HDL (Hardware Description Language)
Donanım Tanımlama Dili

Dijital devreleri modellemek için kullanılır.

- Used to describe the structure and behavior of electronic circuit - digital logic circuits
  - Formal description of an electronic circuit
  - Automated analysis and simulation
  - allows for the synthesis of an HDL description into netlist.

a specification of physical electronic components and how they are connected together

Karşılaştırıcı Simülasyon (Test Bench)

`timescale 1ns/1ps
zaman ölçeği tanımlanır
1 nanosaniyelik analizlerde çalışır

module stimulus;
test benchin ismi girilir.

hafızalı — reg x; reg y; } inputlar reg ile gösterilir

hafızasız — wire z; } outputlar wire ile gösterilir

comparator uut(.x(x), .y(y), .z(z));
modülün bir örneği oluşturulur.
• nokta ile yazılanlar sana modülündeki değişkenler
parantez içindekiler de testbenchdeki değişkenleri
eşler.

initial begin
x = 0; y = 0; } x ve y 0' sıfıra eşitlenir.
#20 x = 1; → 20 nanosaniye sonra durum x=1 y=0
#20 y = 1; → 20 nanosaniye sonra x=1 y=1
#20 y = 0;
#20 x = 1;
#40; → 40 nanosaniye bekleme

20 nanosaniye sonra x=1 y=0

20 nanosaniye sonra x=1 y=0

end

initial begin
$monitor("x=%d, y=%d, z=%d", x,y,z);
end
endmodule

TCL consola verileri yazar.

iki initial de aynı anda çalışır.

Yanlış modüler yapıda tasarlanmıştır.
Portlar input, output, inout, olabilir
Portlar modülün giriş çıkış bağlantılarını belirtir
Port tanımlamları modül tanımının parantezleri arasına yazılır.
input dahili wire olmalı
hariçi olacak wire veya reg
output dahili wire veya reg
hariçi wire
inout dahili ve hariçi wire olmalıdır.

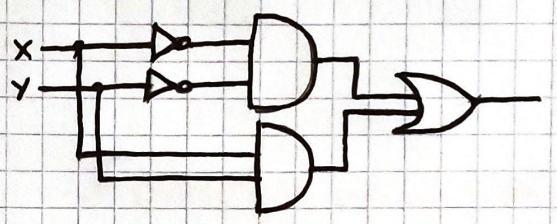
Karşılaştırıcı (Comparator)

Hem x hem y girişleri aynı olduğunda çıkışı 1 olan bir karşılaştırıcı devre

X	Y	Output
0	0	1
0	1	0
1	0	0
1	1	1

X \ Y	0	1
0	1	0
1	0	1

$$Output = \overline{X}\,\overline{Y} + XY$$



module comparator(
input x,
input y,
output z
);
assign z = (~x & ~y) | (x & y);
endmodule

giriş çıkış portlarını tanımladık

atomo işlemi yaptık

Kombinasyonel mantık uygulamasıdır.
Her clock tetiklemede kullanılır.

VERILOG

Identifiers (Tanımlayıcılar)

- comparator modül tanımlamak için kullanılmıştır. x,y girişlerini z çıkışı tanımlamak için kullanılmıştır
  - Tanımlayıcılar alfabetik karakter yada _ alfabesi ile başlamalıdır.
  - Tanımlayıcılar rakam, $ veya karakterler içerebilir.
  - Karakter tanımlamak istediğiniz başına \ ters slaş eklemelidir.

Keywords (Ayrılmış Kelimeler)

Verilogun yapısında bulunan kelimelerdir. module assign gibi
// yorum satırı
/* yorum */ çoklu yorum satırı
Boşluklar verilog yorumlarda yoktur.
Büyük küçük harf duyarlıdır.
Tüm anahtar kelimeler küçük harfle başlar

Vektörler

Çok bitli veri yollarını temsil etmektedir

Reg [7:0] Sekiz bitlik veri yolu MSB: 7
LSB: 0

```
module comparator2bit (
    input [1:0] x,   → 2 bitlik bir giriş
    input [1:0] y,   → 2 bitlik bir giriş
    output z
    );

assign z = (x[0] & y[0] & x[1] & y[1]) |
           (~x[0] & ~y[0] & x[1] & y[1]) |
           (~x[0] & ~y[0] & ~x[1] & ~y[1]) |
           (x[0] & y[0] & ~x[1] & ~y[1]);

endmodule
```

Vector	Scaler
wire [3:0] x;	wire x;
x[3] x[2] x[1] x[0]	
reg [3:0] y;	reg y;
3 2 1 0	

Kapı Seviyesinde Programlama

assign ile yaptığımız ifadeleri dijital kapılar ile tanımlayabiliriz.

```
module gate(
    input x,
    input y,
    output z
    );

wire x_, y_, p, q;
not (x_, x);
not (y_, y);
and (p, x, y);
and (q, x_, y_);
or (z, p, q);
endmodule
```

giriş portunu tanımladık

çıkış portunu tanımladık

User Defined Primitives (UDP's)

Kullanıcı tanımlı programlama

Normalde yapılıyor fakat Vivado uygulaması desteklemiyor.

```
module comparator(
    input x,
    input y,
    output z
    );
compare co(z, x, y);
end module

primitive compare (out, in1, in2);
output out;
input in1, in2;

table
    0 0 : 1;
    0 1 : 0;
    1 0 : 0;
    1 1 : 1;
endtable
endprimitive
```

- primitive tüm verileri tanımlamadan desteklemiyor, ve desteklemeye yacak.
- parantez içinde ilk önce çıkışı yazmak zorundayız.

OPERATÖRLER

+	toplama operatörü	~	her bitin tersini alır
−	çıkarma operatörü	&	her biti and'lar
/	bölme operatörü	|	her biti or'lar
*	çarpma operatörü	^	her biti xor'lar
%	modunu alma operatörü	^~	her biti xnor'lar

Always Bloğu

- Prosedürel bloktur
- Blok içindeki kodlar sırayla yürütülür
- initial bloğundan farklı olarak bir kez yürütülür
- event listesine göre ne zaman çalışacağı belirlenir

```
always @ (event)
    [single statement]

always @ (event) begin
    [multiple statement]
end
```

birçok always olabilir

Seviye hassasiyeti
- Kombinasyonel lojik
bir giriş veya çıkışın değişmesinde

Kenar hassasiyeti
- flip flop
saat sinyalinin yükselen veya alçalan kenarında

Sıralı Bloklar – Paralel Bloklar

```
begin: name
    statement1;
    statement2;
    .....
    .....
end
```

```
initial begin
#10 data = 8'hfe;
fork
    #10 data = 8'h11;
    begin
        #20 data = 8'h00;
        #30 data = 8'haa;
    end
join
end
```

```
fork: name
    statement1;
    statement2;
join
```

J-K Flip Flop

J=0 K=0 durumunda Q'nun değeri ne ise aynı kalmaya devam eder
J=1 K=0 durumunda Q'nun değeri J olur yani 1
J=0 K=1 durumunda Q'nun değeri J olur yani 0
J=1 K=1 durumunda Q'nun değeri Toggle yapar

Tüm bu bahsedilen değişimler clk sinyalinin yükselen kenarında aktif olur.

D Flip Flop

Her saat sinyalinden sonra D girişi ne ise Q çıkışı da o olur.

Verilog Veri Tipleri

0 (lojik 0) veya (false)
1 (lojik 1) veya (True)
X (belirsiz bit)
Z (yüksek empedans) (bağlı değil)

bit (kullanıcı tanımlı bit)
byte 8bit signed
shortint 16bit signed
int 32 bit signed
long int 64 bit signed

reg kullanıcı tanımlı
integer 32 bit signed
real 64 bit virgüllü sayılar.

Kontrol Blokları

if-else-if

```
if (koşul) begin
    [multiple statement]
end else begin
    [multiple statement]
end
```

while

```
while(koşul) begin
    [multiple statement]
end
```

forever

```
forever begin
    [multiple statement]
end

forever
    [single statement]
```

repeat

```
repeat (number) begin
    [multiple statement]
end
```

for

```
integer i;
for(i=0; i<5; i=i+1)
begin
    [multiple statement]
end
```

Verilog Fonksiyonlar

- #, @, wait, posedge, negedge gibi zaman ifadeleri içermezler
- Fonksiyonlar diğer fonksiyonları çağırabilirler.
- En az bir girdi argumanı olmalıdır.
- <= paralel çalıştırma ve atama işlemleri yapılamaz.
- inout ve output atamaları yapılamaz
- return ifadesi içermeli ve bir değer geri döndürmelidir.

```
module tb;
  initial begin
    integer result = factorial(4);
    $display("fact(4) = %0d", result);
  end

  function automatic integer factorial(integer i);
    integer result = i
    if (i) begin
      result = i * factorial(i-1);
      $display("i=%0d result=%0d", i, result);
    end else
      result=1;

    return result;
  endfunction
endmodule
```

Verilog Task'lar

- Fonksiyonlardan daha geniş kullanım
- Zaman kontrolü, gecikme ve istenilen zamanda sonlandırma yapılabilmektedir.
- Diğer task ve fonksiyonlar çağırabilir ve çağrılabilir.
- Argument alabilir veya almayabilir
- return ifadesi yok ancak output ile çıkışa veri gönderebilir.

```
//style1
task [name];
input [portlist];
inout [portlist];
output [portlist];
begin
  statement;
  statement;
end
endtask
```

```
//empty portlist
task [name] ();
begin
  statement;
  statement;
endtask
```

```
//style2
task [name] (input [portlist], inout [portlist], output [portlist]);
begin
  statement;
  statement;
end
endtask
```

Verilog Parametreler

Parametreler sabit tanımlamalardır. Kod'ularda değiştirilmezler. defparam ile üzerine yazılabilir.

```
parameter BUS_WIDTH = 32,
          DATA_WIDTH = 64,
          FIFO_DEPTH = 512;
input wdata;
wire [DATA_WIDTH-1:0] wdata;
```

Verilog Case Blok

- case (Selection) yapısında seçim yapılacak koşul ifade bulunur.
- ifadenin değişimi sekmeleri değiştirir.
- X ve Z bit değişikliklerine cevap verebilir

```
module my_mux(input [2:0] a,b,c
              [1:0] sel,
              output reg [2:0] out);

always @(a,b,c,sel) begin
  case (sel)
    2'b00 : out = a;
    2'b01 : out = b;
    2'b10 : out = c;
    2'b11 : out = 0;
  endcase
end
endmodule
```

Zamanlama (Timing)

- #20 20 nanosaniye gecikme ekler
- Gecikme çağrıldığında Verilog hiçbir işlem yapmaz
- @(posedge a) ⟶ a değişkeninin yükselen kenarına kadar bekler

Basit RAM

- 32 bit bellek hücreli 16 hücre kapasiteli RAM verilog kodu
- cs RAM aktif
- we RAM yazmak aktif
- oe RAM okuma aktif

```
module single_port_sync_ram #( parameter ADDR_WIDTH =4,
                               parameter DATA_WIDTH =32,
                               parameter DEPTH = 16
                              )
                              (
                               input clk,
                               input [ADDR_WIDTH-1:0] addr,
                               input [DATA_WIDTH-1:0] data,
                               input cs,
                               input we,
                               input oe
                              );
    reg [DATA_WIDTH-1:0] tmp_data;
    reg [DATA_WIDTH-1:0] mem [DEPTH];

    always @ (posedge clk) begin
      if(cs & we) begin
          mem [addr] <= data;
      end
    end

    always @ (posedge clk) begin
      if (cs & !we) begin
          tmp_data <= mem[addr];
      end
    end

    assign data = cs & oe & !wr ? tmp_data : 'hz;
endmodule.
```

Kaydedicler

```
`timescale 1ns/1ps

module regfile8x16b
    ( input clk,
      input reset,
      input write
      input [2:0] wrAddr,
      input [15:0] wrData,
      input [2:0] rdAddrA,
      input [15:0] rdDataA,
      input [2:0] rdAddrB,
      input [15:0] rdDataB);

reg [15:0] regFile [0:7];

assign rdDataA = regFile [rdAddrA];
assign rdDataB = regFile [rdAddrB];

always @ (posedge clk) begin
    if (reset) begin
      regFile [0] <=0;
      regFile [1] <=0;
      regFile [2] <=0;
      regFile [3] <=0;
      regFile [4] <=0;
      regFile [5] <=0;
      regFile [6] <=0;
      regFile [7] <=0;
    end else begin
      if (write) regFile [wrAddr] <= wrData;
    end
end
endmodule
```