3. Temel VHDL Bileşenleri

VHDL ile tasarım yapılmadan önce VHDL dilinin temel mantığını kavramak gerekmektedir. Her şeyden önce VHDL ile yazılan kodların herhangi bir programlama dilindeki gibi yorumlanıp, derlenip çalıştırıldığı **düşünülmemelidir.** VHDL ile yazılan kodlara karşılık FPGA üzerinde belirtilen işi yapacak bir mantık devresi sentezlenmektedir. Yapılan tasarıma karşılık fiziksel bir devre oluşturulmaktadır. VHDL ile tasarım yapılırken bu durum asla unutulmamalıdır.

Bir VHDL kodu toplamda 3 ana kısımdan oluşmaktadır (Şekil 3-1). Bunlar sırasıyla:

- Kütüphane tanımlama kısmı
- Varlık oluşturma kısmı
- Mimari tasarım kısmı.

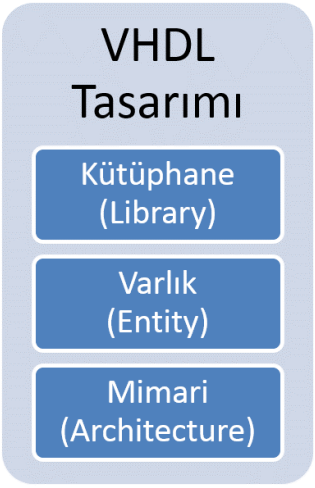
Bu bölümde VHDL koduna ait yukarıda verilen kısımlar hakkında açıklamalar yapılmış olup VHDL diline ait tanımlamalar ve yazım kuralları bir sonraki bölümde anlatılmıştır. Bu kısım okuyucuya bir VHDL kodunun neye benzediği ve hangi kısımlardan oluştuğu hakkında genel bir fikir verme amacı taşımaktadır.

3.1. VHDL Kodunun Bölümleri

İlk kısım kütüphanelerin tanımlandığı kısımdır. Burada hazır kütüphaneler kullanabileceğimiz gibi, kendi oluşturduğumuz kütüphaneleri de ekleyebiliriz.

İkinci kısımda ise **`entity`** olarak tanımlanan tasarladığımız yapının ana hatlarının tanımlandığı kısım bulunmaktadır. Burada yapılacak tanımlamalar ile tasarlanan varlığın hatları belirlenir. Bir benzetme yapmak gerekirse; yaptığınız çalışmayı bir televizyon olarak kabul edersek **`entity`** kısmında o televizyona ait açma kapama düğmelerini, anten girişini, kanal değiştirme düğmelerini tanımlarsınız.

Üçüncü kısım olan **`architecture`** (mimari) kısmında ise yaptığımız tasarımın içyapısını şekillendiririz. Bir önceki paragrafta yapılan benzetimden devam edecek olursak; televizyonumuzun içinde olan o karmaşık yapının tümü olarak tanımlayabiliriz.



Şekil 3-1 VHDL tasarımının temel bileşenleri

3.1.1. Kütüphane (Library) Bildirimi

Kütüphane bildirimi kısmında tasarımda kullanılacak kütüphanelerin tanımlamaları yapılmaktadır. Bu kısma ait örnek söz dizimi aşağıda verilmiştir:

```
library kutuphane_adi;
use kutuphane_adi.paket_adi.paket_bolumu;
use kutuphane_adi.paket_adi.paket_bolumu;
…
…
use kutuphane_adi.paket_adi.paket_bolumu;
```

Verilen sözdiziminden görüleceği üzere **library** kelimesi ile birlikte kullanılacak olan kütüphane adı yazılmaktadır. **use** kelimesi ile de kullanılacak kütüphane içerisinde bulunan ilgili paketin adı ve paket ile ilgili bölüm yazılmaktadır.

Aşağıda örnek bir kütüphane bildirimi verilmiştir. Verilen tanımlamaya göre kullanılacak olan kütüphane adı **IEEE** olup kullanılacak paket olarak **STD_LOGIC_1164** seçilmiştir. **ALL** ifadesi ile de bu paketin bütün içeriğinin kullanılacağı belirtilmiştir.

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

Kütüphane tanımlama işlemini kavramak adına şu şekilde bir benzetme yapabiliriz:

- Kütüphaneye gidip (**IEEE**),
- Kütüphanede bulunan ilgili kitabı seçip (**STD_LOGIC_1164**),
- Seçilen kitabın hangi kısımlarını kullanacağımızı (**ALL**) bildirmek

Tasarım sırasında hazır kütüphaneler kullanılabileceği gibi kendi oluşturduğunuz kütüphaneleri de kullanabilirsiniz. Bu konu hakkında gerekli bilgiler kitabın ilerleyen bölümlerinde sunulacaktır. Bu aşamada var olan hazır kütüphaneler kullanılacaktır.

3.1.2. Varlık (Entity) Bildirimi

Tasarıma ait giriş ve çıkış bilgileri, **entity** kısmında belirtilmektedir. **entity**'de adlandırma işlemi VHDL sözdizimine uygun olmak şartıyla kullanıcı tarafından istenilen şekilde yapılabilir. **entity** tanımlamaya ait söz dizimi aşağıda verilmiştir.

```
entity varlik_adi is
port(
      port_adi : [port_modu] tip_adi;
      port_adi : [port_modu] tip_adi
);
end varlik_adi;
```

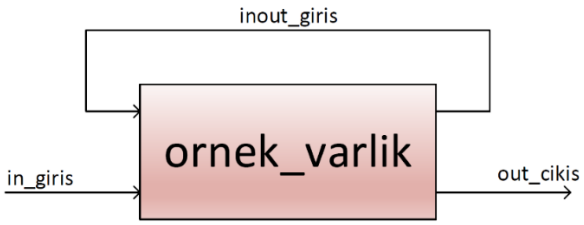
Varlığa ait tüm giriş ve çıkışlar **port** adı verilen tanımlama içerisinde belirtilmektedir. Bu kısımda yapılan tanımlamaların türleri (giriş, çıkış, tampon (buffer), giriş-çıkış) **port_modu** adı verilen kısımda yapılmaktadır. VHDL dilinde port türü tanımlanmamış ise **in** olarak kabul edilir. Portların tanımlanabileceği türler Tablo 3-1'de verilmiştir.

Tablo 3-1 Varlık Port Modları

Mod	Amaç
in	Tanımlanan portun giriş olduğunu gösterir.
out	Tanımlanan portun çıkış olarak kullanıldığını belirtir. Bir port çıkış olarak tanımlandıysa o porttan veri okuma işlemi yapılamaz. Sadece veri yazılabilir.
inout	Tanımlanan portun hem giriş hem de çıkış olarak kullanılabileceğini belirtir.
buffer	**Out** durumundan farklı olarak, okuma işlemi de yapılabilir.

Şekil 3-2'de örnek bir varlık gösterimi verilmiştir. **ornek_varlik** olarak adlandırılan varlığımızın tüm portlarının tipi **std_logic'** dir.

```
entity ornek_varlik is
port(
        in_giris :in std_logic;
        inout_giris :inout std_logic;
        out_cikis :out std_logic
);
end ornek_varlik;
```



Şekil 3-2 ornek_varlık port gösterimi

3.1.3. Mimari (Architecture)

Mimari, aslında yaptığımız tasarımın en önemli kısmını oluşturmaktadır. Şu ana kadar olan kısımlarda kullanılacak kütüphane(ler), tasarıma ait giriş-çıkış birimleri tanımlanmıştır. Bundan sonraki kısımda ise yaptığımız tasarımın içyapısını oluşturmakta ve tasarımın davranışı belirlemekteyiz. Aşağıda mimari kısmına ait genel bildirim yapısı verilmiştir.

```
architecture mimari_adi of varlik_adi is
        [signal tanımlama]
        [constant tanımlama]
        [type tanımalama]
        [component tanımlama]
        [attribute tanımlama]
begin
        {COMPONENT örnek ifadeleri ;}
        {PROCESS ifadeleri;}
        {GENERATE ifadeleri ;}
        {Eş zamanalı atama ifadeleri;}
```

Tanımlama Bölgesi

Mimari Bileşenleri

```
end mimari_adi;
```

architecture iki kısımdan oluşmaktadır. Birinci bölge sinyal, sabit, tip, bileşen ve özelliklerin tanımlandığı, tanımlama bölgesidir. Bu bölge mimarinin tanımlanmaya başlandığı satırda bulunan **is**'den sonra başlamaktadır. Tanımlama işlemi ilk **begin** deyimine kadar yapılmalıdır. **begin** ile **end** arasında kalan ikinci bölge mimari bileşenlerin, atamaların, **process** ve **generate** işlemlerinin yapıldığı bölgedir.

Mimari kısmında atama işlemleri yapılırken **<=** operatörü kullanılır. Bu operatörün sağ tarafında yazılan ifade sol tarafındaki ifadeye atanır. Akılda kalması açısından **<=** operatörünü bir ok olarak düşünebiliriz ve atama işleminin okun gösterdiği yönde yapıldığını söyleyebiliriz. Atama işlemi **entity** kısmında tanımlanan **port**'lar ile olacağı gibi, **architecture**'ın tanımlama kısmında tanımlanan sinyaller arasında da olabilir. Eğer atama işlemi **port**'lar ile alakalı ise şu durumlara dikkat edilmesi gerekmektedir:

- Eğer **port** giriş (**in**) olarak tanımlanmışsa atama operatörünün "**<=**" sağ tarafında yer alabilir. Giriş olarak tanımlanmış bir **port**'a değer atanamaz.
- Eğer **port** çıkış (**out**) olarak tanımlanmışsa atama operatörünün "**<=**" sol tarafında yer alabilir. Çıkış olarak tanımlanmış bir **port** üzerinden veri okunamaz.
- Eğer **port** giriş-çıkış (**inout**) olarak tanımlanmışsa atama operatörünün "**<=**" her iki tarafında da yer alabilir.

Atamalarla ilgili dikkat edilecek bir diğer önemli nokta ise, bir **port**'a ya da bir sinyale ancak tek bir kaynaktan atama yapılabiliyor olduğudur.

Aşağıda örnek bir mimari bildirimi yapılmaktadır. Mimaride **in_giris** değeri, **<=** operatörünün sağ tarafında yer almaktadır ve **inout_giris** değerine atanmaktadır. **out_cikis** değeri, **<=** söz diziminin sol tarafında yer almakta ve **inout_giris** değeri atanmaktadır. **inout_giris** degeri ise **<=** söz diziminin her iki yanında yer alabilmektedir.

```
architecture behavioral of ornek_varlik is
begin
      inout_giris <= in_giris;
      out_cikis <= inout_giris;
end behavioral;
```

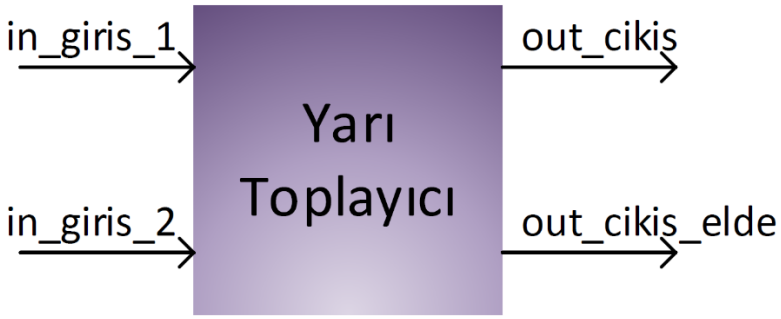
3.2. Örnek Uygulama: Yarı Toplayıcı Devresi

Yarı toplayıcı devresi dışardan elde girişi olmadan sadece ikili (binary) giriş değerlerini toplayarak çıkışa aktarır. Tablo 3.2'de yarı toplayıcı doğruluk tablosu verilmiştir.

Tablo 3-2 Yarı Toplayıcı Doğruluk Tablosu

Girişler		Çıkışlar	
in_giris_1	**in_giris_2**	**out_cikis**	**out_cikis_elde**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tablo 3-2 ve Şekil 3-3.'den görüleceği üzere yarı toplayıcı devresi birer bitlik iki girişe ve toplam sonucu ile birlikte elde değerinin tutulduğu iki çıkışa sahiptir.



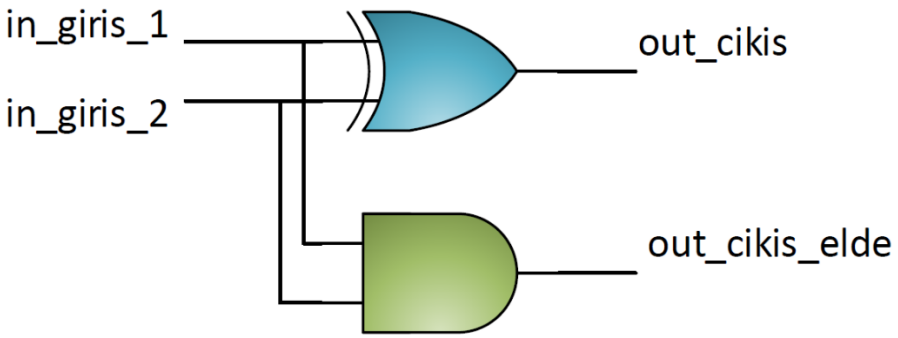
Şekil 3-3 Yarı toplayıcı giriş-çıkış yapısı gösterimi

VHDL dilinde yarı toplayıcıya ait varlık (entity) tanımlama aşamasında iki girişli ve iki çıkışlı bir tanımlama yapılmalıdır. **yari_toplayici** varlığına ait port tanımlama işlemi aşağıdaki gibi yapılmaktadır. Tanımlamada **in_giris_1** ve **in_giris_2**, **std_logic** tipinde **in** moduna tanımlanmış bağlantı noktalarıdır. **out_cikis** ve **out_cikis_elde**, **std_logic** tipinde **out** modunda tanımlanmış bağlantı noktalarıdır.

```
entity yari_toplayici is
port (
      in_giris_1 : in  std_logic;
      in_giris_2 : in  std_logic;
      out_cikis : out std_logic;
      out_cikis_elde : out std_logic
);
end yari_toplayici;
```

Yarı toplayıcı çıkış fonksiyonları aşağıda verilmiştir. Bu fonksiyonlar aynı zamanda tasarlanacak devrenin davranışını göstermektedir. Şekil 3-4'den de görüleceği üzere **out_cikis** değerini elde etmek için giriş değerlerini **ÖZEL VEYA** (XOR) işlemine tabi tutmak gerekmektedir. Aynı şekilde **out_cikis_elde** değerini elde etmek için giriş değerlerini **VE** (AND) işlemine tabi tutmak gerekmektedir.

```
out_cikis <= in_giris_1 xor in_giris_2;
out_cikis_elde <= in_giris_1 and in_giris_2;
```



Şekil 3-4 Yarı toplayıcı temel mantık kapıları ile gösterimi

Yarı toplayıcıya ait mimari tasarımında yukarıda verilen çıkış fonksiyonları kullanılarak tanımlama yapılmalıdır. Aşağıda yarı toplayıcıya ait mimari tanımlama sözdizimi verilmiştir. Sözdiziminden de görüleceği üzere mimari tanımlama bölgesinde hiçbir tanımlama yapılmamıştır. İkinci bölgede ise çıkış portlarına, giriş fonksiyonlarının çıkış fonksiyonlarına tabi tutulduktan sonraki atama işlemleri yapılmıştır.

```
architecture Behavioral of yari_toplayici is
begin
      out_cikis <= in_giris_1 xor in_giris_2;
      out_cikis_elde <= in_giris_1 and in_giris_2;
end Behavioral;
```

Aşağıda yarı toplayıcı devresinin **yari_toplayici.vhd** VHDL kodu verilmiştir. Kodda 1-2. satırlar arasında kullanılacak olan kütüphanelerin bildirimi yapılmıştır. **yari_toplayici** varlığına ait port tanımlama işlemleri 5-10. satırlar arasında yapılmıştır. 17-18. satırlarda **yari_toplayici** varlığının davranışı tanımlanmıştır.

```
1. library IEEE;
2. use IEEE.STD_LOGIC_1164.all;
3.
4. entity yari_toplayici is
5.   Port (
6.     in_giris_1 : in std_logic;
7.     in_giris_2 : in std_logic;
8.     out_cikis : out std_logic;
9.     out_cikis_elde : out std_logic
10.  );
11.end yari_toplayici;
12.
13.architecture Behavioral of yari_toplayici is
14.
15.begin
```

```
16.
17.  out_cikis <= in_giris_1 xor in_giris_2;
18.  out_cikis_elde <= in_giris_1 and in_giris_2;
19.
20.end Behavioral;
```