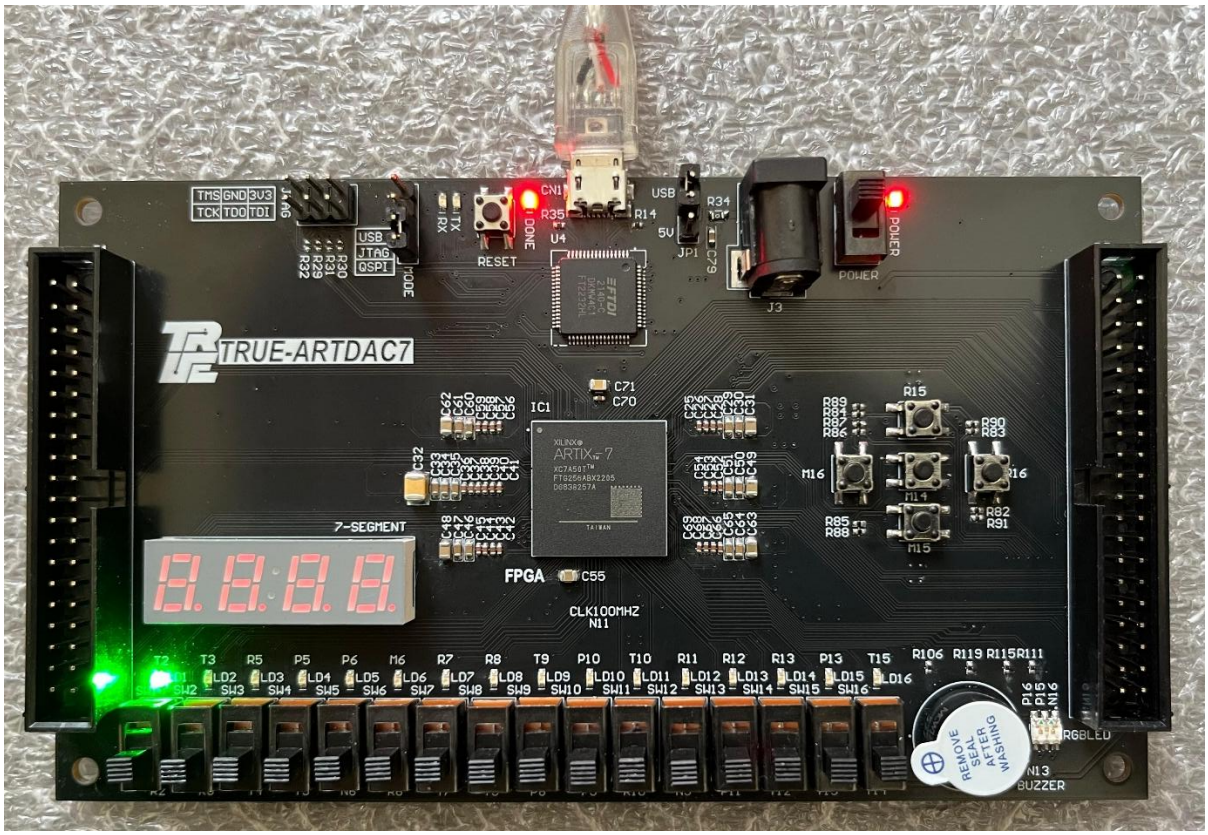
Şekil 2.1. 6 - XDC Dosyasının Oluşturulması

Bu işlemler tamamlandıktan sonra Project Manager bölümünde bulunan Generate Bitstream seçeneği seçilmesi gerekmektedir. Bitsream dosyası oluşturulduktan sonra kartın bilgisayara bağlanması gerekmektedir. Karta enerji verildikten sonra Open Hardware Manager

seçeneğinin seçilmesi gerekmektedir. Oluşan arayüz şekil 2.1.7'de verilmektedir bu adımda son olarak program device seçeneği seçilip dosyanın karta yüklenmesi gerekir.



Şekil 2.1. 7 - Oluşturulan Kodun Karta Yüklenmesi



Şekil 2.1. 8 – TRUE- ARTDAC7 Geliştirme Kartı Üzerinde Tasarımın Görüntüsü

2.2 Anahtar (Switch) ile LED Kontrolü Uygulaması

Amaç: Bu uygulama, TRUE- ARTDAC7 FPGA geliştirme kartı üzerindeki anahtar (Switch) girişleri ile LED çıkışlarının doğrudan nasıl kontrol edileceğini gösteren basit bir kombinasyonel devrenin tasarımını ve uygulamasını anlatmaktadır. Temel giriş/çıkış (I/O) eşleme mantığı pekiştirilecektir.

Teorik Altyapı ve Çalışma Prensibi

1. Giriş/Çıkış Birimleri (I/O)

TRUE- ARTDAC7 FPGA geliştirme kartı, kullanıcı etkileşimi için anahtarlar (giriş) ve durumu göstermek için LED'ler (çıkış) gibi çevre birimlerine sahiptir.

Anahtarlar (Switches): Kullanıcıdan veri almak için kullanılır. Genellikle HIGH (1) veya LOW (0) lojik seviyesinde bir sinyal üretirler.

LED'ler (Işık Yayan Diyotlar): Devrenin lojik durumunu görsel olarak göstermek için kullanılır. LED'in yanması HIGH (1), sönmesi ise LOW (0) lojik seviyesi olarak ayarlanır.

VHDL kodunun oluşturulması;

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity top is
   Port ( i_s1 : in STD_LOGIC_VECTOR(7 downto 0 );
          o_led : out STD_LOGIC_VECTOR(7 downto 0 ));
end top;
architecture Behavioral of top is
begin
o_led <= i_s1 ;
end Behavioral;
```

Kısıt dosyasının oluşturulması;

```
set_property -dict { PACKAGE_PIN R2   IOSTANDARD LVCMOS33 } [get_ports {i_s1[0]}]
set_property -dict { PACKAGE_PIN T3   IOSTANDARD LVCMOS33 } [get_ports {i_s1[1]}]
set_property -dict { PACKAGE_PIN R5   IOSTANDARD LVCMOS33 } [get_ports {i_s1[2]}]
set_property -dict { PACKAGE_PIN T5   IOSTANDARD LVCMOS33 } [get_ports {i_s1[3]}]
set_property -dict { PACKAGE_PIN N6   IOSTANDARD LVCMOS33 } [get_ports {i_s1[4]}]
set_property -dict { PACKAGE_PIN R6   IOSTANDARD LVCMOS33 } [get_ports {i_s1[5]}]
set_property -dict { PACKAGE_PIN T7   IOSTANDARD LVCMOS33 } [get_ports {i_s1[6]}]
set_property -dict { PACKAGE_PIN T7   IOSTANDARD LVCMOS33 } [get_ports {i_s1[7]}]

set_property -dict { PACKAGE_PIN T2   IOSTANDARD LVCMOS33 } [get_ports {o_led[0]}]
set_property -dict { PACKAGE_PIN R3   IOSTANDARD LVCMOS33 } [get_ports {o_led[1]}]
set_property -dict { PACKAGE_PIN T4   IOSTANDARD LVCMOS33 } [get_ports {o_led[2]}]
set_property -dict { PACKAGE_PIN P5   IOSTANDARD LVCMOS33 } [get_ports {o_led[3]}]
set_property -dict { PACKAGE_PIN M6   IOSTANDARD LVCMOS33 } [get_ports {o_led[4]}]
set_property -dict { PACKAGE_PIN P6   IOSTANDARD LVCMOS33 } [get_ports {o_led[5]}]
set_property -dict { PACKAGE_PIN R7   IOSTANDARD LVCMOS33 } [get_ports {o_led[6]}]
set_property -dict { PACKAGE_PIN R8   IOSTANDARD LVCMOS33 } [get_ports {o_led[7]}]
```

2.3 Mantık Kapıları ve Çoklayıcı (MUX) Uygulaması

Amaç: Bu uygulamada, TRUE- ARTDAC7 FPGA geliştirme kartı üzerindeki anahtarları (Switch) giriş olarak kullanarak, temel Mantık Kapılarının (AND, OR, NOT, NAND, XOR, XNOR) ve bir 2'ye 1 Çoklayıcının (Multiplexer - MUX) çalışma prensiplerini VHDL ile nasıl modelleneceğini göstermek amaçlanmaktadır. Temel kombinasyonel mantık tasarımı becerileri pekiştirilecektir.

Teorik Altyapı ve Çalışma Prensibi

Kombinasyonel devreler, çıkışın yalnızca o anki giriş değerlerine bağlı olduğu devrelerdir. Bu uygulamada, temel mantık fonksiyonları gerçekleştirilecektir.

Fonksiyon	Mantık İfadesi	Açıklama
AND	C = A · B	İki girişin de '1' olması durumunda çıkış '1' olur.
NAND	C = not (A · B)	AND kapısının tersidir. Yalnızca A ve B '1' iken çıkış '0' olur.
OR	C = A + B	Girişlerden en az birinin '1' olması durumunda çıkış '1' olur.
NOR	C = not (A + B)	OR kapısının tersidir. Yalnızca A ve B '0' iken çıkış '1' olur.
NOT	C = not A	Girişi tersler. Giriş '1' ise çıkış '0', '0' ise '1' olur.
XOR	C = A ⊕ B	Girişler farklı ise çıkış '1' olur.
XNOR	C = not (A ⊕ B)	Girişler aynı ise çıkış '1' olur.

Fonksiyon	Mantık İfadesi	Açıklama
2'ye 1 MUX	$C = (S \cdot A) + (\overline{S} \cdot B)$	Seçici (S) sinyaline göre iki girişten (A,B) birini çıkışa (C) yönlendirir. S=1 iken A seçilir, S=0 iken B seçilir

VHDL kodunun oluşturulması;

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity top is
    Port ( i_s1 : in STD_LOGIC_VECTOR(7 downto 0 );
           o_led : out STD_LOGIC_VECTOR(7 downto 0 ));
end top;
architecture Behavioral of top is
begin
-- AND / NAND (A=i_s1(0), B=i_s1(1))
    o_led(0) <= i_s1(0) and  i_s1(1);           -- AND
    o_led(1) <= not (i_s1(0) and i_s1(1));    -- NAND
  -- OR / NOR (A=i_s1(2), B=i_s1(3))
   o_led(2) <= i_s1(2) or i_s1(3);          -- OR
   o_led(3) <= not (i_s1(2) or i_s1(3));     -- NOR
  -- NOT (A=i_s1(4))
  o_led(4) <= not i_s1(4);
  -- XOR / XNOR (A=i_s1(6), B=i_s1(7))
  o_led(5) <= i_s1(6) xor i_s1(7);          -- XOR
  o_led(6) <= not (i_s1(6) xor i_s1(7));    -- XNOR
  -- 2:1 MUX (S=i_s1(5)) -> S=1 iken A=i_s1(7), S=0 iken B=i_s1(6)
  with i_s1(5) select
    o_led(7) <= i_s1(7) when '1',
                i_s1(6) when '0';
end Behavioral;
```

Kısıt dosyasının oluşturulması;

```
set_property -dict { PACKAGE_PIN R2   IOSTANDARD LVCMOS33 } [get_ports {i_s1[0]}]
set_property -dict { PACKAGE_PIN T3   IOSTANDARD LVCMOS33 } [get_ports {i_s1[1]}]
set_property -dict { PACKAGE_PIN R5   IOSTANDARD LVCMOS33 } [get_ports {i_s1[2]}]
set_property -dict { PACKAGE_PIN T5   IOSTANDARD LVCMOS33 } [get_ports {i_s1[3]}]
set_property -dict { PACKAGE_PIN N6   IOSTANDARD LVCMOS33 } [get_ports {i_s1[4]}]
set_property -dict { PACKAGE_PIN R6   IOSTANDARD LVCMOS33 } [get_ports {i_s1[5]}]
set_property -dict { PACKAGE_PIN T7   IOSTANDARD LVCMOS33 } [get_ports {i_s1[6]}]
set_property -dict { PACKAGE_PIN T7   IOSTANDARD LVCMOS33 } [get_ports {i_s1[7]}]

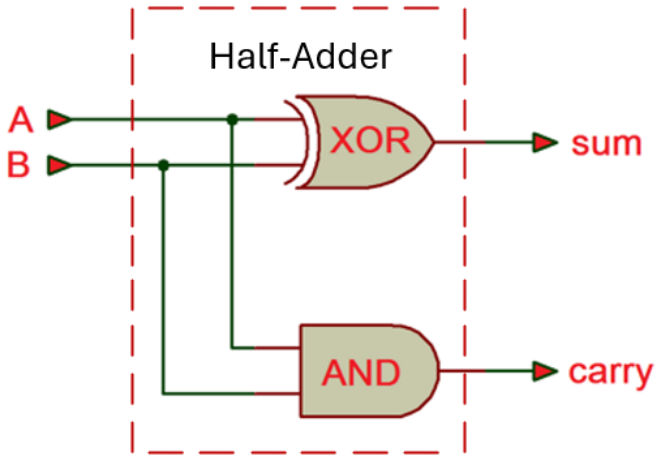
set_property -dict { PACKAGE_PIN T2   IOSTANDARD LVCMOS33 } [get_ports {o_led[0]}]
set_property -dict { PACKAGE_PIN R3   IOSTANDARD LVCMOS33 } [get_ports {o_led[1]}]
set_property -dict { PACKAGE_PIN T4   IOSTANDARD LVCMOS33 } [get_ports {o_led[2]}]
set_property -dict { PACKAGE_PIN P5   IOSTANDARD LVCMOS33 } [get_ports {o_led[3]}]
set_property -dict { PACKAGE_PIN M6   IOSTANDARD LVCMOS33 } [get_ports {o_led[4]}]
set_property -dict { PACKAGE_PIN P6   IOSTANDARD LVCMOS33 } [get_ports {o_led[5]}]
set_property -dict { PACKAGE_PIN R7   IOSTANDARD LVCMOS33 } [get_ports {o_led[6]}]
set_property -dict { PACKAGE_PIN R8   IOSTANDARD LVCMOS33 } [get_ports {o_led[7]}]
```

2.4 Yarım ve Tam Toplayıcı Uygulaması

Amaç: Bu uygulamada TRUE- ARTDAC7 FPGA geliştirme kartı üzerindeki anahtarları (Switch) giriş olarak kullanarak, Half Adder (yarım toplayıcı) ve Full Adder (tam toplayıcı) devrelerinin VHDL ile modellenmesi ve kart üzerindeki LED'ler üzerinden sonuçların gözlemlenmesi amaçlanmaktadır. Bu sayede Temel kombinasyonel mantık tasarımı becerileri pekiştirilecektir.

Teorik Altyapı ve Çalışma Prensibi

Half Adder (Yarım Toplayıcı): Half Adder, iki adet 1 bitlik ikili sayının toplanmasını gerçekleştiren en temel toplama devresidir. Bu devre iki girişe sahiptir (A ve B) Toplama işlemi sonucunda iki çıkış üretilir Sum (S) ve Carry (C). Sum çıkışı, girişlerin XOR (özel veya) işlemi ile elde edilirken, Carry çıkışı girişlerin AND (ve) işlemi ile oluşturulur. Half adder doğruluk tablosu şekil 2.4.1'de verilmiştir.



A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

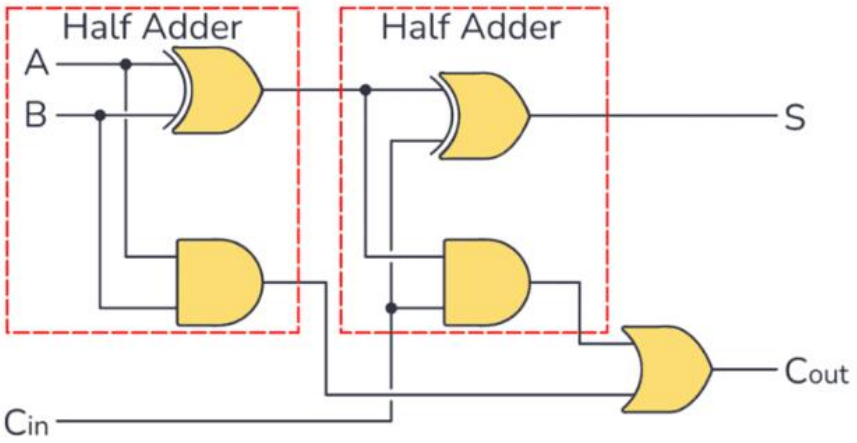
Şekil 2.4. 1 Half Adder Doğruluk Tablosu

Half-adder VHDL kodunun oluşturulması;

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity half_adder is
port (
   i_A    : in  std_logic;
   i_B    : in  std_logic;
   o_sum   : out std_logic;
   o_carry: out std_logic
);
end half_adder;
architecture Behavioral of half_adder is
begin
   o_sum  <=    i_A    xor    i_B ;
  o_carry <=     i_A    and    i_B;
end Behavioral;
```

Full Adder (Tam Toplayıcı): Full Adder, üç adet 1 bitlik ikili sayının toplanmasını gerçekleştiren bir toplama devresidir. Bu devre iki ana giriş (A ve B) ile bir önceki basamaktan gelen elde bilgisi olan Carry-in (Cin) girişine sahiptir. Toplama işlemi sonucunda Sum (S) ve bir sonraki basamağa iletilen Carry-out (Cout) olmak üzere iki çıkış üretilir.

Sum çıkışı A, B ve Cin girişlerinin XOR işlemleri ile elde edilirken Carry-out çıkışı bu girişlerin AND ve OR işlemlerinin birleşimi ile oluşturulur. Full Adder doğruluk tablosu Şekil 2.4.2'de verilmiştir.



Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Şekil 2.4. 2 Full Adder Doğruluk Tablosu

N-bit Toplayıcı (N-bit Adder): N-bit toplayıcı, iki adet N bitlik ikili sayıyı toplayan ve sonucunda N bitlik toplam ile taşma (carry-out) üreten yapıdır. Bu uygulamada, 1 bitlik Full Adder blokları ardışık bağlanarak (Ripple Carry yapısı) N-bit toplama işlemi gerçekleştirilmiştir. Her basamağın elde çıkışı bir sonraki basamağın elde girişine aktarılır. FPGA kartında girişler switch'ler üzerinden verilmekte, toplam ve taşma sonucu LED'ler üzerinden gözlemlenmektedir.

Full-adder VHDL kodunun oluşturulması;

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_adder is
port (
   i_A   : in  std_logic;
   i_B   : in  std_logic;
   i_Cin  : in  std_logic;
   o_sum  : out std_logic;
   o_Cout : out std_logic
);
end full_adder;
architecture Behavioral of full_adder is
   -- COMPONENT ekleme
   component half_adder is
      port (
         i_A    : in  std_logic;
         i_B    : in  std_logic;
         o_sum   : out std_logic;
         o_carry : out std_logic
      );
   end component;
   signal s_ilk_sum   : std_logic := '0';
   signal s_ilk_carry : std_logic := '0';
   signal s_ikinci_carry: std_logic := '0';
```

```
begin
   birinci_half_adder : half_adder
      port map(
         i_A    => i_A,
         i_B    => i_B,
         o_sum  => s_ilk_sum,
         o_carry => s_ilk_carry
      );
   ikinci_half_adder : half_adder
      port map(
         i_A    => s_ilk_sum,
         i_B    => i_Cin,
         o_sum  => o_sum,
         o_carry => s_ikinci_carry );
   o_Cout <= s_ilk_carry or s_ikinci_carry;
end Behavioral;
```

2.5 Clock Divider (Saat Bölücü) Uygulaması

Amaç: Bu uygulama, TRUE- ARTDAC7 FPGA geliştirme kartının yüksek frekanslı dahili saat sinyalini (100MHz) bölerek daha düşük bir frekans (1Hz) elde etmeyi amaçlamaktadır. Elde edilen düşük frekanslı sinyal, bir LED'i saniyede bir yanıp sönecek şekilde kontrol etmek için kullanılacaktır. Bu proje, sekansiyel (sıralı) mantık tasarımını ve sayaç (counter) kullanımını pekiştirecektir.

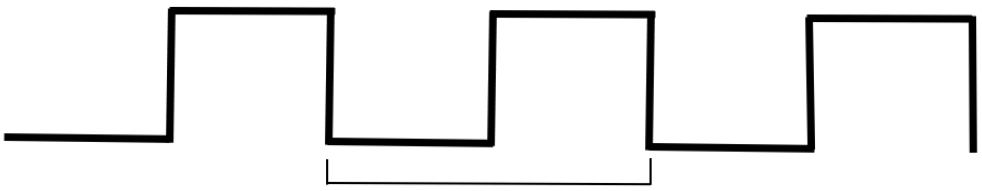
Teorik Altyapı ve Çalışma Prensibi

1. Periyot ve Frekans Tanımı

Frekans (F): Bir sinyalin saniyedeki tekrar sayısıdır. Birimi Hertz (Hz) olarak ifade edilir.

Periyot (T): Bir sinyalin bir tam çevrimini tamamlaması için geçen süredir. Periyot ve frekans arasındaki ilişki bu şekildedir;

$$T = \frac{1}{F}$$



1 Periyot

2. Gerekli Bölme Faktörünün Hesaplanması: TRUE- ARTDAC7 Geliştirme Kartı üzerinde bulunan 100 MHz'lik osilatör, kartın temel saat sinyalini sağlar.

Bu, saniyede 100,000,000 (yüz milyon) kare dalga oluşturduğu anlamına gelir.

TRUE- ARTDAC7 kartının periyotu ise bu şekilde hesaplanabilir;

$$T_{\text{clk}} = \frac{1}{100\ \text{MHz}} = 0.00000001\ \text{saniye}\ (10\ \text{ns})$$

İhtiyacımız olan 1Hz frekanslı bir sinyal oluşturabilmek için periyodu 1 saniye olan bir kare dalga elde etmek gerekir. 1 saniyelik periyot için, sinyalin 0.5saniye LOW (0) ve 0.5 saniye HIGH (1) kalması gerekmektedir.

Buna göre, sayacımızın 0.5 saniyelik bir zaman aralığı oluşturmak için kaç adet 100 MHz'lik clock vuruşu sayması gerektiğini bu şekilde hesaplayabiliriz;

$$\text{Sayaç Değeri (MAX)} = \frac{\text{İstenen Yarım Periyot Süresi}}{\text{Giriş Clock Periyodu}} = \frac{0.5 \text{ saniye}}{0.00000001 \text{ saniye}} = 50{,}000{,}000$$

Bu hesaplama sonucunda, sayacın 0'dan 49,999,999'a kadar sayması gerektiği belirlenir. 50,000,000 adet sayım tamamlandığında, çıkış sinyalinin lojik durumu terslenerek 0.5 saniye sonra 1 Hz'lik kare dalga üretimi sağlanır.

Kısıt dosyasının oluşturulması;

```
set_property -dict { PACKAGE_PIN N11   IOSTANDARD LVCMOS33 } [get_ports clk]
set_property -dict { PACKAGE_PIN P15  IOSTANDARD LVCMOS33 } [get_ports o_led]
```

VHDL kodunun oluşturulması

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
generic (
 max : integer := 100_000_000 );
   Port ( clk : in STD_LOGIC;
          o_led : out STD_LOGIC);
end top;
architecture Behavioral of top is
signal sayici_1hz : integer range 0 to max := 0;
signal clk_1hz: std_logic := '0';
begin
process(clk)
begin
   if rising_edge(clk) then
      if sayici_1hz = max - 1 then
         sayici_1hz <= 0;
         clk_1hz <= not clk_1hz;
      else
         sayici_1hz <= sayici_1hz + 1;
      end if; end if;
end process;
o_led<= clk_1hz;
end Behavioral;
```

2.6 7-Parçalı Gösterge (7-segment) Uygulaması

Bu uygulama, TRUE- ARTDAC7 FPGA geliştirme kartı üzerindeki anahtar (Switch) girişlerini kullanarak, 7-parçalı göstergeye (7-Segment Display) istenen rakamları veya karakterleri kodlamak ve göstermek amaçlanmaktadır. Bu sayede, temel kombinasyonel kod çözücü (Decoder) mantığı ve 7-segment Ortak Anot sürüş prensibi pekiştirilecektir.

Teorik Altyapı ve Çalışma Prensibi

1. **Gösterge Yapısı ve Lojik Seviyeler**

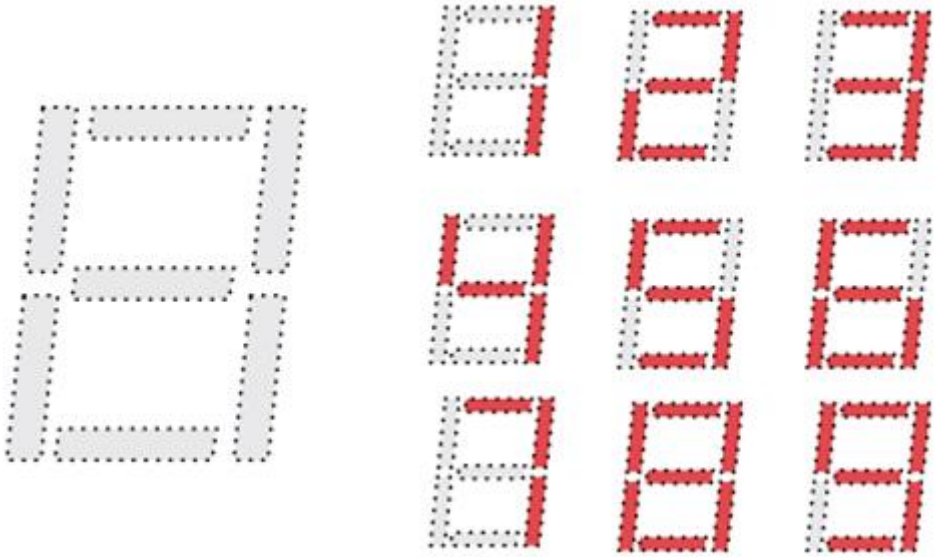
TRUE- ARTDAC7 FPGA Geliştirme kartı üzerinde bulunan 7-parçalı LED göstergeler, Ortak Anot (Common Anode) prensibiyle çalışır.

Anot Girişleri (Digit Enable): Kart üzerindeki Ortak anot girişleri, PNP transistörler ile kontrol edilmektedir. İlgili haneyi etkinleştirmek (seçmek) amacıyla Transistörün Base'ine LOW (0) lojik seviyesi verildiğinde sinyal üretir ve hane aktif olur.

Katot Çıkışları (Segment Kontrol): İlgili LED'in yanması için hem Anot değerinin LOW (0) (hane seçimi) hem de ilgili Katot değerinin LOW (0) (segment seçimi) olması gerekir Bu yapı nedeniyle, Ortak Anot göstergelerde bir segmentin yanması için çıkışa (0) lojik seviyesi atanması gerekir.

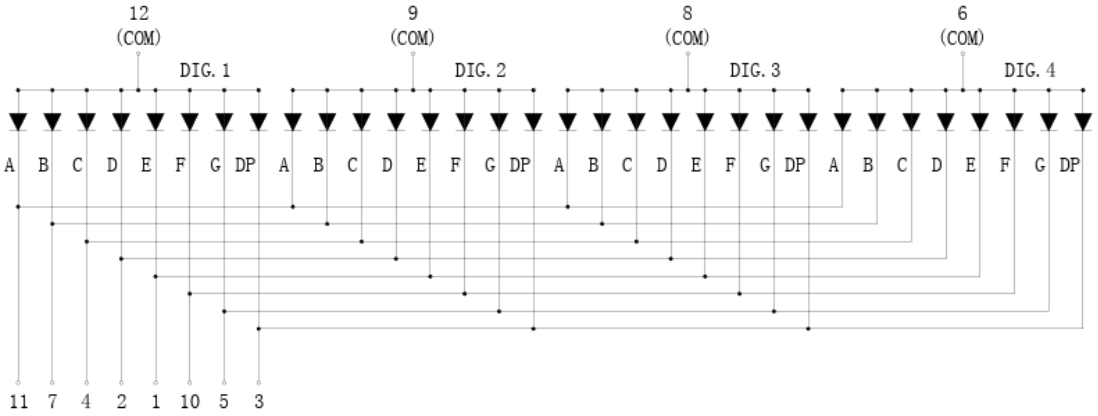
2. **Segment Kodlaması ve Haneler**

7-Segmentli ekran ve ondalık hanelere karşılık gelen aydınlatma modelleri Şekil 2.6.1'de verilmiştir.



Şekil 2.6. 1 : 7-segment Aydınlatma Modeli

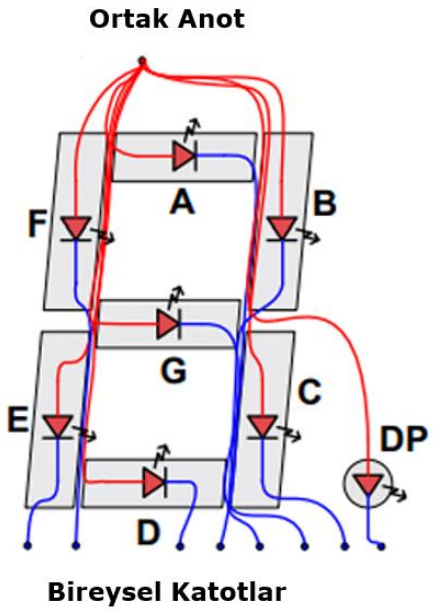
Kullanılan 4 haneli 7-parçalı göstergenin dahili bağlantı şeması Şekil 2.6.2'de gösterilmiştir.



Şekil 2.6. 2 - 4 Haneli 7-Parçalı Göstergenin Bağlantı Şeması

- 4 adet 7-parçalı LED göstergenin anotları ortak bir anot devresine bağlanmıştır, fakat LED katotları ayrıdır.
- 4 hanedeki benzer parçaların katotları 7 adet devre noktasına bağlıdır; bunlar A'dan G'ye kadar etiketlenmiştir.

Şekil 2.6.3'te 7-segment düğüm girişlerinin verilmiştir. 7-segment Display'de istediğiniz sayıları yazdırmak için gerekli olan çıkışlar burada gösterilmektedir.



Şekil 2.6. 3 7-segment Düğüm Girişleri

7-Segment göstergede istenen sayıları yazdırmak için, hangi segmentlerin yanması gerektiği belirlenmeli ve bu segmentlerin katotlarına LOW (0) lojik seviyesi atanmalıdır. Segment sırası olarak GFEDCBA kabul edilmektedir.

Örneğin ekranda '0' sayısının yazılması için G (orta segment) dışında hepsinin LOW (0) olması gerekmektedir.

Aşağıdaki tabloda rakamlar ve Harfler için örnek lojik değerler verilmiştir.

Giriş	Segment Deseni	Rakam /Harf	Lojik
“0000”	"1000000"	0	A, B, C, D, E, F yanık, G sönük.
“0001”	"1111001"	1	B, C yanık, A, D, E, F, G sönük.
“0010”	"0100100"	2	A, B, D, E, G yanık, C, F sönük
“1010”	"0001000"	A	A, B, C, E, F, G yanık, D sönük.
“1111”	"0001110"	F	A, E, F, G, yanık, B, C, D sönük

Tablo 2.6. 1 – Örnek Lojik Değerler

VHDL kodunun oluşturulması;

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity top is
    Port ( giris : in STD_LOGIC_VECTOR (3 downto 0);
           seg : out STD_LOGIC_VECTOR (6 downto 0);
           nokta : out STD_LOGIC;
           anotlar : out STD_LOGIC_VECTOR (3 downto 0));
end top;
architecture Behavioral of top is
begin
nokta<='0';
anotlar<="0000";
with giris select
seg<="1000000" when "0000", --0
"1111001" when "0001", --1
"0100100" when "0010", --2
"0110000" when "0011", --3
"0011001" when "0100", --4
"0010010" when "0101", --5
"0000010" when "0110", --6
"1111000" when "0111", --7
"0000000" when "1000", --8
"0010000" when "1001", --9
"0001000" when "1010", --a
"0000011" when "1011", --b
"1000110" when "1100", --C
"0100001" when "1101", --d
"0000110" when "1110", --E
"0001110" when "1111"; --F
end Behavioral;
```

Kısıt Dosyasının oluşturulması;

```
set_property -dict { PACKAGE_PIN P1   IOSTANDARD LVCMOS33 } [get_ports {seg[0]}]
set_property -dict { PACKAGE_PIN N3   IOSTANDARD LVCMOS33 } [get_ports {seg[1]}]
set_property -dict { PACKAGE_PIN M4   IOSTANDARD LVCMOS33 } [get_ports {seg[2]}]
set_property -dict { PACKAGE_PIN N2   IOSTANDARD LVCMOS33 } [get_ports {seg[3]}]
set_property -dict { PACKAGE_PIN N1   IOSTANDARD LVCMOS33 } [get_ports {seg[4]}]
set_property -dict { PACKAGE_PIN R1   IOSTANDARD LVCMOS33 } [get_ports {seg[5]}]
set_property -dict { PACKAGE_PIN M5   IOSTANDARD LVCMOS33 } [get_ports {seg[6]}]

set_property -dict { PACKAGE_PIN N4   IOSTANDARD LVCMOS33 } [get_ports {anotlar[0]}]
set_property -dict { PACKAGE_PIN P4   IOSTANDARD LVCMOS33 } [get_ports {anotlar[1]}]
set_property -dict { PACKAGE_PIN L4   IOSTANDARD LVCMOS33 } [get_ports {anotlar[2]}]
set_property -dict { PACKAGE_PIN L5   IOSTANDARD LVCMOS33 } [get_ports {anotlar[3]}]

set_property -dict { PACKAGE_PIN P3   IOSTANDARD LVCMOS33 } [get_ports nokta]

set_property -dict { PACKAGE_PIN R2   IOSTANDARD LVCMOS33 } [get_ports {giris[0]}]
set_property -dict { PACKAGE_PIN T3   IOSTANDARD LVCMOS33 } [get_ports {giris[1]}]
set_property -dict { PACKAGE_PIN R5   IOSTANDARD LVCMOS33 } [get_ports {giris[2]}]
set_property -dict { PACKAGE_PIN T5   IOSTANDARD LVCMOS33 } [get_ports {giris[3]}]
```

2.7 Kronometre Uygulaması

Bu uygulama, TRUE- ARTDAC7 FPGA geliştirme kartı üzerinde bulunan yüksek frekanslı sistem saatini (clk) kullanarak doğru zamanlama sinyalleri üretmeyi, bu sinyallerle dakikaları ve saniyeleri saymayı ve elde edilen zaman bilgisini, ortak anot (Common Anode) prensibiyle çalışan 4 haneli 7-parçalı LED gösterge üzerinde dinamik olarak göstermeyi amaçlamaktadır.

Bu sayede clock bölme (clock division), sayaç tasarımı (counter design) ve çoklama (multiplexing) prensiplerini pekiştirilmiş olacaktır.

Teorik Altyapı ve Çalışma Prensibi

1. **Saat Bölme (Clock Division)**

TRUE- ARTDAC7 FPGA kartındaki yüksek frekanslı giriş saatinden (100 MHz) daha düşük frekanslı saat sinyalleri (1 Hz ve 1 kHz) elde edilir. Böylece hem zaman ölçümü hem de 7-segment tarama işlemleri uygun hızda gerçekleştirilir.

2. **Sayıcı Tasarımı**

Kronometre mantığını uygulayabilmek için 60'a kadar sayabilen, BCD formatında çalışan saniye ve dakika sayaçları tasarlanır. Bu sayaçlar her saniyede bir artarak geçen süreyi hesaplar.

3. **Çoklama (Multiplexing) Tekniği**

Dört haneli 7-segment göstergedeki her bir hane, 1 kHz frekanslı bir saat sinyaliyle sırayla aktif edilir. Bu işlem, insan gözünün 60 Hz ve üzerindeki sinyalleri ayırt edememesi nedeniyle tüm hanelerin aynı anda yanıyormuş gibi algılanmasını sağlar.

4. **Ortak Anot Sürüşü**

Kullanılan 7-segment göstergeler ortak anot (common anode) tipindedir. Bu nedenle bir segmentin yanması için ilgili çıkış pinine LOW (0) lojik seviyesi uygulanmalıdır.

5. **Tarama Prensibi ve Görüntü Yenileme**

Tarama prensibine göre, görüntü denetleme devresi dört haneli gösterge birimini kontrol edebilir. Bu devre her hane için anot sinyalini ve o haneye ait katot (segment) desenlerini tekrarlı olarak sırayla gönderir.

Tekrarlama (tazeleme) frekansı, insan gözünün algılama frekansından yüksek olmalıdır.

Her hane, tekrarlama periyodunun dörtte biri kadar bir süre boyunca aktif (yanık) kalır.

Ancak insan gözü bir hanenin sönüp tekrar yandığını fark edemediğinden, her hane sürekli yanıyormuş gibi görünür.

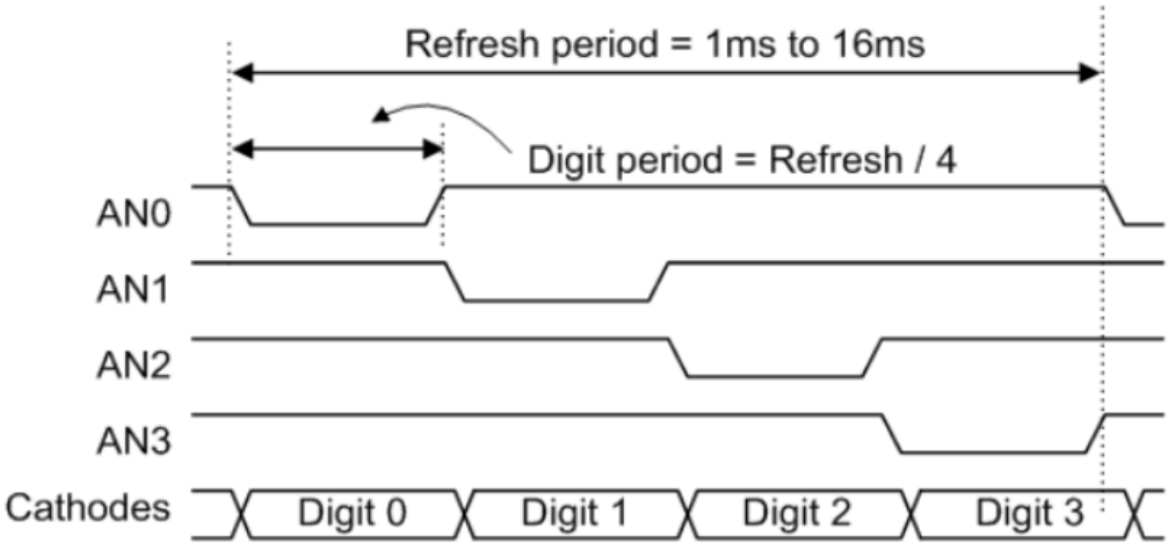
Eğer tekrarlama frekansı yaklaşık 45 Hz civarına düşerse, çoğu insan hanelerdeki görüntüyü titrek olarak algılar. Dört hanenin parlak ve sürekli görünmesi için, bir hanenin bilgisi 1 ms ile 16 ms arasında bir sürede tazelenmelidir. Bu süre aralığı yaklaşık 60 Hz ile 1 kHz arasında değişen bir tazeleme frekansına karşılık gelir. Örneğin, 60 Hz'lik bir tazeleme frekansı uygulandığında tüm gösterge 16 ms'de bir yenilenecektir. Bu durumda, her bir hanenin verisi 4 ms'de bir güncellenmelidir. Bu nedenle, gösterge denetleme devresi doğru katot bilgisini, ilgili anot aktif edildiği anda hazır etmelidir.

Şekil 2.7.1'de tazeleme işleminin zaman diyagramı gösterilmektedir.

Görüldüğü gibi, AN1 aktif edildiğinde B, C, F ve G segmentleri etkinleştirilirse, birinci hanede "4" rakamı görüntülenir.

Benzer şekilde, AN2 aktif edildiğinde A, B, C, D, F ve G segmentleri etkinleştirilirse, ikinci hanede "9" rakamı görünür.

Eğer AN1 ve AN2 sinyalleri 4 ms süresince tekrarlı olarak sürülürse, ekranda "49" değeri sürekli yanıyor gibi görünür.



Şekil 2.7. 1 - Tazeleme İşleminin Zaman Diyagramı

VHDL kodunun oluşturulması;

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity top is
    generic (
        max50MHz : integer := 50_000_000;  -- 50 MHz
        max50KHz : integer := 50_000  -- 50 KHz
    );
    Port (
        clk     : in  STD_LOGIC;
        nokta   : out STD_LOGIC;
        segmentler : out STD_LOGIC_VECTOR(6 downto 0);
        anotlar    : out STD_LOGIC_VECTOR(3 downto 0)
    );
end top;
architecture Behavioral of top is

signal sayici_1hz : integer range 0 to max50MHz := 0;
signal clk_1hz : std_logic := '0';

signal sayici_1khz : integer range 0 to max50KHz := 0;
signal clk_1kHz : std_logic := '0';

signal saniye_birler : integer range 0 to 9 := 0;
signal saniye_onlar  : integer range 0 to 5 := 0;
signal dakika_birler : integer range 0 to 9 := 0;
signal dakika_onlar  : integer range 0 to 5 := 0;
-- Aktif digit
signal aktif_digit : std_logic_vector(1 downto 0) := "00";
signal mevcut_rakam  : integer range 0 to 9;
```

```
begin
-- 1 Hz Clock
process(clk)
begin
if rising_edge(clk) then
        if sayici_1hz = max50MHz - 1 then
            sayici_1hz <= 0;
            clk_1hz <= not clk_1hz;
        else
            sayici_1hz <= sayici_1hz + 1;
        end if;
    end if;
end process;
-- 1 kHz  Clock
process(clk)
begin
if rising_edge(clk) then
        if sayici_1khz = max50KHz - 1 then
            sayici_1khz <= 0;
            clk_1kHz <= not clk_1kHz;
        else
            sayici_1khz <= sayici_1khz + 1;
        end if;
    end if;
end process;
```

```
-- Zaman Sayacı
process(clk_1hz)
begin
   if rising_edge(clk_1hz) then
      if saniye_birler = 9 then
         saniye_birler <= 0;
         if saniye_onlar = 5 then
            saniye_onlar <= 0;
            if dakika_birler = 9 then
               dakika_birler <= 0;
               if dakika_onlar = 5 then
                  dakika_onlar <= 0;
               else
                  dakika_onlar <= dakika_onlar + 1;
               end if;
            else
               dakika_birler <= dakika_birler + 1;
            end if;
         else
            saniye_onlar <= saniye_onlar + 1;
         end if;
      else
         saniye_birler <= saniye_birler + 1;
      end if;
   end if;
end process;
process(clk_1kHz)
begin
   if rising_edge(clk_1kHz) then
      case aktif_digit is
         when "00" =>
            anotlar <= "1110";
            mevcut_rakam <= saniye_birler;
            aktif_digit <= "01";
```

```
            when "01" =>
                anotlar <= "1101";
                mevcut_rakam <= saniye_onlar;
                aktif_digit <= "10";
            when "10" =>
                anotlar <= "1011";
                mevcut_rakam <= dakika_birler;
                aktif_digit <= "11";
            when others =>
                anotlar <= "0111";
                mevcut_rakam <= dakika_onlar;
                aktif_digit <= "00";
        end case;
    end if;
end process;
-- Segment Kodlayıcı
process(mevcut_rakam)
begin
    case mevcut_rakam is
        when 0 => segmentler <= "1000000";
        when 1 => segmentler <= "1111001";
        when 2 => segmentler <= "0100100";
        when 3 => segmentler <= "0110000";
        when 4 => segmentler <= "0011001";
        when 5 => segmentler <= "0010010";
        when 6 => segmentler <= "0000010";
        when 7 => segmentler <= "1111000";
        when 8 => segmentler <= "0000000";
        when 9 => segmentler <= "0010000";
        when others => segmentler <= "1111111";
    end case;
end process;
-- Nokta LED
nokta <= '0';
end Behavioral;
```

Kısıt Dosyasının oluşturulması;

```
set_property -dict { PACKAGE_PIN P1   IOSTANDARD LVCMOS33 } [get_ports {seg[0]}]
set_property -dict { PACKAGE_PIN N3   IOSTANDARD LVCMOS33 } [get_ports {seg[1]}]
set_property -dict { PACKAGE_PIN M4   IOSTANDARD LVCMOS33 } [get_ports {seg[2]}]
set_property -dict { PACKAGE_PIN N2   IOSTANDARD LVCMOS33 } [get_ports {seg[3]}]
set_property -dict { PACKAGE_PIN N1   IOSTANDARD LVCMOS33 } [get_ports {seg[4]}]
set_property -dict { PACKAGE_PIN R1   IOSTANDARD LVCMOS33 } [get_ports {seg[5]}]
set_property -dict { PACKAGE_PIN M5   IOSTANDARD LVCMOS33 } [get_ports {seg[6]}]

set_property -dict { PACKAGE_PIN N4   IOSTANDARD LVCMOS33 } [get_ports {anotlar[0]}]
set_property -dict { PACKAGE_PIN P4   IOSTANDARD LVCMOS33 } [get_ports {anotlar[1]}]
set_property -dict { PACKAGE_PIN L4   IOSTANDARD LVCMOS33 } [get_ports {anotlar[2]}]
set_property -dict { PACKAGE_PIN L5   IOSTANDARD LVCMOS33 } [get_ports {anotlar[3]}]

set_property -dict { PACKAGE_PIN P3   IOSTANDARD LVCMOS33 } [get_ports nokta]

set_property -dict { PACKAGE_PIN N11   IOSTANDARD LVCMOS33 } [get_ports clk]
```

2.8 Servo Motor Açı Kontrolü (PWM Sinyali ile)

Bu uygulama, TRUE- ARTDAC7 FPGA geliştirme kartı üzerinde bulunan yüksek frekanslı sistem saatini (100 MHz) kullanarak hassas zamanlama sinyalleri (1 MHz ve 50 Hz PWM periyodu) üretmeyi, bu sinyaller aracılığıyla Darbe Genişlik Modülasyonu (PWM) sinyali oluşturmayı ve harici bir servo motorun açısını (30^ 60^ 90^) kontrol etmeyi amaçlamaktadır.

Bu sayede saat bölme (clock division), PWM sinyal üretimi ve yumuşak geçiş (smooth transition) mekanizmaları pekiştirilmiş, motor kontrolünün temel prensipleri uygulamalı olarak öğrenilmiş olacaktır.

Teorik Altyapı ve Çalışma Prensibi

1. **Saat Bölme (Clock Division)**

   TRUE- ARTDAC7 FPGA kartındaki 100 MHz'lik giriş saatinden, darbe genişliği kontrolü için gerekli olan 1 MHz frekanslı saat sinyali elde edilir. Bu 1µs (1 mikro saniye 10^{-6}) çözünürlükle darbe genişliğini ayarlamamızı sağlar.

2. **PWM Periyodu ve Darbe Genişliği**

   Servo motorlar, genellikle 50 Hz (20 ms periyotlu) bir PWM sinyali ile kontrol edilir. Bu periyot sabittir; sadece darbe genişliği (pulse width) değiştirilerek açı ayarı yapılır.

   Aşağıdaki Tabloda Servo Motor Açı – PWM İlişkisi verilmiştir;

İstenen Açı (Uygulama)	**Darbe Genişliği (Pulse Width)**	**Görev Döngüsü (Duty Cycle)**	**Periyot (Period)**
0° (Başlangıç Konumu)	≈ 1.0 ms (1000 µs)	%5	20 ms (50 Hz)
30° – 60° (Orta Konumlar)	1.0 – 2.0 ms aralığında değişir	%5 – %10 arası	20 ms (50 Hz)
90° (Maksimum Konum)	≈ 2.0 ms (2000 µs)	%10	20 ms (50 Hz)

Tablo 2.8. 1 Servo Motor Açı – PWM İlişkisi

Görev döngüsü (duty cycle), PWM sinyalinin bir periyot içindeki aktif (HIGH) kalma oranını ifade eder. Şekil 2.8.1 Görev süresinin hesaplanabilmesi için gereken formül verilmiştir.

$$\text{Duty Cycle} = \frac{\text{Aktif Süre (ms)}}{\text{Periyot (ms)}} \times 100$$

Şekil 2.8 1 Görev Döngüsü Formülü

Buna göre:

1.0 ms / 20 ms = %5

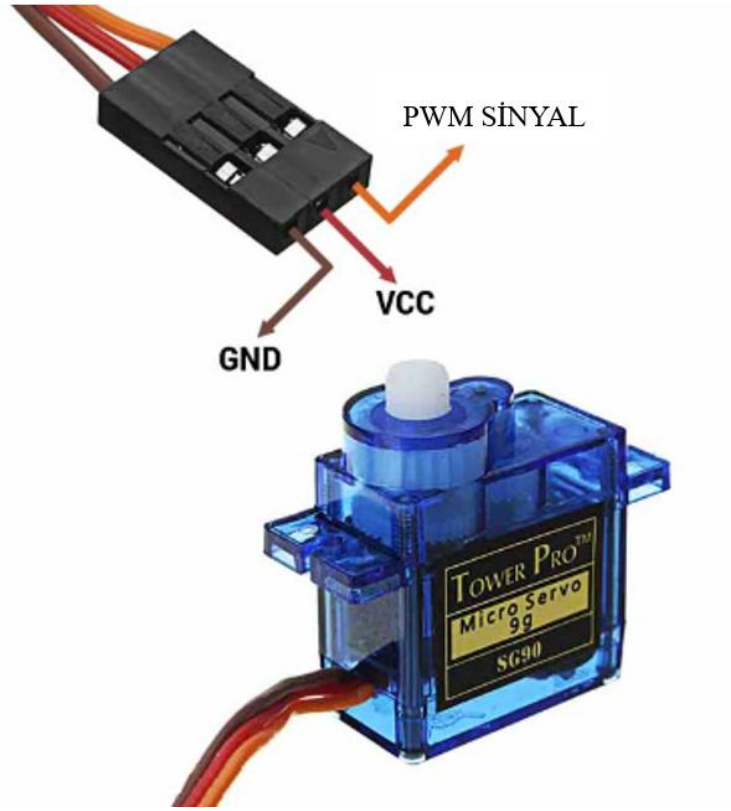
2.0 ms / 20 ms = %10

Görev oranları hesaplanmıştır.

3. Yumuşak Geçiş Mekanizması

Gerçekleştirdiğimiz projede, hedef açı değiştiğinde motorun anlık ve sarsıntılı hareketini önlemek amacıyla 20 ms'de bir derecelik adımlarla yumuşak geçiş mekanizması uygulanmıştır. Bu, motorun hedef açısına kademeli ve kontrollü bir şekilde ulaşmasını sağlar.

Şekil 2.8.2 servo motorun pin yapısı verilmiştir.



Şekil 2.8 2 Servo Motor Pin Yapısı

VHDL kodunun oluşturulması;

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity servo_project is
    generic(
        max : integer := 50  -- 100MHz -> 1MHz clock bölücü
    );
    port(
        clk          : in  std_logic;
        sw_30        : in  std_logic;
        sw_60        : in  std_logic;
        sw_90        : in  std_logic;
        servo_kontrol : out std_logic;
        led_30       : out std_logic;
        led_60       : out std_logic;
        led_90       : out std_logic
    );
end entity;

architecture Behavioral of servo_project is
    signal clk_1MHz    : std_logic := '0';
    signal sayici_1MHz : integer range 0 to max := 0;

    signal target_angle  : integer range 0 to 180 := 0;
    signal current_angle : integer range 0 to 180 := 0;
```

```
begin
    process(clk) -- 100MHz'ten 1MHz clock üretimi
    begin
        if rising_edge(clk) then
            if sayici_1MHz = max - 1 then
                sayici_1MHz <= 0;
                clk_1MHz <= not clk_1MHz;
            else
                sayici_1MHz <= sayici_1MHz + 1;
            end if;
        end if;
    end process;
    process(clk_1MHz)      -- Switch'e göre hedef açı seçimi
    begin
        if rising_edge(clk_1MHz) then
            if sw_90 = '1' then
                target_angle <= 90;
                led_90 <= '1'; led_60 <= '0'; led_30 <= '0';
            elsif sw_60 = '1' then
                target_angle <= 60;
                led_90 <= '0'; led_60 <= '1'; led_30 <= '0';
            elsif sw_30 = '1' then
                target_angle <= 30;
                led_90 <= '0'; led_60 <= '0'; led_30 <= '1';
            else
                target_angle <= 0;
                led_90 <= '0'; led_60 <= '0'; led_30 <= '0';
            end if;
        end if;
    end process;
```

```
    -- PWM üretimi (titreşimsiz)
    process(clk_1MHz)
        variable pwm_time : integer range 0 to 20000 := 0;
        variable pulse_width : integer := 1000;  -- µs cinsinden
    begin
        if rising_edge(clk_1MHz) then
            -- PWM zamanlayıcısı
            if pwm_time < 20000 then
                pwm_time := pwm_time + 1;
            else
                pwm_time := 0;
                -- Hedef açıya yumuşak geçiş (her 20ms'te bir adım)
                if current_angle < target_angle then
                    current_angle <= current_angle + 1;
                elsif current_angle > target_angle then
                    current_angle <= current_angle - 1;
                end if;
                -- Hedef açıya göre darbe süresi (µs)
                pulse_width := 1000 + (current_angle * 1000 / 90);
            end if;
            -- PWM çıkışı
            if current_angle = 0 then
                servo_kontrol <= '0';  -- 0° konumda sinyali kes
            elsif pwm_time < pulse_width then
                servo_kontrol <= '1';
            else
                servo_kontrol <= '0';
            end if;
        end if;
    end process;
end Behavioral;
```

Kısıt Dosyasının oluşturulması;

```
set_property -dict { PACKAGE_PIN T4 IOSTANDARD LVCMOS33 } [get_ports led_30]
set_property -dict { PACKAGE_PIN R3 IOSTANDARD LVCMOS33 } [get_ports led_60]
set_property -dict { PACKAGE_PIN T4 IOSTANDARD LVCMOS33 } [get_ports led_90]

set_property -dict { PACKAGE_PIN R2 IOSTANDARD LVCMOS33 } [get_ports sw_30]
set_property -dict { PACKAGE_PIN T3 IOSTANDARD LVCMOS33 } [get_ports sw_60]
set_property -dict { PACKAGE_PIN R5 IOSTANDARD LVCMOS33 } [get_ports sw_90]

set_property -dict { PACKAGE_PIN D14 IOSTANDARD LVCMOS33 } [get_ports servo_kontrol]

set_property -dict { PACKAGE_PIN N11   IOSTANDARD LVCMOS33 } [get_ports clk]
```

2.9 Ultrasonik Sensör ile Mesafe Ölçümü ve 7-Segment Ekranda Gösterimi

Bu uygulama, TRUE- ARTDAC7 FPGA geliştirme kartı üzerinde HC-SR04 ultrasonik mesafe sensöründen alınan verilerin işlenmesi ve ölçülen uzaklık değerinin 7-segment display üzerinde gerçek zamanlı olarak görüntülenmesini amaçlamaktadır.

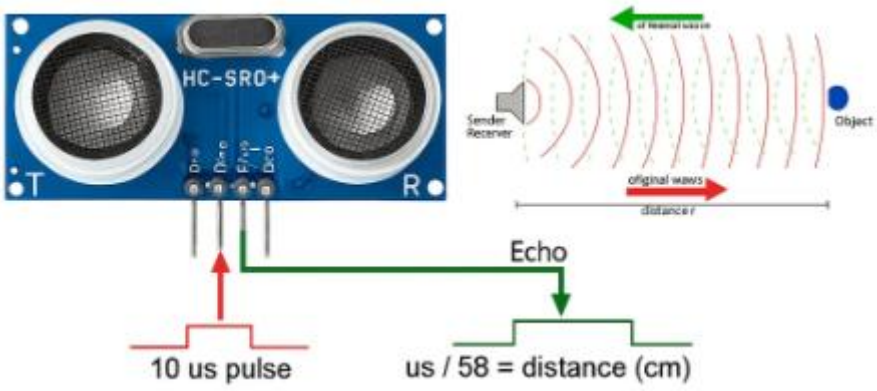
Uygulamada 100 MHz sistem saatinden üretilen 1 MHz'lik sistem saati sayesinde ses dalgalarının uçuş süresi (time-of-flight) hesaplanmış, ölçülen süre üzerinden uzaklık değeri (cm) FPGA içerisinde hesaplanarak hem LED gösterge ile hem de iki basamaklı 7-segment ekran ile kullanıcıya sunulmuştur.

Bu sayede ultrasonik mesafe ölçüm prensibini, FPGA tabanlı zaman ölçümü (pulse width measurement), clock division, sayısal veri işleme, gerçek zamanlı 7-segment sürme (multiplexing) kavramlarını uygulamalı olarak öğrenilmiş olacaktır.

Teorik Altyapı ve Çalışma Prensibi

1. Ultrasonik Mesafe Ölçüm Yöntemi (Time of Flight)

HC-SR04 sensörü, çevreye kısa süreli bir 40 kHz ultrasonik ses darbesi (TRIG) gönderir. Ses engelden yansıyıp sensöre geri döndüğünde, modül ECHO pinini HIGH düzeyinde tutarak uçuş süresini FPGA çipinin iletir.



Şekil 2.10 1 HC-SR04

Sensörün temel çalışma prensibi bu şekildedir;

$$\textbf{Mesafe (cm)} = \frac{\text{ECHO Süresi (µs)}}{58}$$

Sesin havadaki hızı ≈ 340 m/s ≈ 0.034 cm/µs

Gidiş + geliş süresi ölçüldüğü için ikiye bölünür → 1 cm ≈ 58 µs.

Bu uygulamada FPGA, ECHO pininin HIGH olduğu süreyi mikro-saniye çözünürlüğüyle ölçer.

2. Clock Division: 100 MHz → 1 MHz Zaman Tabanı

TRUE- ARTDAC7 FPGA kartı 100 MHz sistem saatine sahiptir. Ultrasonik sensör zamanlamasının sağlıklı yapılabilmesi için mikro-saniye düzeyinde çözünürlüğe gereksinim duyulur.

$$100\,\text{MHz} \rightarrow 1\,\text{MHz} \quad (1\,\mu s \text{ çözünürlük})$$

Üretilen 1 MHz sistem saat sayesinde TRIG palsi hassas şekilde üretilebilir, ECHO sinyali mikro-saniyelik doğrulukla ölçülebilir.

3. TRIG Darbesinin Üretilmesi

Sensörün tetiklenmesi için; 10 µs boyunca TRIG = 1 yapılır. Daha sonra TRIG = 0 konumuna çekilip 60 ms civarında bekleme süresi ile sensörün yeni ölçüme hazır olması sağlanır. Bu işlem FPGA içinde otomatik olarak tekrarlanır.

4. ECHO Süresinin Ölçülmesi

TRIG darbesi sonrası sayaç 1 MHz clock ile çalıştırılır. ECHO = '1' olduğu sürece sayaç artmaya devam eder. ECHO = '0' konumuna düştüğünde sayaç durdurulur. Sayaç değeri uçuş süresidir ve formül kullanılarak uzaklığa çevrilir.

Sayaç değeri bu şekilde hesaplanır;

$$580\,\mu s \Rightarrow 580/58 = 10\,cm$$

5. LED Bar ile Yakınlık Seviyesinin Gösterimi

Ölçülen mesafe aralıklarına göre TRUE- ARTDAC7 FPGA kartındaki 16 adet LED kademeli olarak yakılır.

0–4 cm → 1 LED

10–14 cm → 3 LED

75–79 cm → tüm LED'ler yanar

Bu, kullanıcının mesafe değişimini görsel olarak takip etmesini sağlar.

6. 7-Segment Display ile Sayısal Gösterim

Hesaplanan mesafe değeri iki basamağa ayrılarak (onlar – birler) 7-segment ekrana aktarılır.

Multiplexing; Anotlar sırayla aktif edilir ve segment bilgisi güncellenir. Bu sayede FPGA yalnızca 7 segment + anot kontrolü ile iki basamaklı gösterim yapar.

1 kHz yenileme frekansı; İnsan gözü iki basamağın yanıp söndüğünü algılamaz, sürekli yanıyormuş gibi görülür.

Örnek ekran çıktısı;

Ölçülen mesafe = 27 cm → ekranda "27" görünür.

Sensör limit aşarsa → "99" gösterilir.

Proje, modüler bir yaklaşımla tasarlanmıştır ve üç ana modülden oluşmaktadır: Top modülü (ana kontrol ve saat üretimi), Sensor modülü (ultrasonik veri okuma) ve 7-Segment modülü (görüntüleme). Bu yapı, kodun anlaşılabilirliğini artırmakta ve aynı zamanda her modülün ayrı test edilmesini sağlamaktadır.

Yapılan projeye ait kodları Eğitim CD'sinde bulabilirsiniz.

2.10 QSPI Flash'a Bitstream Yükleme

Amaç: Bu uygulamada, TRUE- ARTDAC7 FPGA geliştirme kartı üzerindeki MT25QL128ABA model 128 Mbit (16 MB) QSPI NOR Flash belleğe FPGA bitstream'inin kalıcı olarak yazılmasını ve kartın enerji verildiğinde otomatik olarak bu bitstream'i okuyup çalıştırmasını göstermek amaçlanmaktadır. Bu sayede kartın konfigürasyon mekanizması (JTAG vs. Flash boot) pekiştirilecektir.

Teorik Altyapı ve Çalışma Prensibi

1. **FPGA çiplerinde Bitstream ve Konfigürasyon Mantığı**

FPGA çipleri, güç verildiğinde çalışabilmek için cihazın iç yapılandırma belleğine yüklenen bir bitstream dosyasına ihtiyaç duyar. Bu bitstream, tasarımın tüm lojik yapısını, bağlantılarını ve yapılandırma ayarlarını içerir. Ancak FPGA çipinin içindeki yapılandırma belleği uçucu (volatile) olduğundan, güç kesildiği anda içerisindeki veri tamamen silinir. Bu nedenle FPGA çipleri her açılışta bitstream'in yeniden yüklenmesine ihtiyaç duyar. FPGA çipleri JTAG arayüzü üzerinden geçici olarak programlanabilir fakat güç kesildiğinde program silinir. FPGA çipleri ayrıca QSPI NOR Flash belleği üzerine yazılan bitstream dosyası üzerinden de programlanabilirler bu sayede bitstream Flash içinde kalıcı olarak saklanır ve karta enerji geldiği anda FPGA flash üzerindeki veriyi otomatik olarak okuyarak kendini konfigüre edebilir.

Bu mekanizma, AMD/Xilinx tarafından yayınlanan UG470 – 7 Series FPGAs Configuration User Guide dokümanında 7-seri FPGA çiplerinin temel çalışma prensibi olarak tanımlanmıştır. Dokümana göre FPGA, güç verildiği anda ilgili konfigürasyon pinlerinin (M0–M2) ayarına göre uygun başlatma modunu seçer ve eğer Master SPI/QSPI modu seçiliyse bitstream'i doğrudan harici Flash belleğinden okur.

Ayrıca her FPGA ailesinin bir bitstream uzunluğu bulunmaktadır. TRUE- ARTDAC7 geliştirme kartında kullanılan XC7A50T FPGA çipinin bitstream uzunluğu şekil 2.10.1 verilmiştir.

Table 1-1: Bitstream Length

Device	Configuration Bitstream Length (bits)	Minimum Configuration Flash Memory Size (Mb)	JTAG/Device IDCODE[31:0] (hex)(1)	Production IDCODE Revision	JTAG Instruction Length (bits)	Super Logic Regions
AMD Spartan 7 Family						
7S6	4,310,752	8	X3622093	0 or later	6	N/A
7S15	4,310,752	8	X3620093	0 or later	6	N/A
7S25	9,934,432	16	X37C4093	0 or later	6	N/A
7S50	17,536,096	32	X362F093	0 or later	6	N/A
7S75	29,494,496	32	X37C8093	0 or later	6	N/A
7S100	29,494,496	32	X37C7093	0 or later	6	N/A
AMD Artix 7 Family						
7A12T	9,934,432	16	X37C3093	0 or later	6	N/A
7A15T	17,536,096	32	X362E093	0 or later	6	N/A
7A25T	9,934,432	16	X37C2093	0 or later	6	N/A
7A35T	17,536,096	32	X362D093	0 or later	6	N/A
7A50T	17,536,096	32	X362C093	0 or later	6	N/A
7A75T	30,606,304	32	X3632093	1 or later	6	N/A
7A100T	30,606,304	32	X3631093	1 or later	6	N/A
7A200T	77,845,216	128	X3636093	1 or later	6	N/A

Şekil 2.10. 1 FPGA Bitstream yapısı

Bitstream Length: 17,536,096 bits

Minimum Flash Boyutu: 32 Mbit

Bu nedenle kart üzerinde kullanılan 128 Mbit (16 MB) Micron MT25QL128ABA Flash bellek, bitstream saklamak için yeterlidir ve gerektiğinde çoklu multi-boot için kullanılabilir.

2. Bitstream Boyutu ve Flash Bellek Gereksinimi

FPGA çiplerinin programlanabilmesi için gereken bitstream dosyasının boyutu, kullanılacak harici Flash belleğin kapasitesinin belirlenmesinde kritik bir rol oynar. Artix-7 serisi cihazlar için her FPGA ailesinin bitstream uzunluğu sabittir ve UG470 “7 Series FPGA Configuration” dokümanında resmi olarak listelenmiştir.

TRUE- ARTDAC7 FPGA kartında Micron MT25QL128ABA kullanılmıştır. 128 Mbit (16 MB) kapasite QSPI x1/x2/x4 destekli olan ve FPGA çipinin hızlı ve güvenilir şekilde boot edebilmesi için üretici firma tarafından önerilen tipte NOR tabanlı bir Flash bellektir. Bu sayede kart

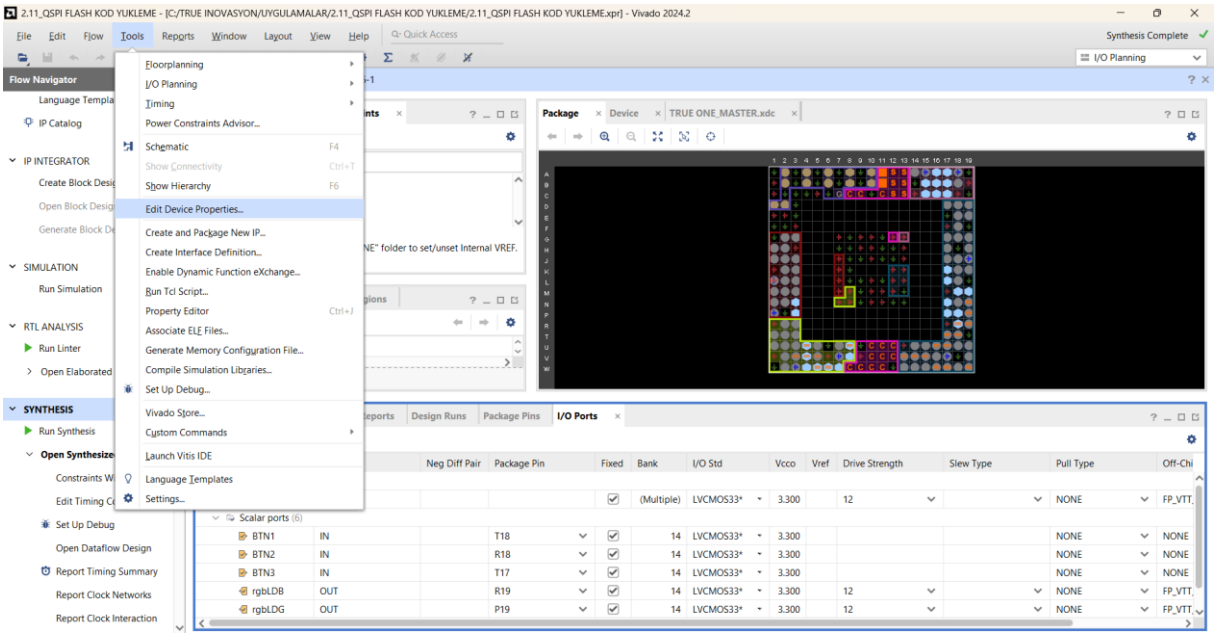
üzerinde Bitstream kalıcı olarak saklanabilir ve güç kesilip verilse bile FPGA her açılışta otomatik olarak Flash'tan kendini konfigüre edebilir.

3. Flash Belleğe Bitstream Yükleme İşlemi

Aşağıdaki adımlar, Vivado üzerinden FPGA bitstream'inin QSPI Flash belleğe kalıcı olarak programlanması için izlenecek süreci göstermektedir.

Adım 1 : Sentez (Synthesis) ve Implementasyon adımları Tamamlandıktan ve Konfigürasyon ayarları tamamlandıktan sonra tasarımdaki tüm giriş–çıkış portları FPGA pinlerine atanmalı ve XDC dosyası kaydedilmelidir. Bu işlem tamamlandıktan sonra Edit Device Properties Menüsüne Girilir.

Tasarım sentezlendikten ve implementasyon tamamlandıktan sonra Flash'a yazılacak bitstream dosyasının bazı konfigürasyon ayarlarını doğrulamak gerekir. Bunun için Vivado menüsünden Tools → Edit Device Properties yolunu izleyiniz.



Şekil 2.10. 2 QSPI Flash'a Bitstream Yükleme Adım-1

Adım 2 : Edit Device Properties Penceresinde Konfigürasyon Ayarlarının Yapılması

Açılan Edit Device Properties penceresinin sol tarafındaki menüden Configuration sekmesine tıklanır. Bu bölüm, FPGA çipinin harici Flash bellekten nasıl konfigüre edileceğini belirleyen temel parametreleri içerir

Configuration Rate (MHz) Bu parametre, FPGA çipinin Flash üzerinden bitstream okurken kullanacağı konfigürasyon saat frekansını belirler varsayılan 3–66 MHz (cihaza göre Vivado

tarafından atanır) Daha yüksek frekans daha hızlı boot etme süresi demektir ancak çok yüksek frekans bazı Flash modelleriyle uyumsuzluğa neden olabilir.

128 Mbit Micron MT25QL128 Flash bellek, Quad-SPI modunda yüksek hız desteklediği için 40 MHz güvenli ve yaygın tercih edilen bir ayardır.

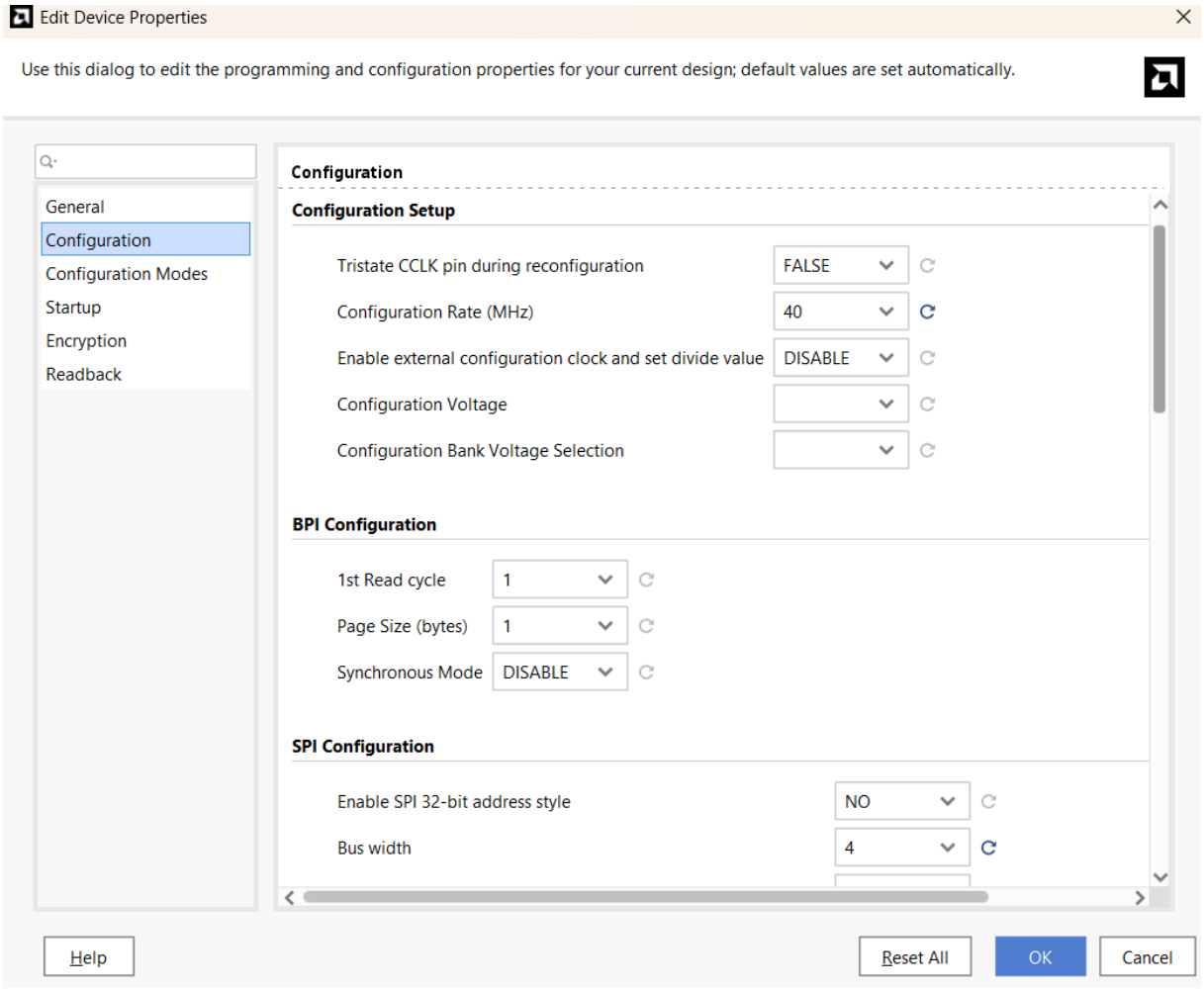
SPI Bus Width Bu parametre, Flash'tan okunacak verinin kaç bit genişliğinde taşınacağını belirler

Bus width = 1 → SPI x1 (yavaş)

Bus width = 2 → Dual-SPI

Bus width = 4 → Quad-SPI (QSPI)

QSPI modunda FPGA, her saat darbesinde 4 bit veri okuyabilir, bu da Flash'tan boot süresini yaklaşık 4 kat hızlandırır. Bus Width parametresi Flash ile FPGA arasındaki veri yolunun genişliğini belirler. TRUE- ARTDAC7 kartındaki MT25QL128 Flash belleği Quad-SPI (x4) modunu desteklediği için Bus Width = 4 seçilir.



Şekil 2.10. 3 QSPI Flash'a Bitstream Yükleme Adım-2

Edit Device Properties penceresinde Configuration Rate ve SPI Bus Width ayarları yapılıp OK butonuna basıldığında, Vivado bu konfigürasyon parametrelerini otomatik olarak projenin .xdc (constraints) dosyasına ekler. Böylece bitstream oluşturma aşamasında FPGA bu ayarları kullanarak QSPI Flash üzerinden seçilen hız ve veri yolu genişliği ile çalışacak şekilde yapılandırılır.

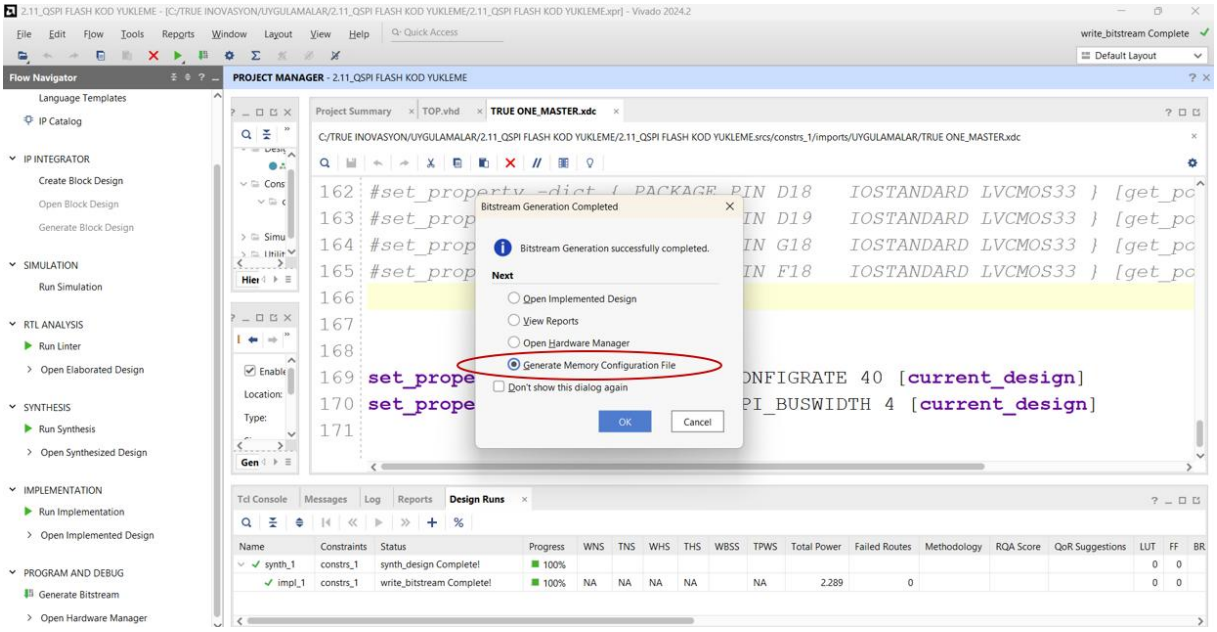
Not: Bu adım tamamlandıktan sonra bitstream generation (Generate Bitstream) işlemine geçmeden önce XDC dosyasının kaydedildiğinden emin olunmalıdır.

TRUE_ ARTDAC7_MASTER.xdc dosyasına Vivado tarafından eklenen satırlar şekil 2.10.4 verilmiştir.

```
set_property BITSTREAM.CONFIG.CONFIGRATE 40 [current_design]
set_property BITSTREAM.CONFIG.SPI_BUSWIDTH 4 [current_design]
```

Şekil 2.10. 4 Bitsrteam XDC dosyası

Bitstream oluşturma tamamlandıktan sonra açılan pencerede (Şekil 2.10.5) Generate Memory Configuration File seçeneği işaretlenir ve ilerlemek için OK butonuna basılır. Bu işlem, QSPI Flash'a yazılacak olan .Bin dosyasının oluşturulması için gerekli süreci başlatır.



Şekil 2.10. 5) Generate Memory Configuration File

Vivado, Flash belleğe yazılabilir konfigürasyon dosyası üretirken farklı formatlar sunar. En yaygın kullanılanlar;

BIN: Ham ikili (binary) format. Özellikle QSPI Flash programlamada oldukça pratiktir.

MCS: Intel HEX tabanlı format. Daha eski PROM programlayıcılarla uyumludur.

HEX: Bazı özel programlayıcılarla veya üretim hatlarında tercih edilir.

BIN dosyası seçildikten sonra Memory Part bölümündeki üç nokta (…) butonuna basılarak uygun Flash modeli seçilir. Açılan pencerede TRUE- ARTDAC7 FPGA geliştirme kartında kullanılan MT25QL128ABA QSPI NOR Flash'a ait özellikler seçilir;

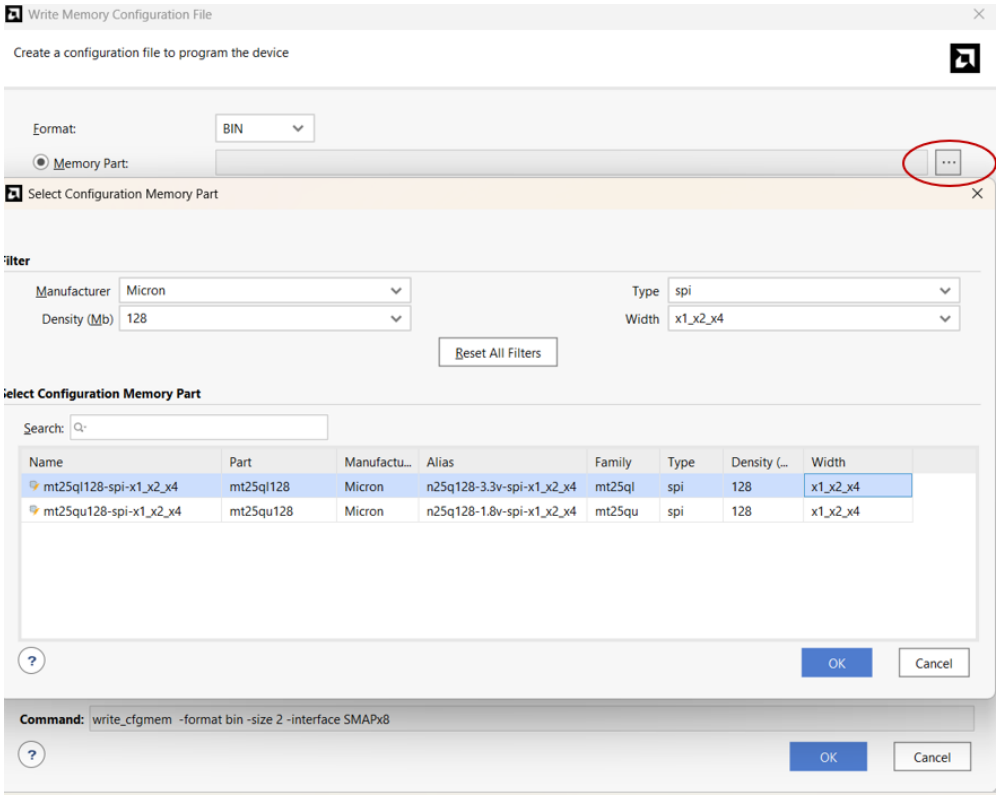
Manufacturer: Micron

Density: 128 Mbit

Type: SPI

Width: x1_x2_x4

Filtreleri seçildiğinde kart üzerinde bulunan QSPI Flash modeli listede görünür. Bu model seçildikten sonra OK butonuna basılır ve File Name seçeneği seçilip BIN dosyasına bir ismin verilir.



Şekil 2.10. 6 QSPI Flash Seçimi

1. Interface Ayarı

Bu bölümde Flash ile haberleşme arayüzü seçilir. TRUE- ARTDAC7 FPGA kartında QSPI tabanlı MT25QL128ABA kullanıldığı için arayüz kısmında SPIx4 seçilmelidir. Bu ayar, FPGA çipinin Flash belleği 4-bit genişlikteki QSPI hattı üzerinden okuyacağını belirtir.

2. "Load Bitstream" Seçeneği

Bu kutu işaretlendiğinde Vivado'ya, oluşturulacak Flash konfigürasyon dosyasına bir bitstream eklenmesi gerektiği bildirilmiş olur. Böylece Flash içerisine sadece "boş bir yapı" değil, FPGA çipinin çalıştıracağı gerçek tasarım dahil edilir.

3. Bitstream Dosyasını Seçmek İçin Göz atma (Browse) Butonu

Bu buton ile bitstream dosyasının (.bit) bulunduğu klasör seçilir. Vivado, her derleme sonrasında bitstream'i /runs/impl_1/ dizinine kaydettiği için seçim bu klasörde yapılacaktır.

4. Dizinin Açılması

Göz atma penceresi açıldığında projeye ait çalışma klasörleri görüntülenir.

Buradan impl_1 klasörüne gidilmelidir. Bu klasör, FPGA tasarımının implementasyon aşamasından sonra oluşturduğu tüm çıktı dosyalarını içerir.

5. "impl_1" Klasörünün İçeriği

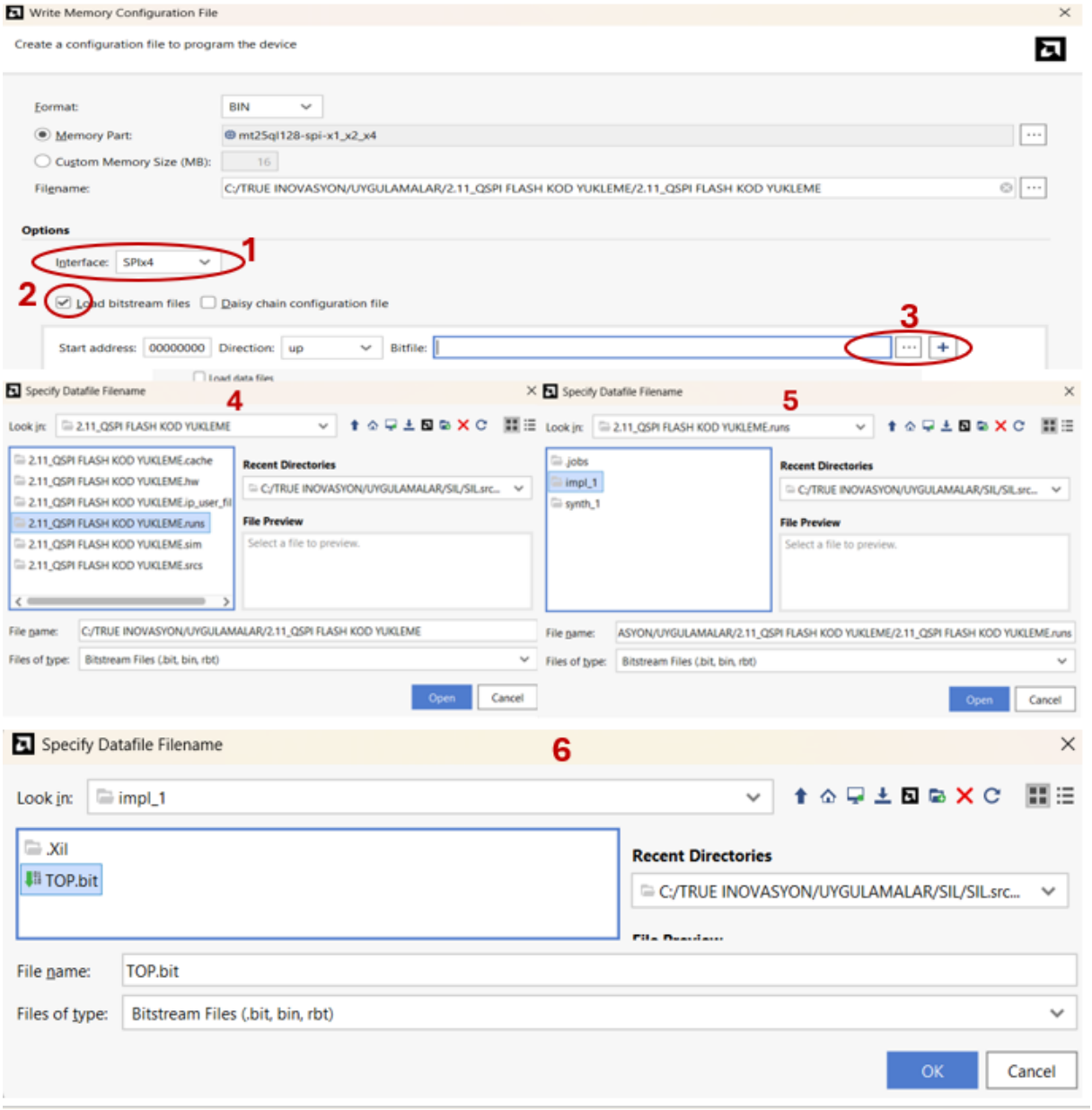
Bu klasör açıldığında Vivado'nun ürettiği çeşitli dosyalar listelenir. Bunların içinde bitstream dosyası da bulunur. Kullanıcı, dosya listesinden TOP.bit ya da projede belirtilen bitstream dosyasını seçmelidir.

6. Bitstream Dosyasının Seçilmesi (TOP.bit)

Son adımda, kullanılacak bitstream dosyası seçilir ve OK butonuna basılarak Flash konfigürasyon dosyası oluşturma adımına geçilir.

Bu işlem tamamlandığında Vivado, seçilen bitstream'i QSPI Flash belleğe yazılmaya hazır bir .bin formatındaki yapı içine yerleştirir.

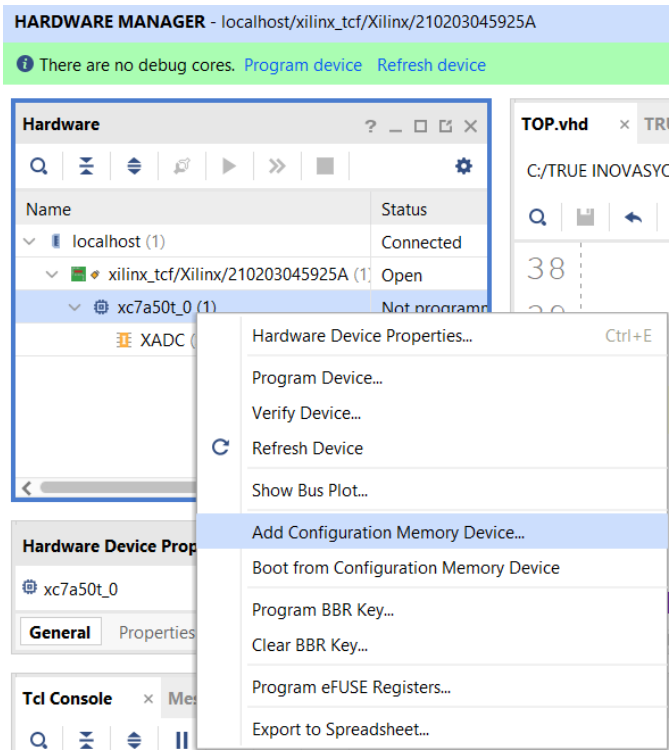
Bu işlemlerin sırası şekil 2.10.7'de verilmiştir.



Şekil 2.10. 7 BIN dosyası oluşturma

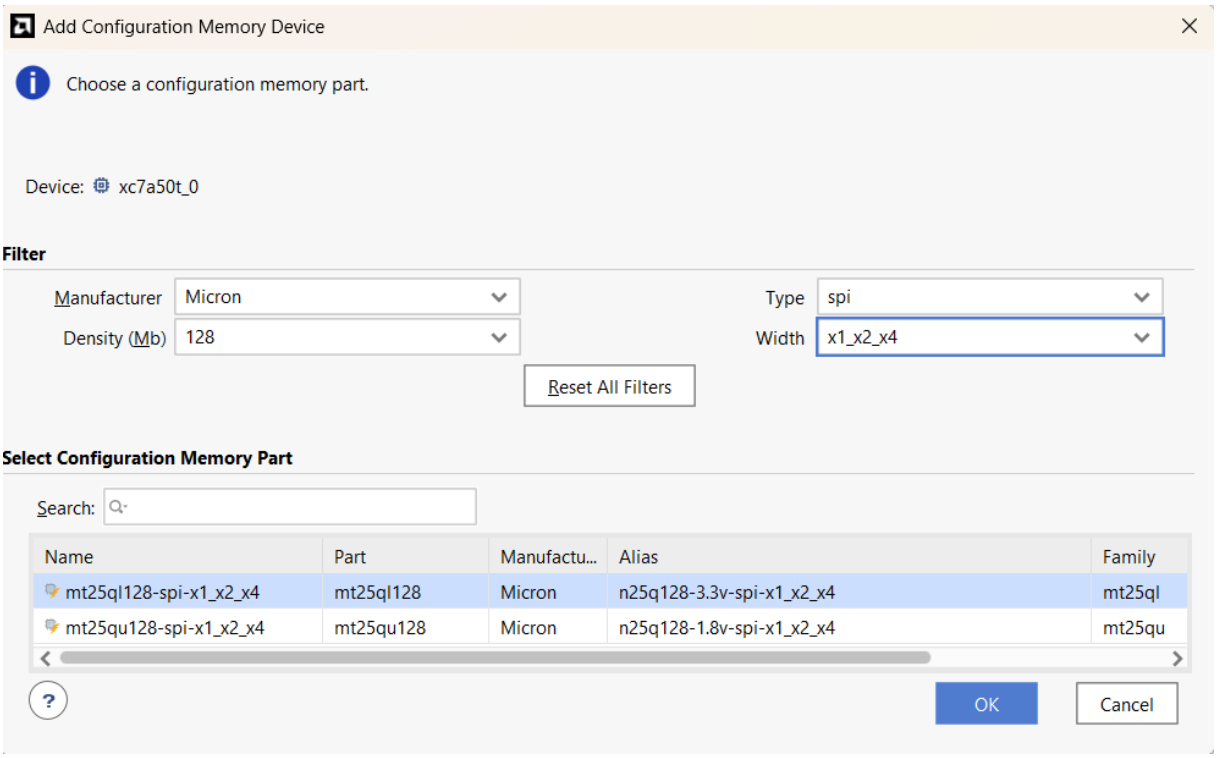
Bitstream dosyası oluşturulduktan ve Memory Configuration File hazırlanmak üzere seçildikten sonra, sıradaki işlem Flash belleğin Vivado tarafından tanınmasıdır. Bunun için Vivado'nun Hardware Manager arayüzü kullanılır.

Aşağıda Şekil 2.10.8'de gösterildiği gibi FPGA cihazının üzerine sağ tıklanır ve açılan menüden Add Configuration Memory Device seçeneği işaretlenir. Bu işlem, Vivado'nun FPGA ile birlikte kullanılacak olan harici QSPI Flash belleği tanımasını sağlar ve programlama akışının (erase → program → verify) başlatılabilmesi için gereklidir.



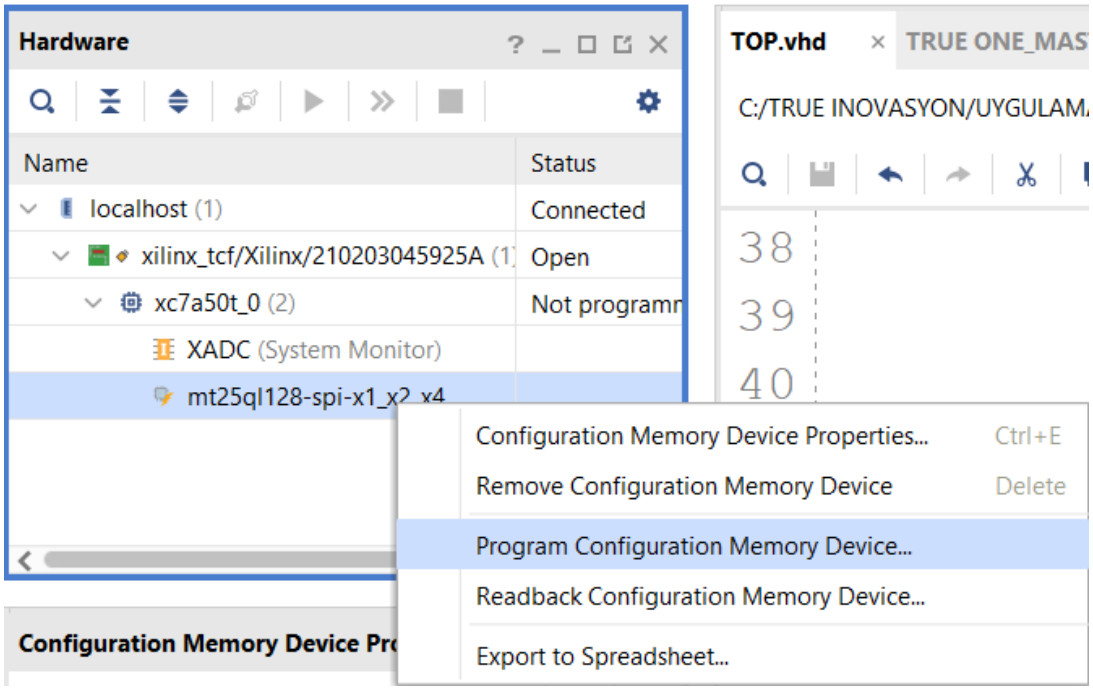
Şekil 2.10. 8 QSPI FLASH Ekleme

Bitstream dosyasını Flash belleğe yazabilmek için ilk olarak Vivado'nun FPGA çipine bağlı hangi Flash modelini kullanacağını bilmesi gerekir. Bunun için Add Configuration Memory Device ekranında gezinilir ve TRUE- ARTDAC7 FPGA geliştirme kartında kullanılan Micron MT25QL128ABA modeli seçilir.



Şekil 2.10. 9 QSPI FLASH Seçimi

Şekil 2.10.10'da görüldüğü gibi, Flash modeli seçildikten ve ilgili ayarlar onaylandıktan sonra artık bitstream dosyasının QSPI Flash belleğe programlanması gerekir. Vivado'nun Hardware Manager penceresi üzerinden bu işleme başlanır.



Şekil 2.10. 10 QSPI FLASH Programlama

Configuration File Flash'a yazılacak olan bitstream dosyasının (.bin) seçildiği alandır. Buradan butonuna basılarak oluşturulan **.bin** dosyası seçilir.

Erase: Flash belleğin ilgili bölümünü siler (her zaman seçili olmalıdır).

Program: Bitstream dosyasını Flash'a yazar.

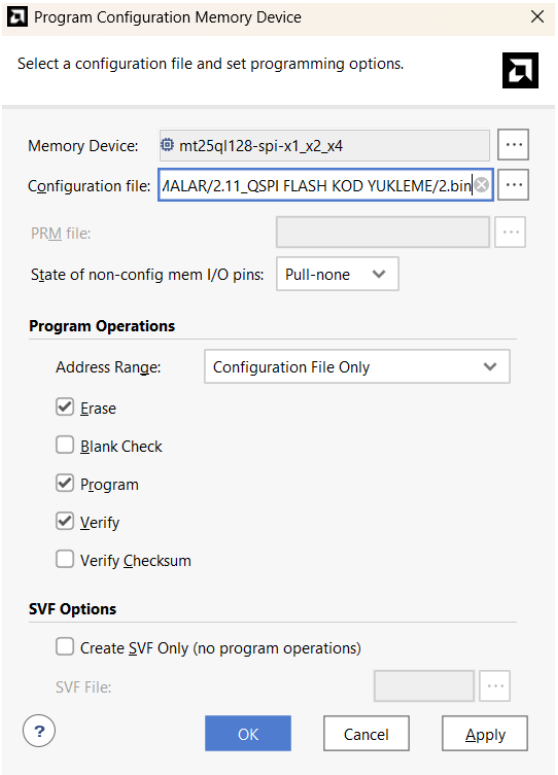
Verify: Yazılan veriyi tekrar okuyarak doğrular.

Verify Checksum: (İsteğe bağlı) Daha derin doğrulama yapar,

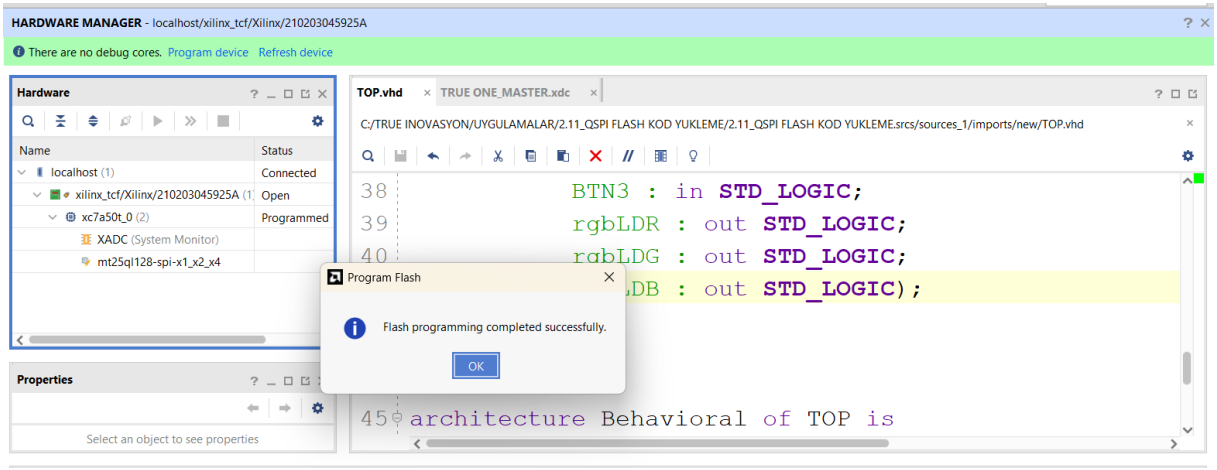
Tüm ayarlar kontrol edildikten sonra TRUE- ARTDAC7 kartı üzerindeki mod seçiminden mod JTAG alınması gerekir son olarak OK tuşuna basılarak programlama işlemi başlatılır.



Şekil 2.10. 11 Mod Seçimi



Şekil 2.10. 12 Programlama İşlemi



Şekil 2.10. 12 Tamamlanan Programlama

Flash başarıyla programlandıktan sonra FPGA çipinin harici Flash belleği kullanabilmesi için kart enerjisi kesilir, MODE jumper'ı JTAG'dan QSPI moduna alınır ve Vivado Hardware Manager kapatılır. Bu işlemlerden sonra kart yeniden açıldığında FPGA, QSPI Flash içindeki bitstream dosyasını otomatik olarak okuyarak çalışmaya başlar.

2.11 LM35 Sensörü ile Sıcaklık Ölçümü

Bu uygulamada, TRUE- ARTDAC7 FPGA geliştirme kartı üzerinde bulunan ADC modülü kullanılarak LM35 sıcaklık sensöründen analog sıcaklık bilgisi okunacak ve bu değer dijital işleme tabi tutularak FPGA üzerinde işlenecektir. Sensörden elde edilen analog çıkış voltajı, FPGA üzerindeki ADC dönüştürücü aracılığıyla sayısal değere çevrilerek sıcaklık (°C) bilgisi kullanıcıya sunulacaktır.

Bu uygulama sayesinde analog–dijital dönüşüm mantığını, sensör veri okuma yöntemlerini ve sıcaklık hesaplama prensibini pekiştirilmiş olacaktır.

Teorik Altyapı ve Çalışma Prensibi

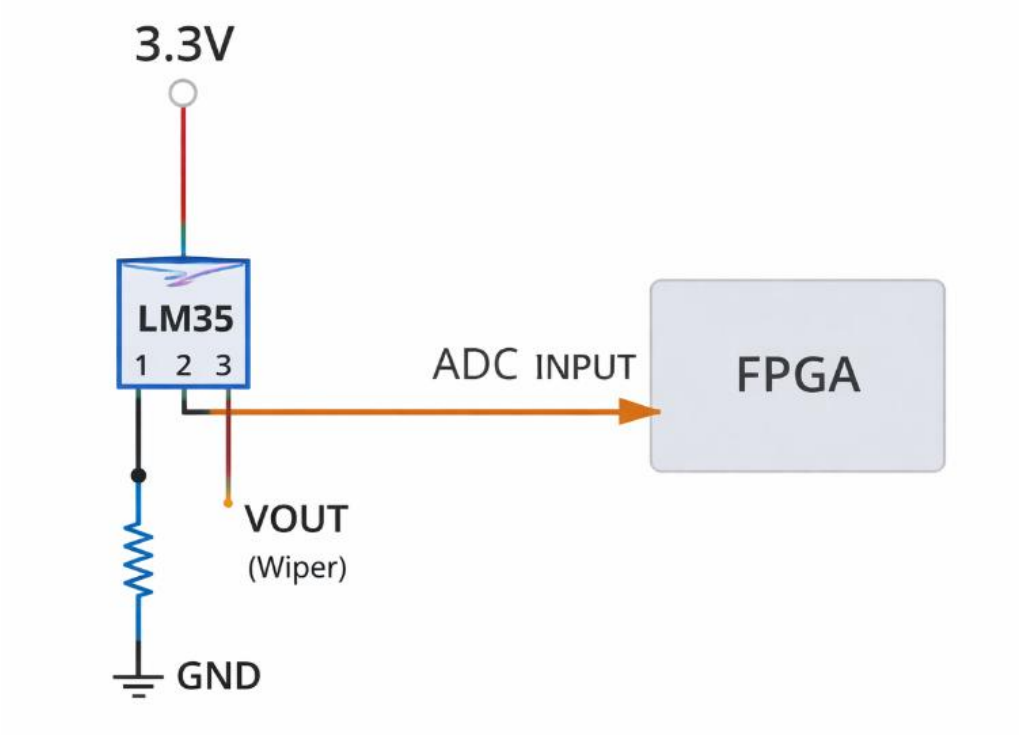
LM35 Sensörü (Temel Tanım)

LM35, sıcaklığa karşı doğrusal olarak değişen bir analog voltaj üreten hassas bir sıcaklık sensörüdür. Sensörün çıkış voltajı sıcaklık ile 10 mV/°C oranında artar.

25 °C → 250 mV

40 °C → 400 mV

100 °C → 1.000 mV



Şekil 2.11. 1 *LM35 Pin Yapısı*

Projede LM35 sensöründen alınan analog değer, FPGA üzerindeki XADC modülü tarafından 12 bit çözünürlükte sayısal değere dönüştürülmektedir. XADC'nin AUX giriş aralığı 0–1.0V olarak yapılandırılmıştır.

0V → 0

1.0V → 4095

LM35 sensörü 10 mV / °C (0.01V) doğrusal çıkış vermektedir.

1.0V → 100°C

0.5V → 50°C

0.25V → 25°C

Bu durumda sıcaklık hesabı için kullanılan formül

$$\text{Sıcaklık (°C)} \approx \frac{ADC_Değeri \times 100}{4095}$$

Bu formül doğrudan uygulama kodunda da kullanılmıştır.

XADC IP Catalog Yapılandırması

Bu uygulamada LM35 sıcaklık sensöründen alınan analog voltajın FPGA tarafından dijital olarak okunabilmesi için XADC (Xilinx Analog-to-Digital Converter) modülü kullanılmıştır. XADC, Vivado IP Catalog üzerinden projeye eklenen bir IP core'dur ve 12-bit çözünürlükte analog-dijital dönüşüm yapar.

IP Catalog Üzerinden XADC Ekleme bu adımlar uygulanmıştır;

Tools → IP Catalog menüsü açılır.

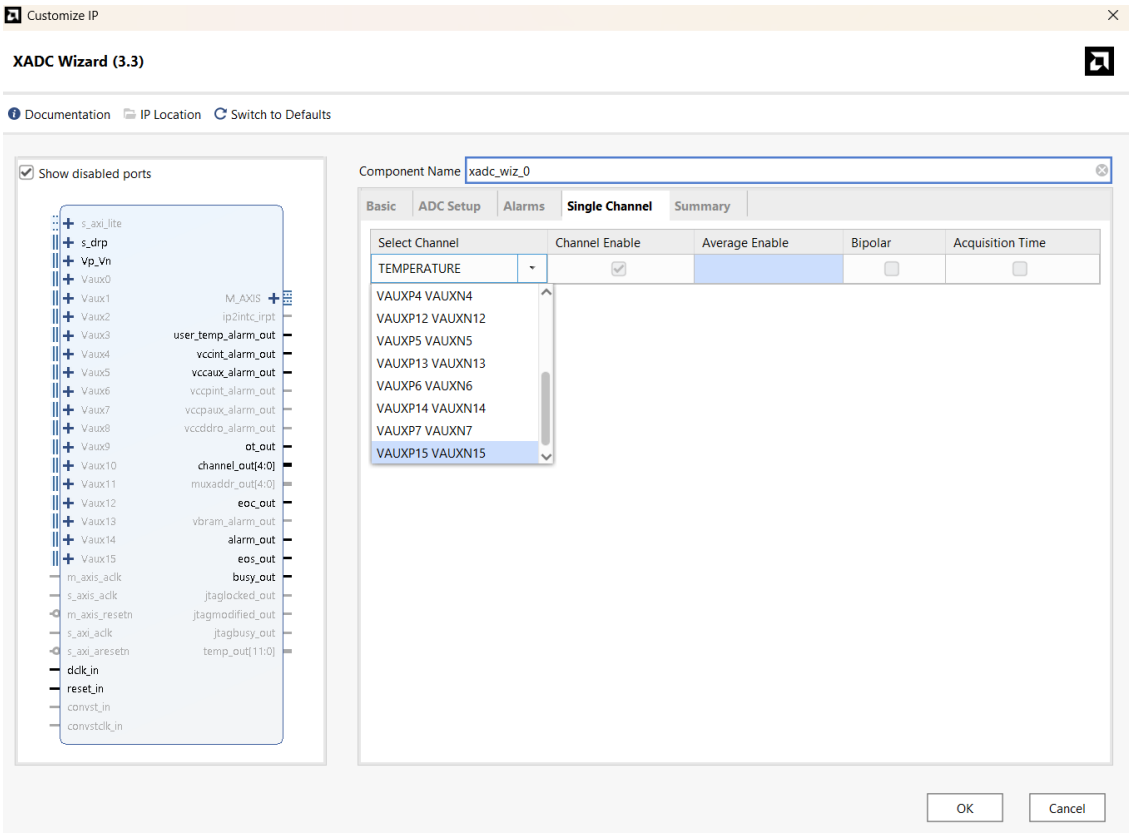
Arama kısmına "XADC" yazılır.

XADC Wizard seçilerek projeye eklenir.

Karşınıza gelen konfigürasyon ekranında

Single Channel Mode seçilir.

AUX girişlerinden kullanılacak kanal (örneğin AUX15P → N2 pinine bağlı LM35 çıkışı AUX15N → GND pinine bağlı) aktif hale getirilir.



Şekil 2.11. 2 *XADC Wizard*

Yapılan projeye ait kodları Eğitim CD'sinde bulabilirsiniz.

2.12 LDR Sensörü ile Parlaklık Ölçümü

Bu uygulamada, TRUE- ARTDAC7 FPGA geliştirme kartı üzerinde yer alan ADC modülü kullanılarak LDR (ışığa bağlı direnç) sensöründen gelen analog değer okunacak ve ortamın ışık seviyesine göre parlaklık yüzdesi hesaplanacaktır. LDR sensörü, üzerine düşen ışık miktarına göre direnci değişen bir yapıya sahiptir; bu direnç değişimi ADC tarafından yorumlanarak dijital bir ışık/parlaklık değeri oluşturulur.

Uygulama sayesinde ADC okuma mantığı, değişken direnç sensörleriyle çalışma prensibi ve parlaklık hesaplama prensibi pekiştirilmiş olacaktır.

Teorik Altyapı ve Çalışma Prensibi

LDR Sensörü (Temel Tanım)

LDR (Light Dependent Resistor), üzerine düşen ışık miktarına göre direnci değişen bir ışık sensörüdür. Sensörün direnci ışık arttıkça azalır, ışık azaldıkça yükselir.

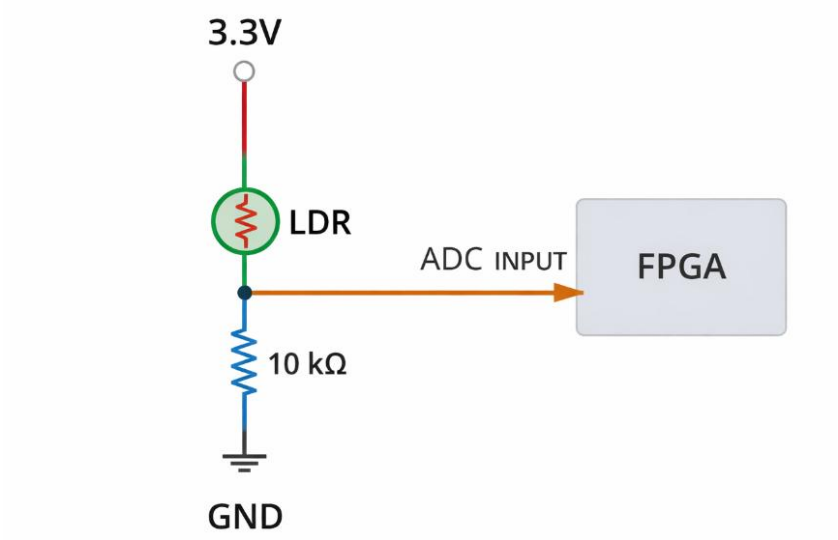
Aydınlık ve karanlık arasındaki tipik direnç değerleri şu şekildedir

Karanlık ortamda → 200 kΩ – 1 MΩ

Normal oda ışığında → 10 kΩ – 50 kΩ

Aydınlık ortamda → 1 kΩ – 5 kΩ

Bu direnç değişimi bir gerilim bölücü üzerinden ADC girişine aktarılır ve FPGA bu voltajı ölçerek parlaklık seviyesini hesaplar.



Şekil 2.12. 1 LDR Sensörünün Pin Yapısı

Projede LDR sensöründen elde edilen analog voltaj değeri, FPGA üzerindeki XADC modülü tarafından 12 bit çözünürlükte sayısal değere dönüştürülmektedir. XADC'nin AUX giriş aralığı 0–1.0V olarak yapılandırılmıştır.

ADC dijital çıkış aralığı:

0V → 0

1.0V → 4095

LDR sensörünün direnci ışık seviyesine göre değişmektedir:

Karanlık → yüksek direnç → düşük voltaj → düşük ADC değeri

Aydınlık → düşük direnç → yüksek voltaj → yüksek ADC değeri

$$\text{Parlaklık} \approx \frac{ADC_Değeri \times 100}{4095}$$

Bu formül doğrudan uygulama kodunda da kullanılmıştır.

XADC IP Catalog Yapılandırması

Bu uygulamada LDR sıcaklık sensöründen alınan analog voltajın FPGA tarafından dijital olarak okunabilmesi için XADC (Xilinx Analog-to-Digital Converter) modülü kullanılmıştır. XADC, Vivado IP Catalog üzerinden projeye eklenen bir IP core'dur ve 12-bit çözünürlükte analog-dijital dönüşüm yapar.

IP Catalog Üzerinden XADC Ekleme bu adımlar uygulanmıştır;

Tools → IP Catalog menüsü açılır.

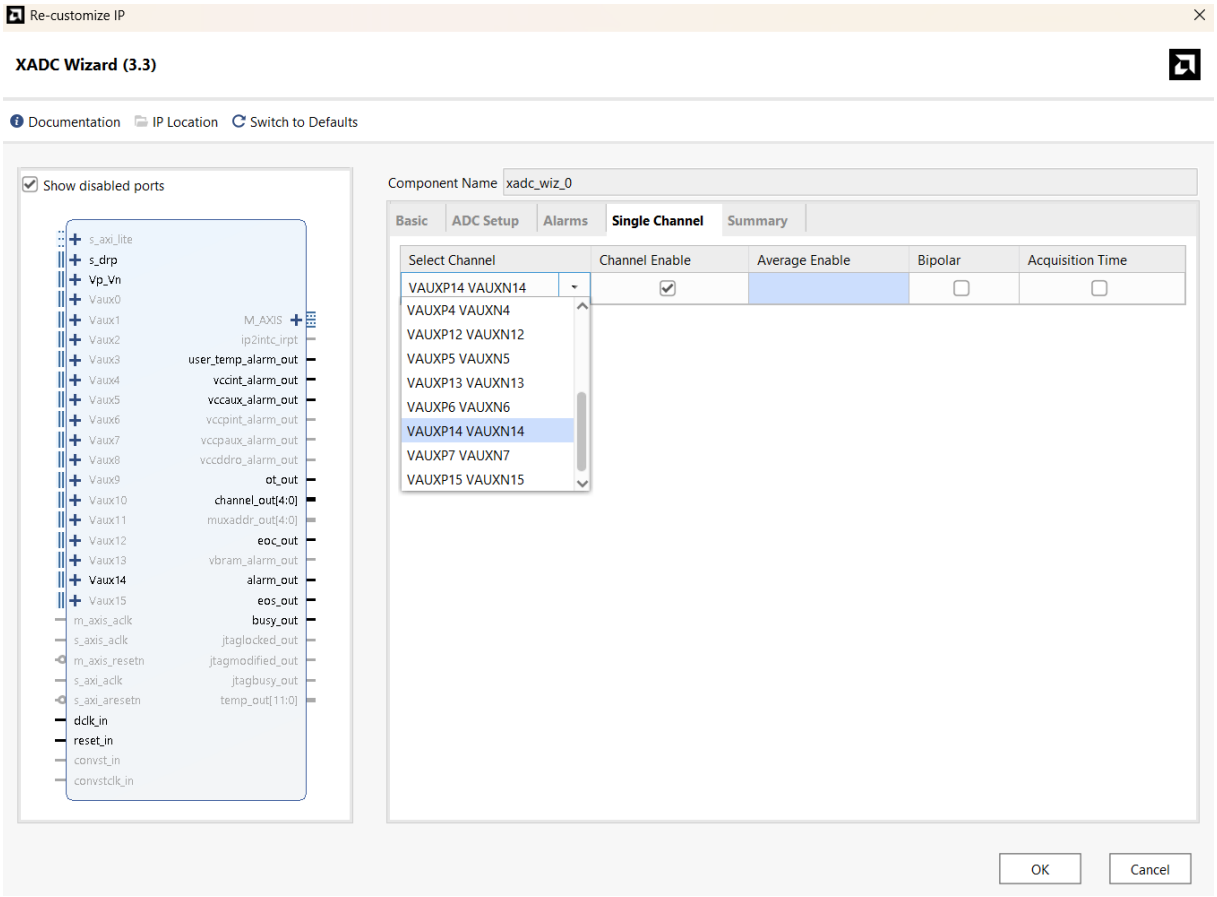
Arama kısmına “XADC” yazılır.

XADC Wizard seçilerek projeye eklenir.

Karşınıza gelen konfigürasyon ekranında

Single Channel Mode seçilir.

AUX girişlerinden kullanılacak kanal (örneğin AUX14P → L3 pinine bağlı LDR çıkışı AUX14N → GND pinine bağlı) aktif hale getirilir.



Şekil 2.12. 2 *XADC Wizard*

Yapılan projeye ait kodları Eğitim CD’sinde bulabilirsiniz.

2.13 Potansiyometre ile Analog Değer Ölçümü

Bu uygulamada, TRUE- ARTDAC7 FPGA geliştirme kartı üzerinde yer alan ADC modülü kullanılarak potansiyometrenin analog çıkış voltajı okunacak ve bu değer dijital forma dönüştürülerek FPGA üzerinde işlenecektir.

Bu uygulama, ADC ölçüm prensibinin anlaşılması, analog voltajların ölçeklenmesi ve dijital yüzdelik değerlere dönüştürülmesi konusunda temel bir deneysel altyapı sağlamaktadır.

Teorik Altyapı ve Çalışma Prensibi

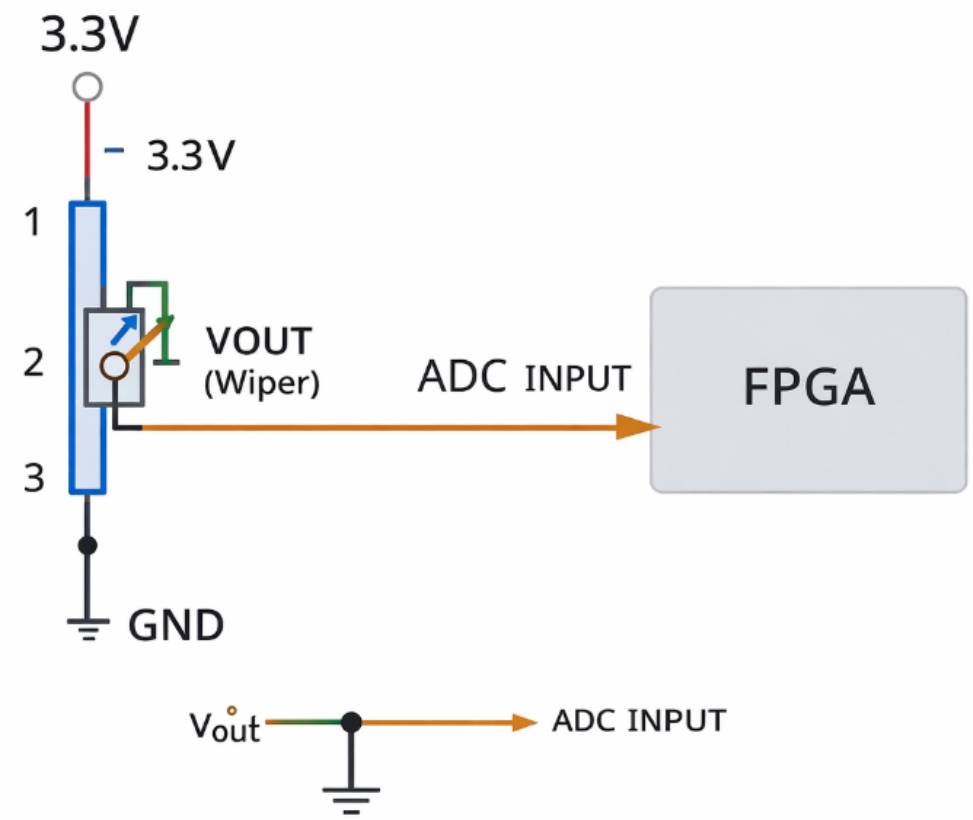
Potansiyometre (Temel Tanım)

Potansiyometre, mekanik olarak çevrildiğinde çıkış uçlarında değişen bir voltaj oluşturan üç terminalli bir analog bileşendir. Kullanıcı potansiyometreyi çevirdiğinde wiper uçtaki voltaj 0V ile 3.3V arasında lineer olarak değişir.

Pot minimumda → 0V

Pot ortada → 1.65V

Pot maksimumda → 3.3V



Şekil 2.13. 1 *Potansiyometre Pin Yapısı*

Projede potansiyometreden gelen analog voltaj, FPGA üzerindeki XADC modülü ile 12 bit çözünürlükte sayısal değere dönüştürülmektedir.

ADC'nin dijital aralığı

0V → 0

1.0V → 4095

Potansiyometre çıkışı tamamen lineer olduğu için ADC değeri doğrudan yüzde hesaplamaya uygundur.

Pot Değerinin Yüzde Hesabı

$$\text{Potansiyometre} \approx \frac{ADC_Değeri \times 100}{4095} \quad (\%)$$

ADC = 0 → %0

ADC = 2048 → %50

ADC = 4095 → %100

Bu hesaplama doğrudan uygulama kodunda da kullanılmıştır.

XADC IP Catalog Yapılandırması

Bu uygulamada Potansiyometreden alınan analog voltajın FPGA tarafından dijital olarak okunabilmesi için XADC (Xilinx Analog-to-Digital Converter) modülü kullanılmıştır. XADC, Vivado IP Catalog üzerinden projeye eklenen bir IP core'dur ve 12-bit çözünürlükte analog-dijital dönüşüm yapar.

IP Catalog Üzerinden XADC Ekleme bu adımlar uygulanmıştır;

Tools → IP Catalog menüsü açılır.

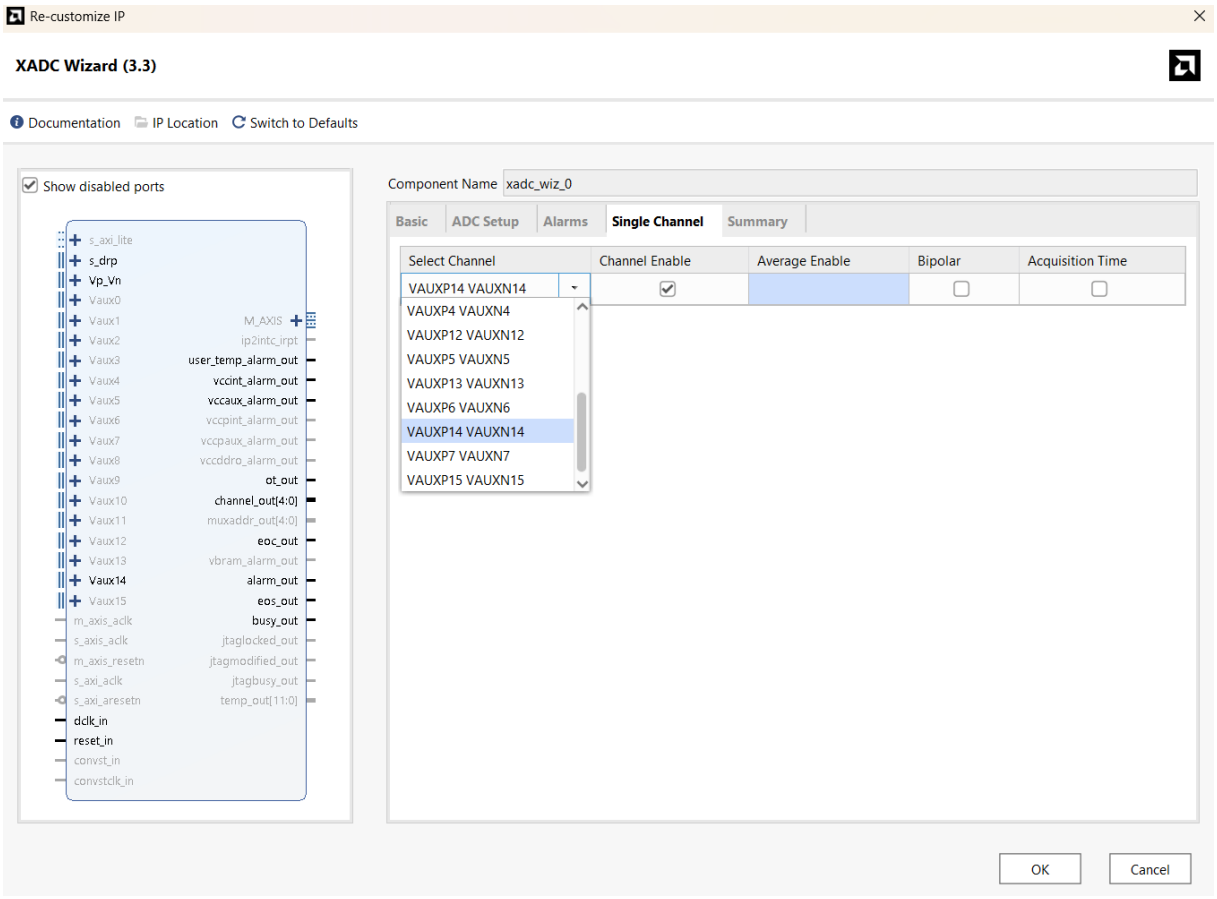
Arama kısmına "XADC" yazılır.

XADC Wizard seçilerek projeye eklenir.

Karşınıza gelen konfigürasyon ekranında

Single Channel Mode seçilir.

AUX girişlerinden kullanılacak kanal (örneğin AUX14P → L3 pinine bağlı Potansiyometre çıkışı AUX14N → GND pinine bağlı) aktif hale getirilir.



Şekil 2.13. 2 *XADC Wizard*

Yapılan projeye ait kodları Eğitim CD'sinde bulabilirsiniz.

2.14 UART TX ile 8-Bit Verinin HTerm ve DSView Ortamında Gösterimi

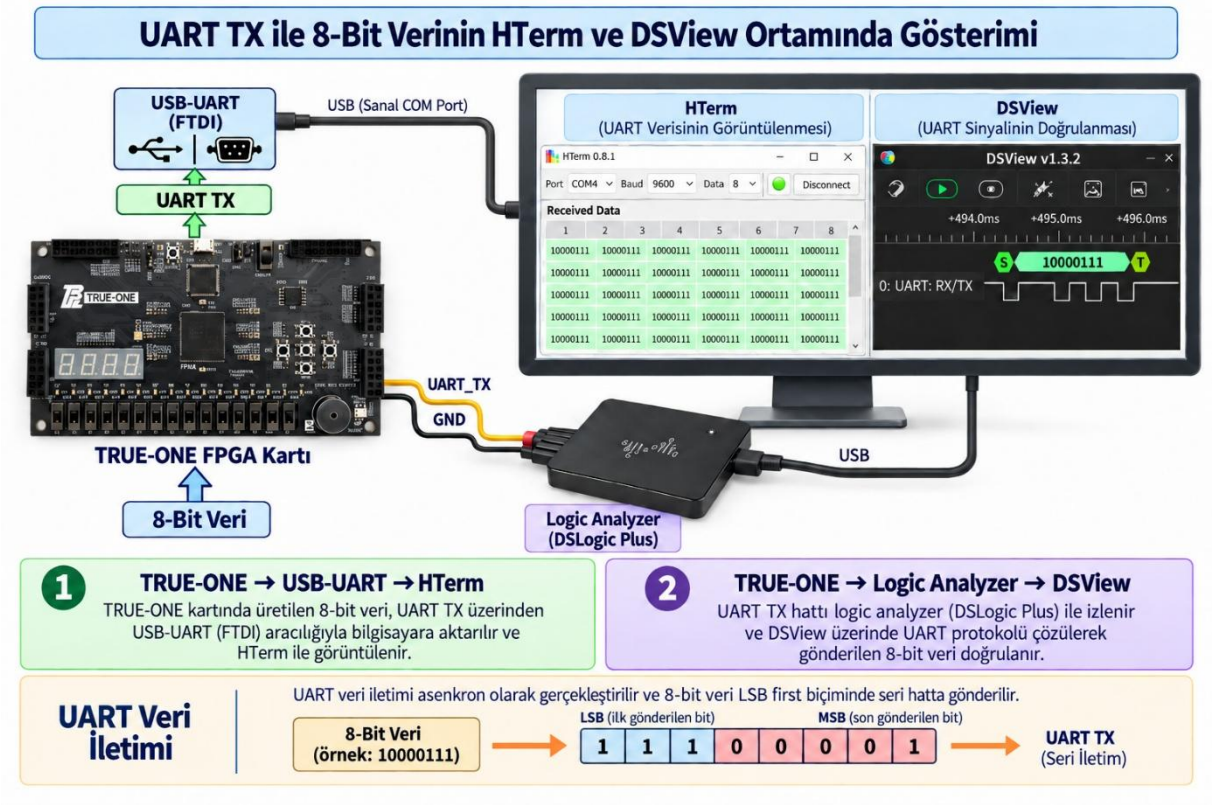
Bu uygulamada, TRUE-ONE FPGA geliştirme kartı üzerindeki 8-bit verinin UART TX modülü kullanılarak seri haberleşme üzerinden bilgisayara aktarılması amaçlanmaktadır. Gönderilen veri, bilgisayar tarafında HTerm uygulaması ile görüntülenmekte; aynı zamanda UART TX hattı logic analyzer kullanılarak izlenmekte ve DSView yazılımı üzerinden doğrulanmaktadır.

Bu sayede FPGA tarafından oluşturulan UART verisi iki farklı yöntemle gözlemlenmektedir. İlk yöntemde veri, TRUE-ONE kartı üzerinde bulunan USB-UART (FTDI) birimi aracılığıyla bilgisayara aktarılmakta ve HTerm üzerinde okunmaktadır. İkinci yöntemde ise aynı UART TX hattı, kart üzerindeki pin header çıkışından alınarak logic analyzer'a bağlanmakta ve DSView ortamında UART protokolü çözülerek incelenmektedir.

Teorik Altyapı ve Çalışma Prensibi

UART haberleşmesi asenkron bir yapıya sahiptir. Veri iletiminde ortak bir saat sinyali kullanılmaz; bunun yerine verici ve alıcı tarafının aynı baud rate değerinde çalışması gerekir. Bu uygulamada UART haberleşme hızı 9600 baud olarak belirlenmiştir. Veri iletimi, hat boşta iken lojik 1 seviyesinde bekleyen UART hattında, önce start biti, ardından 8-bit veri ve son olarak stop biti gönderilerek gerçekleştirilmektedir. Ayrıca veri iletimi LSB first biçiminde yapılmaktadır.

TRUE-ONE kartı üzerinde switchler ile belirlenen 8-bit veri FPGA içerisinde UART TX modülüne uygulanmakta ve seri hatta aktarılmaktadır. Üretilen bu seri sinyal bir yandan HTerm ile görüntülenirken, diğer yandan logic analyzer ile ölçülerek DSView üzerinde çözümlenmektedir. Böylece gönderilen verinin hem yazılım düzeyinde hem de fiziksel sinyal düzeyinde doğrulanması sağlanmaktadır.



Şekil 2.14.1. *UART TX ile 8-bit verinin HTerm ve DSView ortamında gösterimi*

Bu uygulama sonucunda TRUE-ONE FPGA geliştirme kartı ile üretilen 8-bit verinin UART üzerinden bilgisayara başarıyla aktarıldığı, HTerm uygulamasında görüntülendiği ve aynı verinin DSView ortamında doğrulandığı gözlemlenmiştir.

Yapılan projeye ait kaynak kodlar ve uygulama dosyaları Eğitim CD'sinde bulunmaktadır.

2.15 Step Motor

Bu uygulamada TRUE- ARTDAC7 FPGA geliştirme kartı kullanılarak 28BYJ-48 model step motorun sürülmesi amaçlanmaktadır. Uygulama kapsamında FPGA üzerinde üretilen faz dizileri aracılığıyla step motorun kontrollü bir şekilde döndürülmesi, dönüş yönünün değiştirilmesi ve hızının adım periyodu üzerinden ayarlanması hedeflenmektedir. Bu sayede FPGA tabanlı sistemlerde zamanlama (clock), bölücü (tick) ve sıralı kontrol mantığı pekiştirilecektir.

Bu uygulamada step motor, doğrudan FPGA çıkışları ile sürülmemiş; motor bobinlerinin ihtiyaç duyduğu akımı sağlayabilmek amacıyla SBT0811 sürücü modülü kullanılmıştır. FPGA tarafından üretilen lojik seviyedeki kontrol sinyalleri (IN1–IN4), SBT0811 sürücü kartına uygulanarak step motorun bobinleri sırasıyla enerjilendirilmiş ve motorun dönmesi sağlanmıştır.

Teorik Altyapı ve Çalışma Prensibi

Step motorlar, girişlerine uygulanan faz sinyallerinin sırasına göre rotorunu adım adım döndüren elektromekanik motorlardır. Her bir adım, motorun sabit bir açısal miktar kadar ilerlemesini sağlar. Bu özellik, step motorların hassas konumlama gerektiren uygulamalarda yaygın olarak kullanılmasını mümkün kılmaktadır

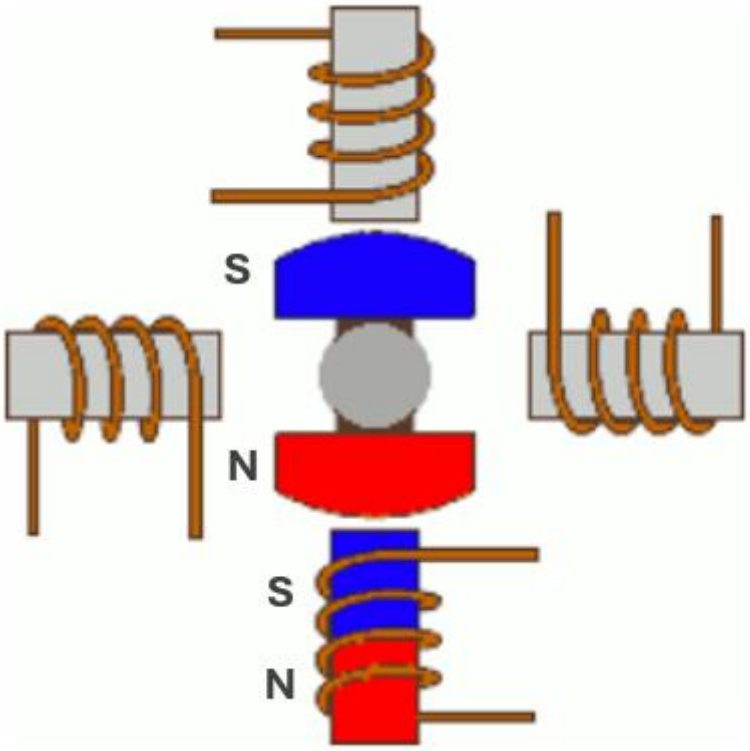
28BYJ-48 model step motor, genellikle unipolar yapıda olup dört fazlı olarak çalışır. Motorun dönebilmesi için bobinlerin belirli bir sırayla enerjilendirilmesi gerekir. Bu uygulamada, 28BYJ-48 için yaygın olarak kullanılan half-step sürme yöntemi tercih edilmiştir. Half-step yönteminde motor, toplam 8 farklı faz durumundan oluşan bir dizi ile sürülür. Bu dizinin sıralı uygulanması motorun saat yönünde, dizinin ters yönde uygulanması ise motorun saat yönünün tersine dönmesini sağlar.

FPGA içerisinde sistem saatinden türetilen bir tick (adım tetikleme) sinyali üretilir. Her tick sinyali geldiğinde faz dizisinde bir sonraki duruma geçilir. Böylece motorun dönme hızı tick sinyalinin periyodu ile doğrudan kontrol edilebilir. Tick frekansı artırıldığında motor hızlanır, azaltıldığında ise motor yavaşlar.

Motorun dönüş yönü ise faz dizisinin ilerleme yönü değiştirilerek kontrol edilir. Faz dizisi ileri sayıldığında motor saat yönünde geri sayıldığında ise saat yönünün tersinde döner.

Özellik	Açıklama
Çalışma Gerilimi	5V DC
Faz Sayısı	1/64
Hız Adım Değişim oranı	1/64
Adım Açısı	5.625°/64
Frekans	100 Hz

Tablo 2.15. 1 Step Motor Teknik özellikler



Şekil 2.15. 1 Step Motor Yapısı

Yapılan projeye ait kodları Eğitim CD'sinde bulabilirsiniz.

2.16 FPGA Tabanlı Analog Dalga Şekillerinin Üretilmesi

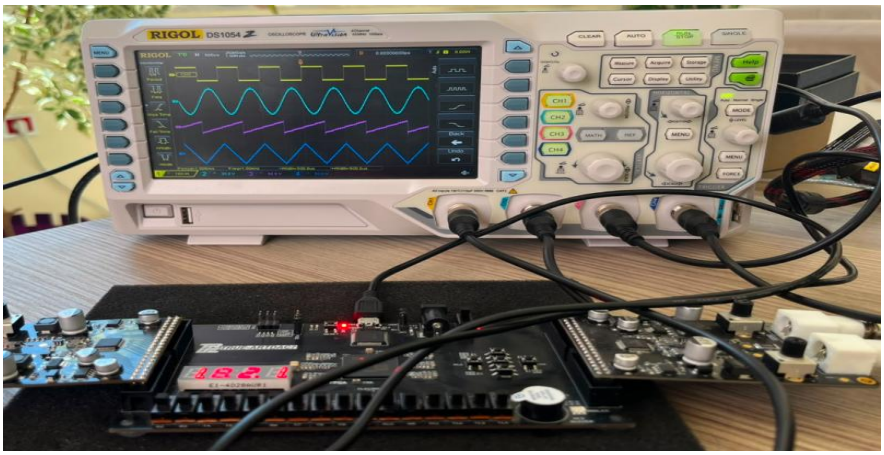
Bu uygulamada TRUE-ARTDAC7 FPGA geliştirme kartı ve karta doğrudan bağlanan iki adet AD9767 tabanlı DAC modülü kullanılarak sinüs, üçgen, kare ve testere dişi dalga şekillerinin eş zamanlı olarak üretilmesi amaçlanmaktadır. FPGA içerisinde sayısal örnekler hâlinde oluşturulan dalga şekilleri DAC modülleri aracılığıyla analog gerilim sinyallerine dönüştürülmekte ve dört kanallı bir osiloskop üzerinden görüntülenmektedir.

Tasarım karta yüklendiğinde dört farklı dalga şekli doğrudan ve sürekli olarak üretilmektedir. Osiloskop bağlantıları sırasıyla CH1 sinüs, CH2 üçgen, CH3 kare ve CH4 testere dişi sinyali gösterecek şekilde yapılmıştır.

Teorik Altyapı ve Çalışma Prensibi

FPGA çıkışları sayısal lojik seviyeler üretir. Sinüs veya üçgen gibi sürekli değişen analog dalga şekillerinin elde edilebilmesi için FPGA tarafından oluşturulan sayısal örneklerin bir dijital-analog dönüştürücüye aktarılması gerekir. Bu uygulamada kullanılan AD9767, iki kanallı ve 14 bit çözünürlüklü bir DAC yapısına sahiptir. İki DAC modülünün birlikte kullanılmasıyla toplam dört bağımsız analog çıkış elde edilmektedir.

FPGA içerisinde tanımlanan örnek sayacı veya faz akümülatörü, belirlenen güncelleme periyodunda ilerletilir. Sinüs dalgasının örnek değerleri bir arama tablosundan (LUT) alınırken üçgen dalga artan ve azalan doğrusal örneklerden oluşturulur. Kare dalga iki farklı sayısal seviye arasında geçiş yapılarak, testere dişi dalga ise örnek değerlerinin minimumdan maksimuma doğru doğrusal biçimde artırılıp yeniden başlangıç değerine döndürülmesiyle elde edilir. Oluşturulan 14 bitlik örnekler DAC veri girişlerine uygulanır ve analog çıkış katı üzerinden osiloskob'a aktarılır.



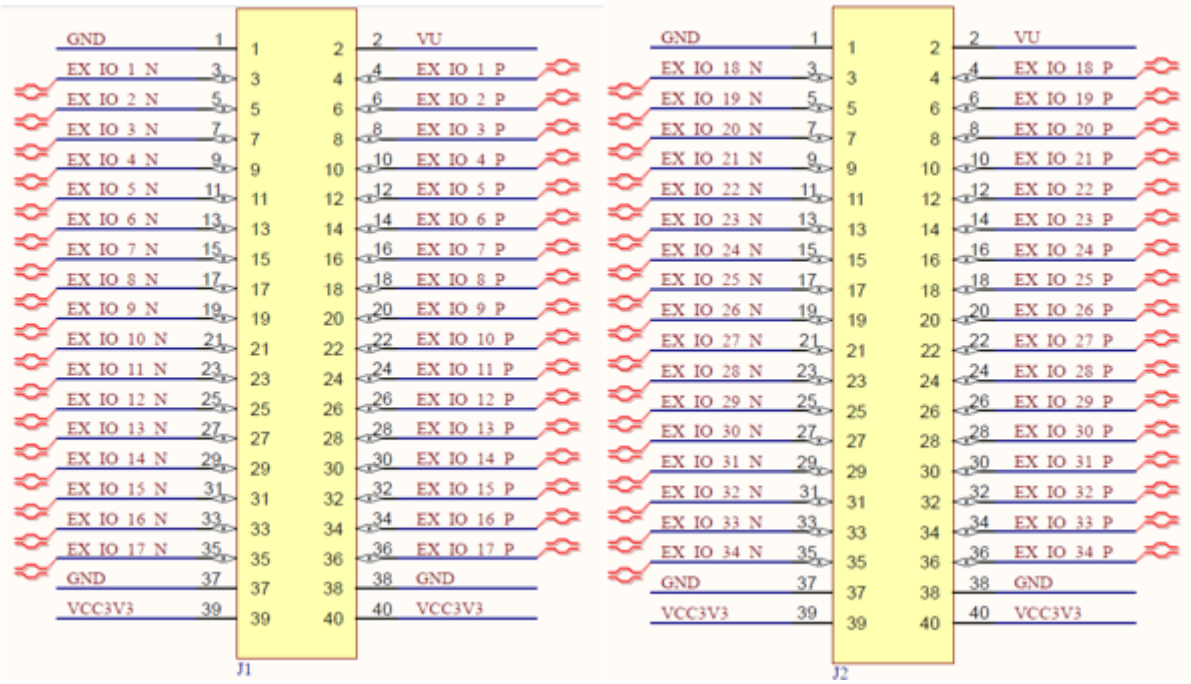
Özellik	Açıklama
Kullanılan geliştirme kartı	TRUE-ARTDAC7
DAC modülü	AD9767 tabanlı, çift kanallı
DAC çözünürlüğü	14 bit
Azami örnekleme hızı	125 MSPS
Kullanılan DAC modülü sayısı	2
Toplam analog çıkış sayısı	4
Üretilen dalga şekilleri	Sinüs, üçgen, kare ve testere dişi
Kart sistem saati	100 MHz
Analog çıkış aralığı	−5 V ile +5 V
Görüntüleme cihazı	Dört kanallı osiloskop

Tablo 2.16.1 Analog dalga şekli üretim uygulamasının teknik özellikleri

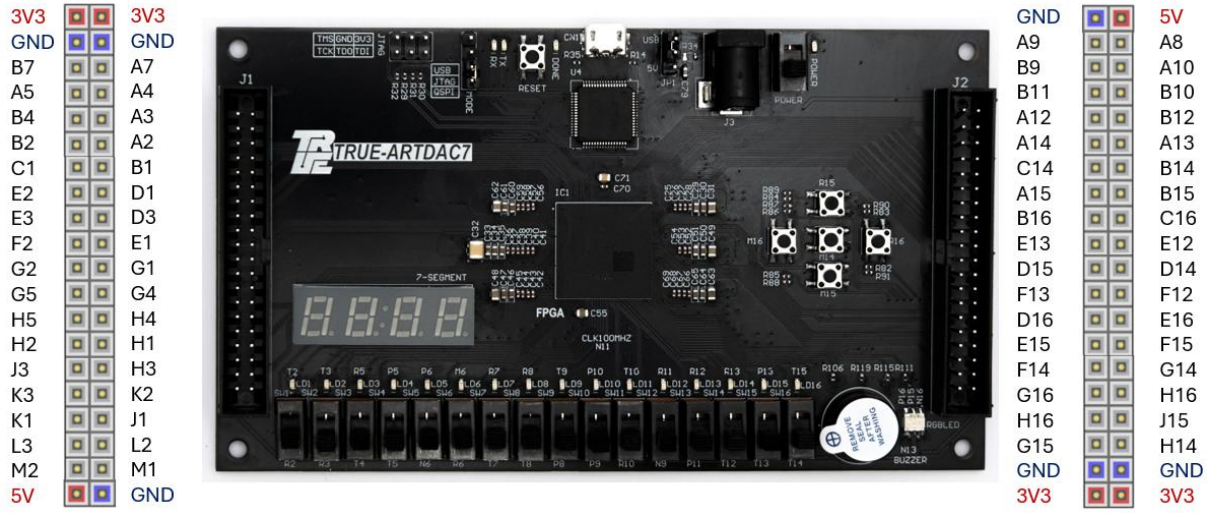
DAC Modüllerinin TRUE-ARTDAC7 Kartına Bağlanması

TRUE-ARTDAC7 kartında bulunan J1 ve J2 genişleme konnektörleri, haricî modüllerin FPGA giriş/çıkış hatlarına doğrudan erişebilmesi için tasarlanmıştır. Bu uygulamada birinci DAC modülü J1 konnektörüne, ikinci DAC modülü ise J2 konnektörüne bağlanmıştır. Modüller takılırken besleme ve toprak pinlerinin yönüne dikkat edilmeli, bağlantı işlemi kart enerjisiz durumdayken gerçekleştirilmelidir.

Konnektörlerde yer alan FPGA I/O hatları, VHDL tasarımındaki DAC veri, yazma ve saat sinyallerine XDC dosyası üzerinden atanmıştır. Böylece DAC modülleri ilave kablolama gerektirmeden kartın iki yanındaki pin header yapılarına doğrudan takılabilmektedir.

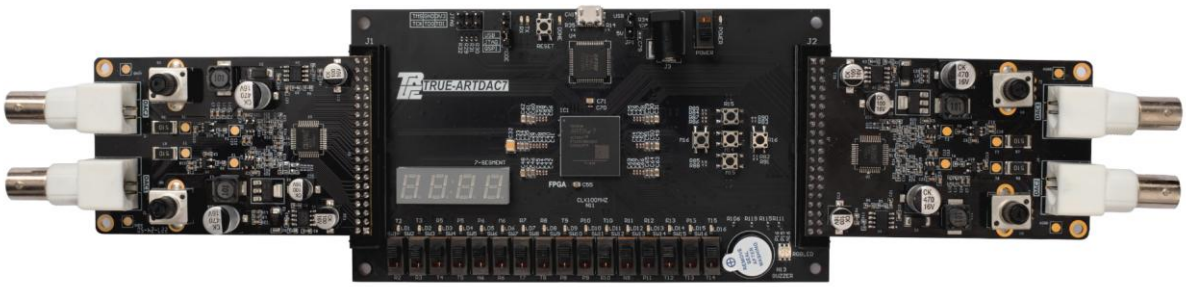


Şekil 2.16. 1 *TRUE-ARTDAC7 J1 ve J2 genişleme konnektörlerinin şematik pin yapısı*



Şekil 2.16. 2 *TRUE-ARTDAC7 kartı J1 ve J2 konnektörlerinin FPGA pin karşılıkları*

Şekil 2.16.2 üzerinde J1 ve J2 konnektörlerine ait besleme, toprak ve FPGA I/O pinleri gösterilmektedir. DAC modülleri, kart üzerindeki bu pin düzeniyle uyumlu olacak biçimde yerleştirilmiş ve sayısal veri hatları doğrudan FPGA tarafından sürülmüştür.



Şekil 2.16. 3 *İki adet DAC modülünün TRUE-ARTDAC7 geliştirme kartına bağlanması*

Osiloskop Bağlantısı ve Uygulama Adımları

- TRUE-ARTDAC7 kartının enerjisi kesilir ve DAC modülleri J1 ile J2 konnektörlerine doğru yönde takılır.
- DAC modüllerinin dört BNC çıkışı, dört kanallı osiloskobun CH1–CH4 girişlerine bağlanır.
- Vivado projesine Eğitim CD'sinde bulunan VHDL kaynak dosyaları ve TRUE-ARTDAC7 XDC dosyası eklenir.
- Sentez, implementasyon ve bitstream oluşturma işlemleri tamamlanarak tasarım karta yüklenir.
- Kart programlandıktan sonra dalga şekilleri otomatik olarak üretilmeye başlar.
- Osiloskopta CH1 sinüs, CH2 üçgen, CH3 kare ve CH4 testere dişi dalga şekilleri gözlemlenir. Gerekli görüldüğünde zaman/div ve volt/div ayarları değiştirilerek sinyaller ekrana uygun biçimde yerleştirilir.

Uygulama sonucunda FPGA üzerinde oluşturulan dört farklı sayısal dalga şekli, iki adet AD9767 tabanlı DAC modülü aracılığıyla analog sinyallere dönüştürülmüş ve osiloskopta eş zamanlı olarak görüntülenmiştir. Bu çalışma ile LUT kullanımı, sayısal örnek üretimi, zamanlama kontrolü, paralel DAC veri aktarımı ve sayısal verinin analog sinyale dönüştürülmesi konuları uygulamalı olarak pekiştirilmiştir.

Yapılan projeye ait kodları Eğitim CD'sinde bulabilirsiniz.

Faz adımı hesaplama

Faz adımı = İstenen çıkış frekansı × 2^{32} / Güncelleme frekansı

Faz adımı = 1.000 × 2^{32} / 1.000.000 ≈ 4.294.967