



# **BM222 Sayısal Tasarım (2022 Bahar Dönemi)**

Arş. Gör. Metehan GÜZEL

# Plan



- Modelsim
- Verilog HDL
- ...
- (TBC)



# Modelsim

# Simülasyon Ortamı

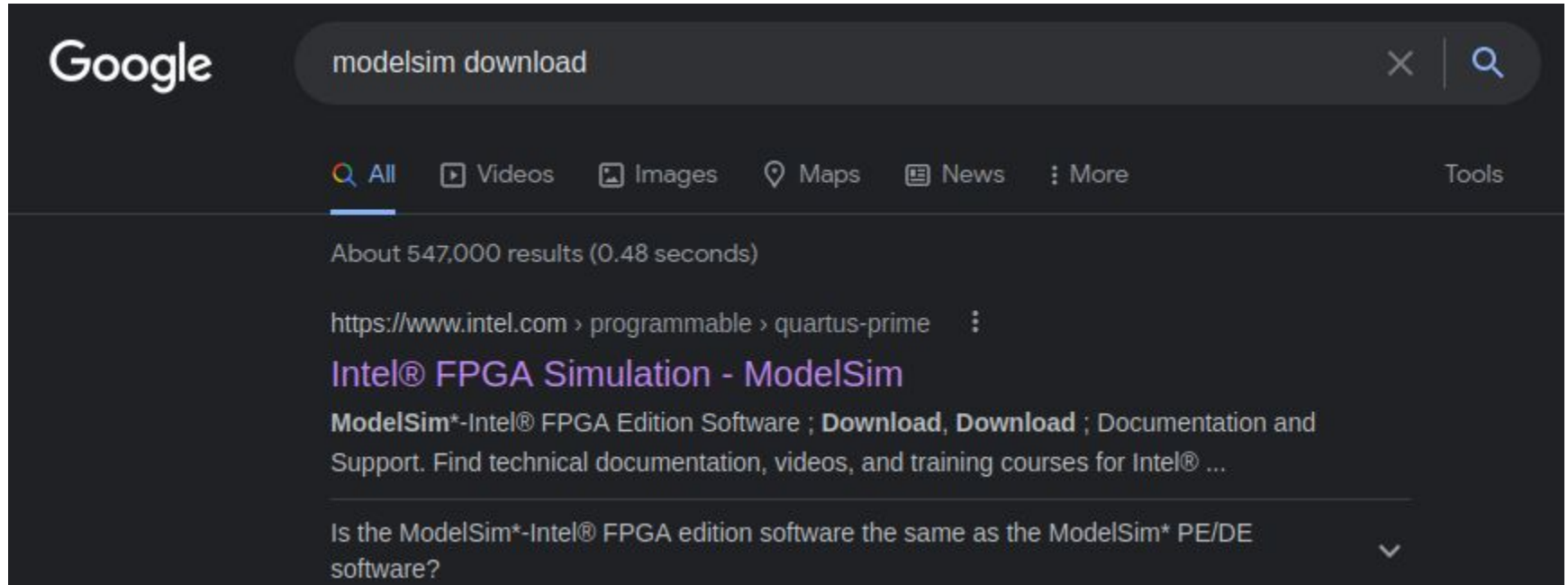
- Bu dönem derse dair simülasyonlar Modelsim 17.1'da yapacağız.

The screenshot shows the Intel FPGA Software Download Center interface. At the top, there is a navigation bar with links for PRODUCTS, SUPPORT, SOLUTIONS, DEVELOPERS, and PARTNERS. The Intel logo is on the left, and user and language icons (USA (ENGL)) are on the right. Below the navigation bar, the page title is "FPGA Software Download Center". The main content area has a dark blue background with the text "Intel® Quartus® Prime Standard Edition Design Software Version 17.1 for Linux". Below this, there is a table with three columns: ID, Date, and Version. The ID is 669392, the Date is 12/5/2017, and the Version is 17.1. At the bottom, a yellow banner contains the text: "A newer version of this software is available, which includes functional and security updates. Customers should click here to update to the latest version."

ID	Date	Version
669392	12/5/2017	17.1

# Modelsim Download

- Google'dan



# ModelSim Download





- Google'dan gelinen sitede
  - **“Download ModelSim\*-Intel® FPGA edition software”**
- Gelen sayfada
  - **“Intel® Quartus® Prime Lite Edition Design Software Version 17.1 for Windows”**
    - *İşletim sistemini kendi tercihinize göre seçebilirsiniz. Ben Windows üzerinden gideceğim.*

# ModelSim Download



Access to additional search results is restricted. Please [sign in](#) or [register](#) to view

Title	ID	Date	Version
 Intel® Quartus® Prime Lite Edition Design Software Version 17.1 for Windows	669444	01/19/18	17.1
 Intel® Quartus® Prime Standard Edition Design Software Version 17.1 for Windows	669393	12/05/17	17.1

# ModelSim Download

- Downloads-Individual Files sekmesinden ModelSim'i indiriyoruz.

## Downloads

[Multiple Download](#) [Individual Files](#) [Additional Software](#) [Updates](#)

### Intel® Quartus® Software

#### ModelSim-Intel® FPGA Edition (includes Starter Edition)

Download  
ModelSimSetup-17.1.0.590-windows.exe

Size: 1.1 GB  
SHA1: f0b4520cd766bffe4373465e1dc7b819d1176f1

#### Intel® Quartus® Prime (includes Nios® II EDS)

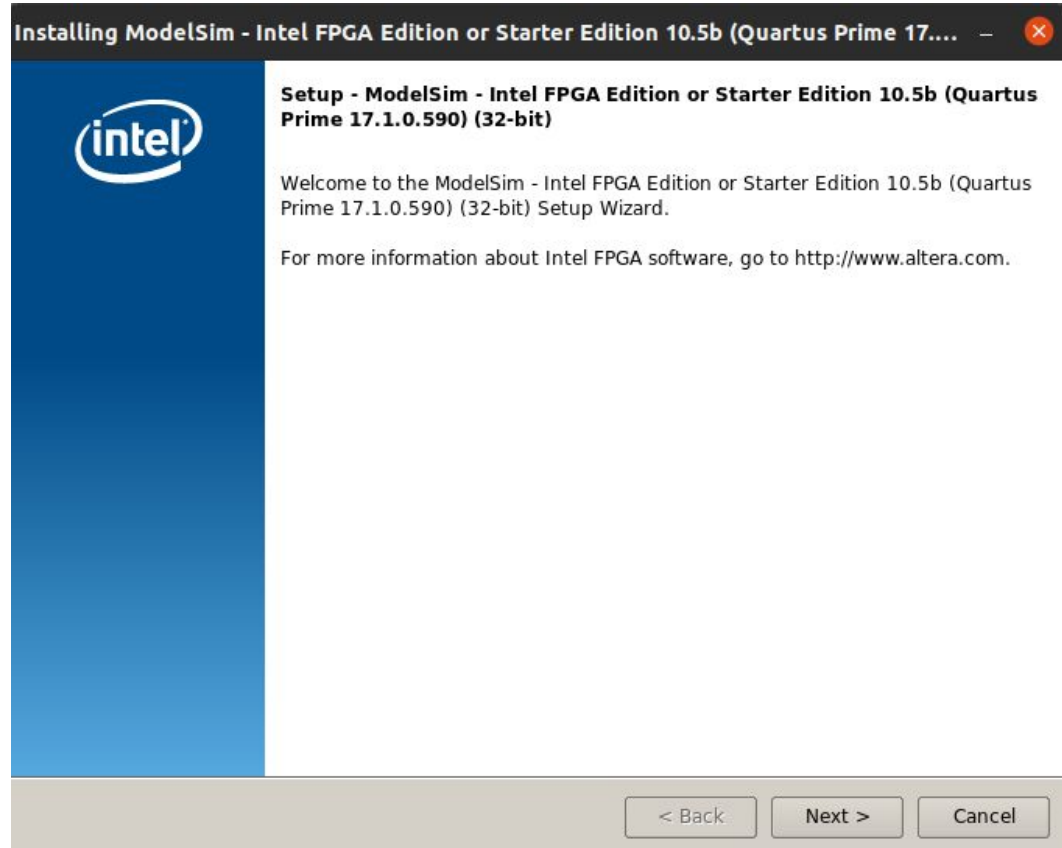
Download  
QuartusLiteSetup-17.1.0.590-windows.exe

Size: 1.7 GB  
SHA1: e6185f220de33432f352cfcb3088be7ee0971af8



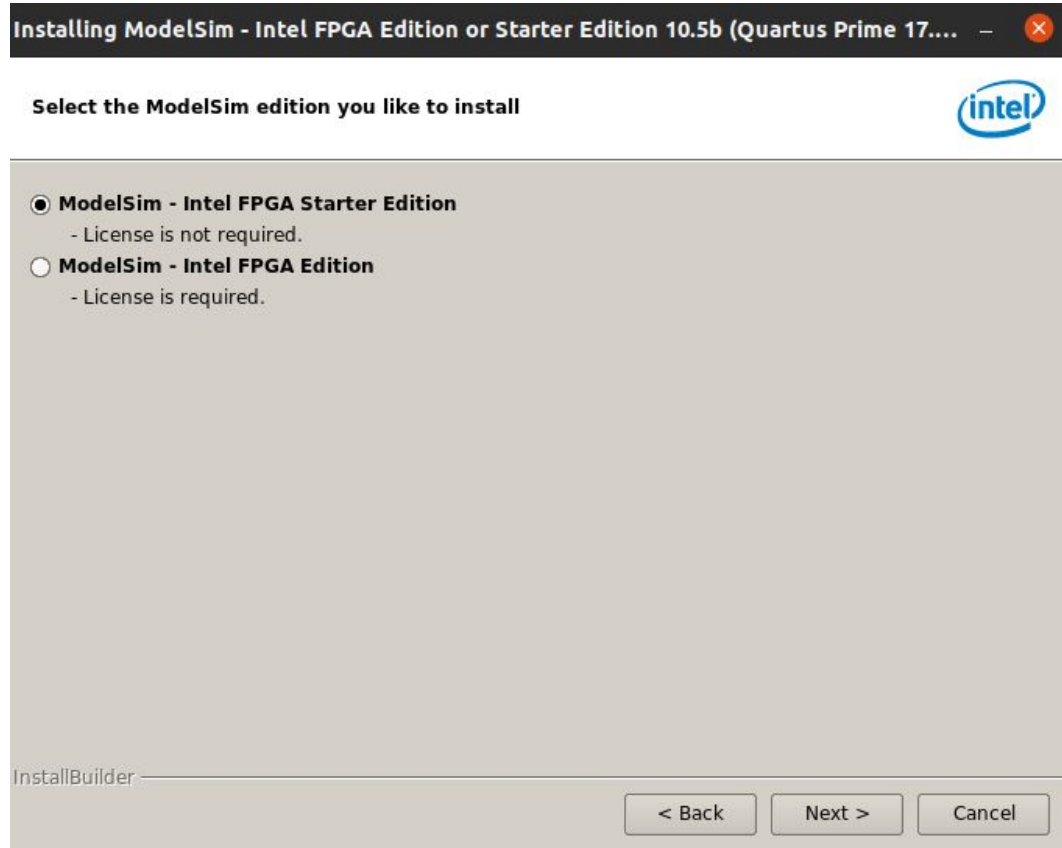
# ModelSim Kurulum - Adım 1

- Next



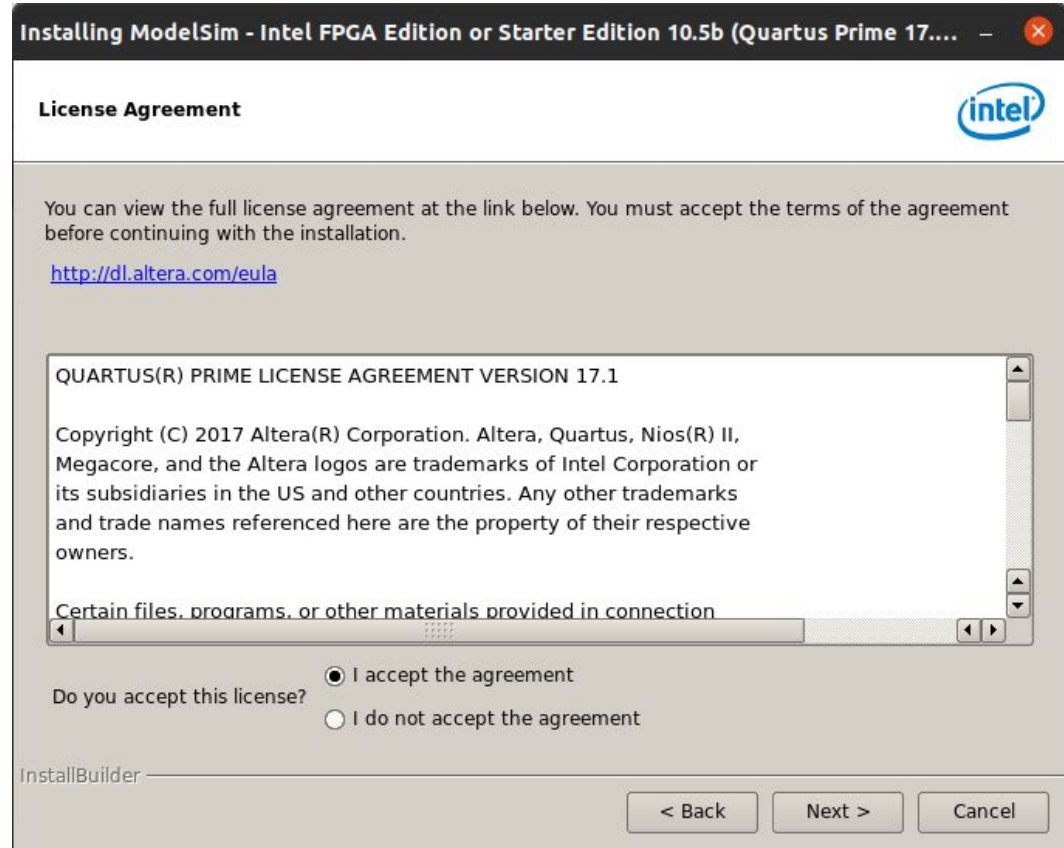
# ModelSim Kurulum – Adım 2

- Starter Edition seçili
  - Next



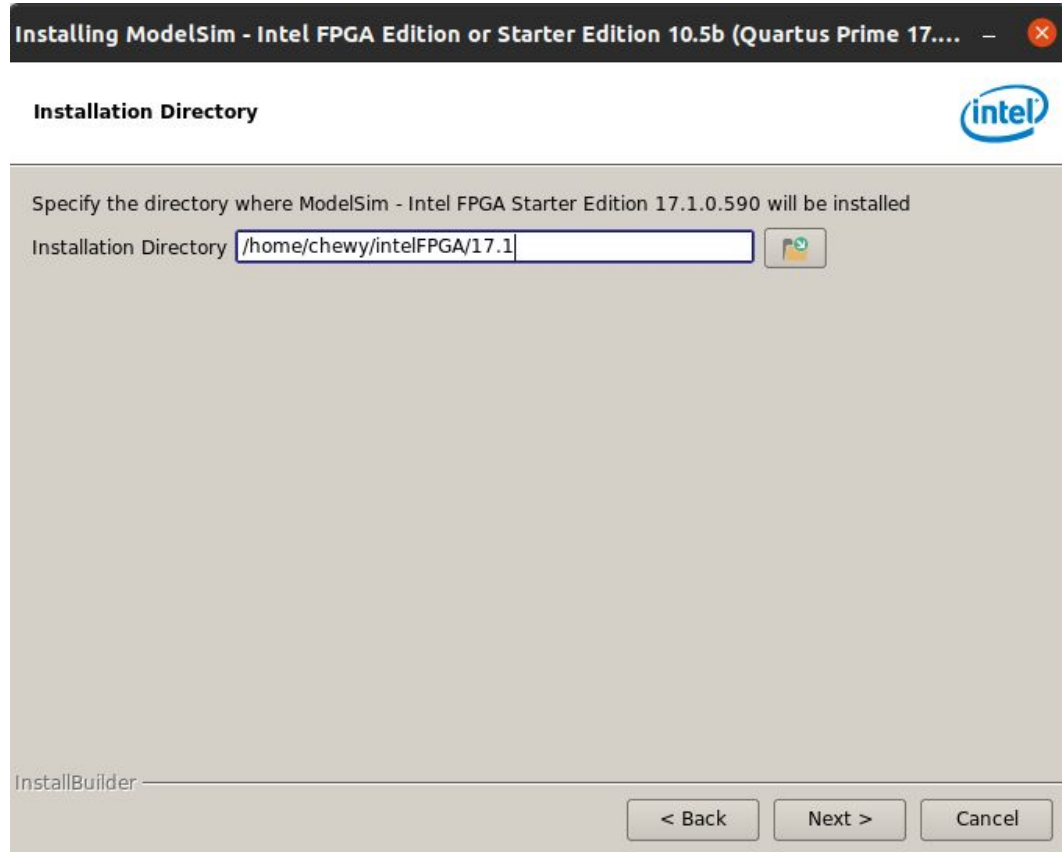
# ModelSim Kurulum - Adım 3

- I accept the agreement seçili
  - Next



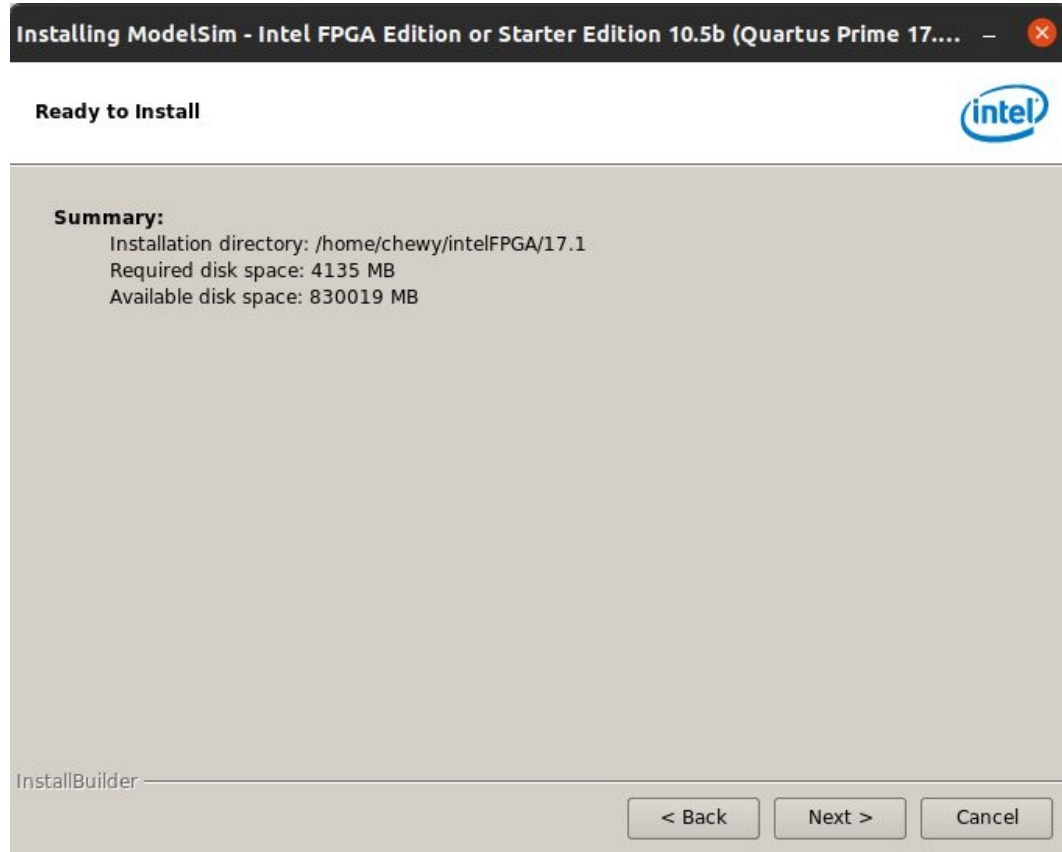
# ModelSim Kurulum - Adım 4

- Kurulum adresini belirtin
  - **Next**



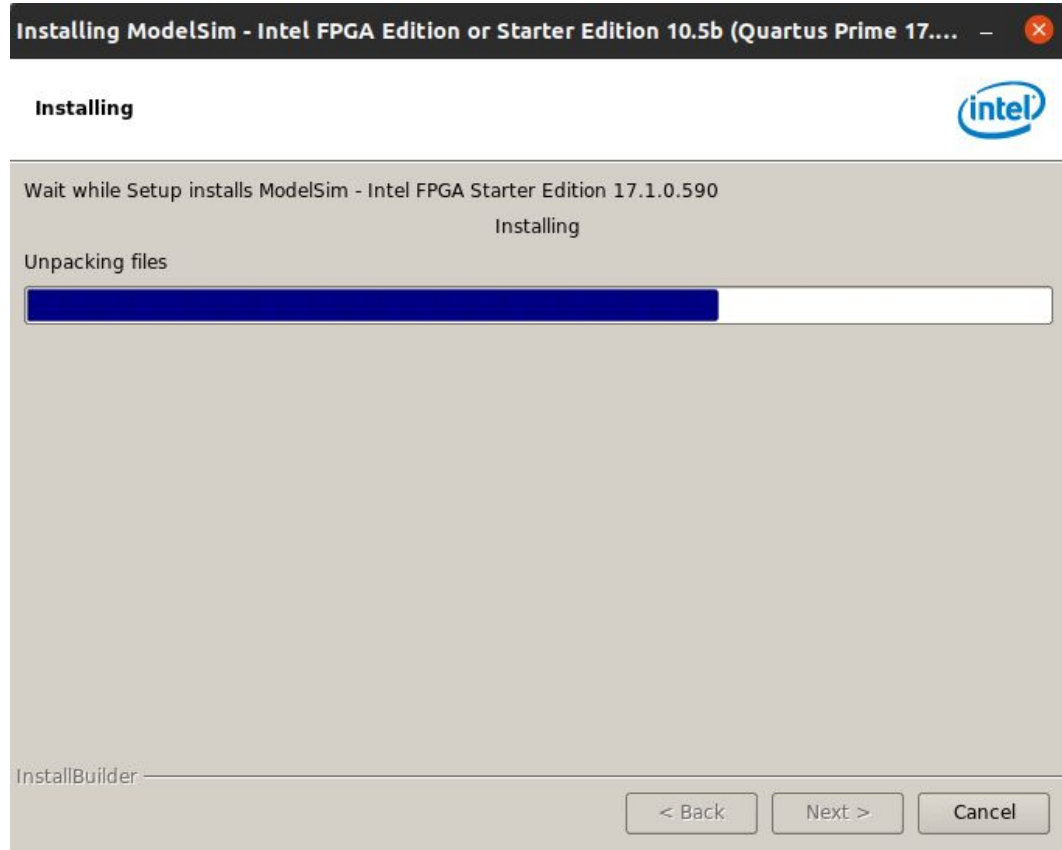
# ModelSim Kurulum – Adım 5

- Kurulum özeti
  - **Next**



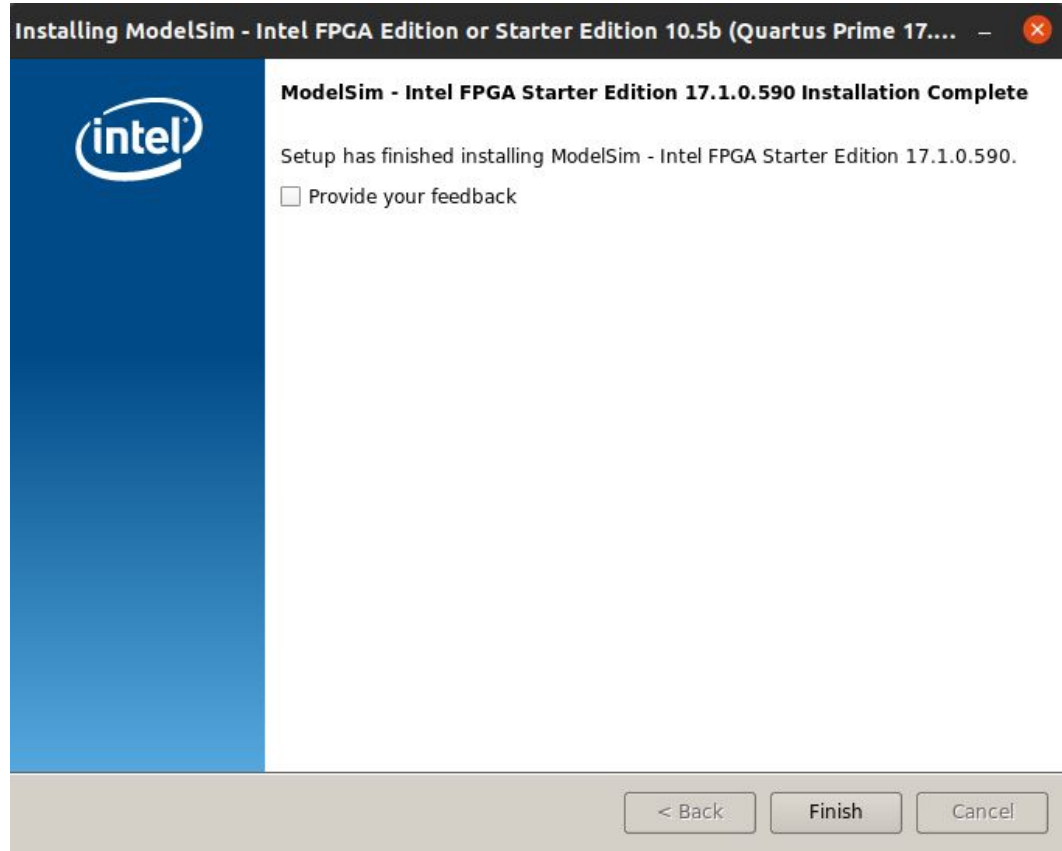
# ModelSim Kurulum - Kurulum

- Kurulum süreci
  - ...



# ModelSim Kurulum – Adım 6

- Kurulum tamamlandı.
  - 1-2 dakika sonunda kurulum biter.
  - **Finish**





# Verilog HDL



# Verilog HDL



- Plan
  - Genel Bilgiler
  - Örnek Modül
  - Açıklamalar
  - Değişken İsimlendirme
  - Değerler ve Sayılar
  - Veri Elemanları
    - Wire ve Registerlar
    - Vektörler
  - Assignment İşlemleri

# Verilog HDL



- Plan (cont)
  - Mantıksal, Bitsel, İndirgenmiş Operatörler
  - Kaydırma Operatörleri
  - Birleştirme Operatörleri
  - İlişki Operatörleri
  - Eşitlik Operatörleri
  - Şartlar
  - Aritmetik Operatörler
  - UDP'ler
  - Modellemeler (Tasarım Türleri)

# Verilog HDL



- Plan (cont)
  - If-Else
  - Case
  - Döngüler (for, forever, while ve repeat)
  - Modellemeler (Tasarım Türleri)
    - Yapısal Modelleme (Structural, Gate-Level)
    - Akışsal Modelleme (Data-Flow)
    - Davranışsal Modelleme (Behavioral)
  - Olaylar ve Hassasiyet Listeleri
  - Gecikmeler, Clock kullanımı ve Edge'ler

# Verilog HDL – Genel Bilgiler



- HDL -> Hardware Description Language
- Verilog HDL
  - basit kapıların, devrelerden karmaşık mikroişlemcilerle kadar bilgisayar elemanlarının tasarlanmasında ve simüle edilmesinde kullanılır. (Sayısal)
  - Aynı zamanda, yakın gerçek zamanlı sinyal işleme işlemlerinde kullanılır. (Analog)
- Bu ders (Sayısal Tasarım) kapsamında bilgisayar bileşenlerinin tasarlanması, kodlanması ve simüle edilmesinde kullanılacak.

# Verilog HDL – Örnek Modül

- Verilog'da modüller kullanılır.
  - Keywords
    - **module**
    - **endmodule**
    - **input**
    - **output**
    - **inout**
  - Portlar
  - Kapılar

Ln#	
1	<code>module demo(a,b,c,d,e);</code>
2	<code>input a,b;</code>
3	<code>output c,d,e;</code>
4	<code>or (c,a,b);</code>
5	<code>and (d,a,b);</code>
6	<code>xor (e,a,b);</code>
7	<code>endmodule</code>
8	

# Verilog HDL – Açıklamalar

- Açıklamalar (Comments)
  - Tek Satır Açıklama
    - //
  - Çok Satır Açıklama
    - /\* \*/

Ln#	
1	module demo(a,b,c,d,e);
2	input a,b;
3	output c,d,e;
4	or (c,a,b);
5	and (d,a,b);
6	xor (e,a,b);
7	// Aciklama Satiri
8	/* Çok Satirli
9	Aciklama
10	*/
11	endmodule

# Verilog HDL – Değişken İsimlendirme



- Dikkat edilecek noktalar
  - {[A-Z], [a-z], [0-9], \_, \$} karakterleri kullanılabilir.
  - \$ veya [0-9] karakterleri ile değişken ismi başlayamaz.
  - Büyük küçük harf duyarlılığı vardır.

# Verilog HDL – Değerler ve Sayılar



- Değerler
  - 0 -> Düşük, False, elektrik yok durumu
  - 1 -> Yüksek, True, elektrik var durumu
  - X -> Bilinmeyen durum
  - Z -> Okunamayan durum



# Verilog HDL – Değerler ve Sayılar

- Sayı Tanımlanması
  - <boyut>'<taban> <değer>
- Örnek
  - 1'b1

Ln#	
1	<code>module test_bench();</code>
2	
3	<code>reg a,b;</code>
4	<code>wire c,d,e;</code>
5	
6	<code>demo x(a,b,c,d,e);</code>
7	
8	<code>initial begin</code>
9	
10	<code>    a = 1'b1;</code>
11	<code>    b = 1'b1;</code>
12	

# Verilog HDL – Değerler ve Sayılar



- **Boyut** ve **Değer** sayısal tanımlanan değerlerdir.
- Taban
  - İkili Taban, Binary → B, b
  - Sekizlik Taban, Octal → O, o
  - Onluk Taban, Decimal → D, d
  - Onaltılık Taban, Hexadecimal → H, h

# Verilog HDL – Değerler ve Sayılar



- Örnek, aşağıda verilen tanımlamaların hepsi aynıdır.
  - `4'b 1011`
  - `4'b 10_11`
  - `4'o 13`
  - `4'd 11`
  - `4'h b`

# Verilog HDL – Wire ve Registerlar



- Wire ve Register tekil veri depolaması(!) için kullanılır.
  - Wire girişlerin fonksiyonudur.
    - Dolayısıyla değeri girişlere göre anlık değişir. Kablo gibi düşünebilirsiniz. (Net)
  - Register veri saklama birimidir.
    - Yüklenen değer üstüne veri yazılmadıkça değişmeden kalır. (Variable)
      - Girişlerin değişimlerinden bağımsız olarak.

# Verilog HDL – Wire ve Registerlar

- Wire ve Register atamasına dair örnek kod.
  - Registerlara atama
    - =
  - Wire'lara atama
    - assign

```
Ln# |  
1  | module wl_reg(a,w,r,clock);  
2  |  
3  |     input clock;  
4  |     output reg a;  
5  |     output reg r;  
6  |     output wire w;  
7  |  
8  |     assign w = a || 1'b0;  
9  |  
10 |     initial  
11 |         begin  
12 |             a = 1'b0;  
13 |             r = a || 1'b0;  
14 |         end  
15 |  
16 |     always @ (clock)  
17 |         begin  
18 |             a = ~a;  
19 |         end  
20 |  
21 | endmodule  
22 |
```

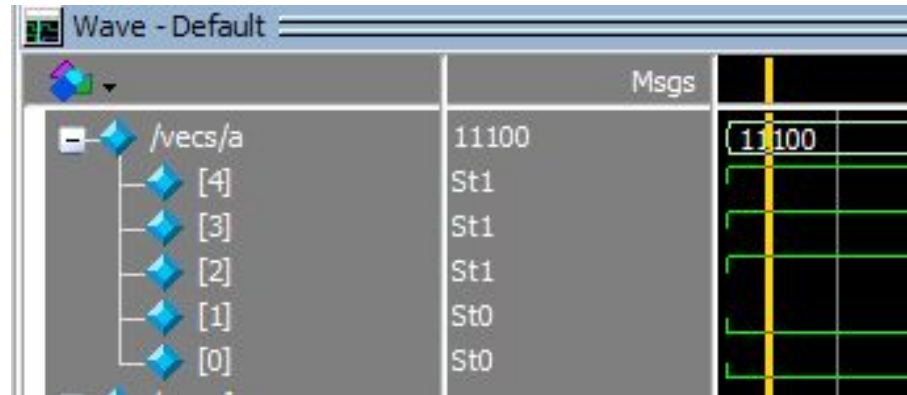
# Verilog HDL – Vektörler

- Register ve Wire serileridir.
  - Tanımlamalarda **range** önemlidir.
    - wire [4:0] a;
    - reg [1:5] b;
    - reg [1:0] c;

```
Ln# |  
1  | module vecs();  
2  |  
3  | wire [4:0] a;  
4  | reg [1:5] b;  
5  | reg [1:0] c;  
6  |  
7  | assign a = 5'b11100;  
8  |  
9  | initial  
10 | | begin  
11 | |     #100  
12 | |     c = a;  
13 | |  
14 | |     #100  
15 | |     c = a[2:1];  
16 | |  
17 | |     #100  
18 | |     b = a;  
19 | | end  
20 |  
21 |  
22 | endmodule  
23 |
```

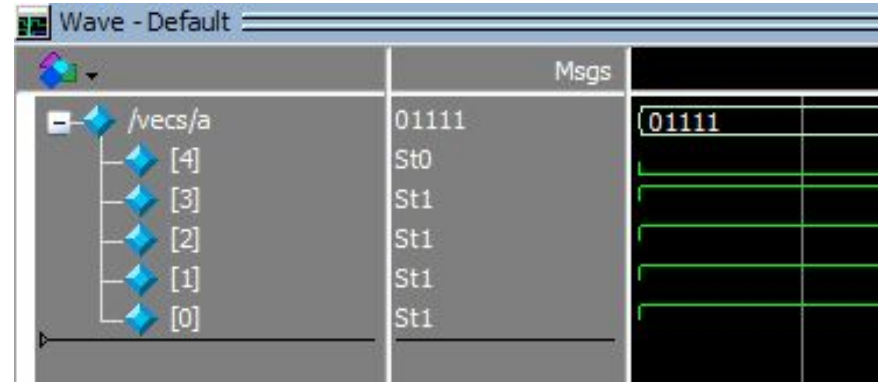
# Verilog HDL – Vektörler

- Operasyonlar
  - assign a = 5'b11100;
    - a[4] <- 1
    - a[3] <- 1
    - a[2] <- 1
    - a[1] <- 0
    - a[0] <- 0



# Verilog HDL – Vektörler

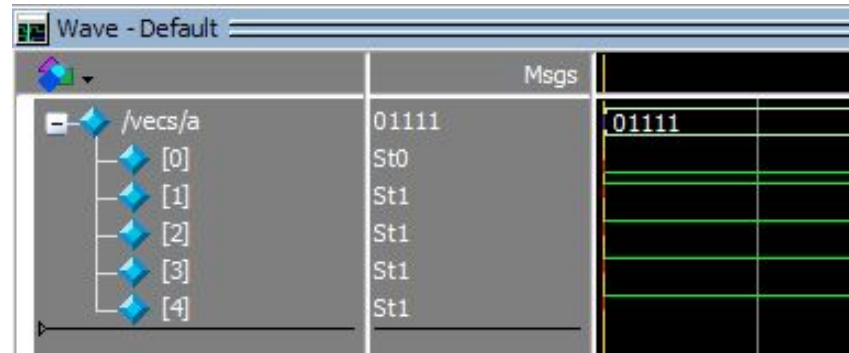
- Operasyonlar
  - `assign a = 4'b1111;`
    - `a[3] <- 1`
    - `a[2] <- 1`
    - `a[1] <- 1`
    - `a[0] <- 1`
    - `a[4]`'a 0 değeri yüklenir.  
Default





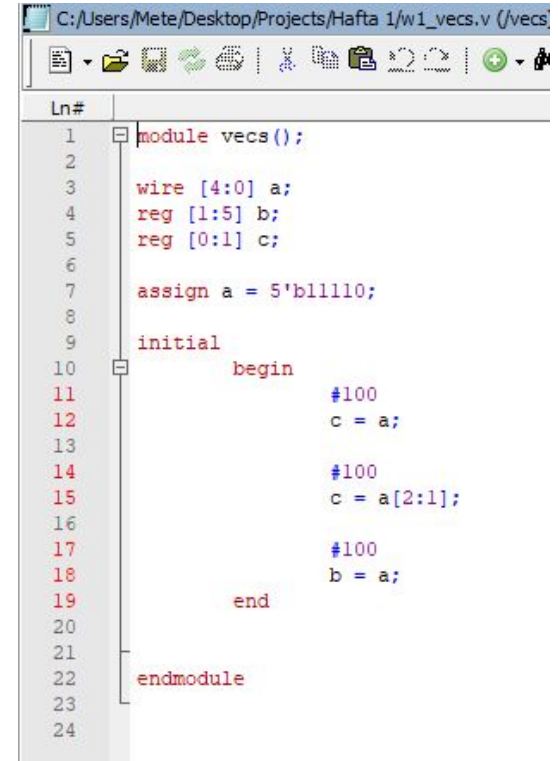
# Verilog HDL – Vektörler

- Operasyonlar
  - `wire [0:4] a;`
  - `assign a = 4'b1111;`
    - `a[4] <- 1`
    - `a[3] <- 1`
    - `a[2] <- 1`
    - `a[1] <- 1`
    - `a[0]`'a 0 değeri yüklenir.  
Default



# Verilog HDL – Vektörler

- Operasyonlar
  - $c = a$ ; (c ve a aynı yönde)
    - $c[1] \leftarrow a[1]$
    - $c[0] \leftarrow a[0]$
  - $c = a$ ; (c ve a zıt yönde)
    - $c[1] \leftarrow a[0]$
    - $c[0] \leftarrow a[1]$



```
1 module vecs();
2
3 wire [4:0] a;
4 reg [1:5] b;
5 reg [0:1] c;
6
7 assign a = 5'b11110;
8
9 initial
10 begin
11     #100
12     c = a;
13
14     #100
15     c = a[2:1];
16
17     #100
18     b = a;
19 end
20
21
22 endmodule
23
24
```

# Verilog HDL – Vektörler

- Operasyonlar
  - `c = a;` (c ve a aynı yönde)
    - `c[1] <- a[1]`
    - `c[0] <- a[0]`
  - `c = a;` (c ve a zıt yönde)
    - `c[1] <- a[0]`
    - `c[0] <- a[1]`



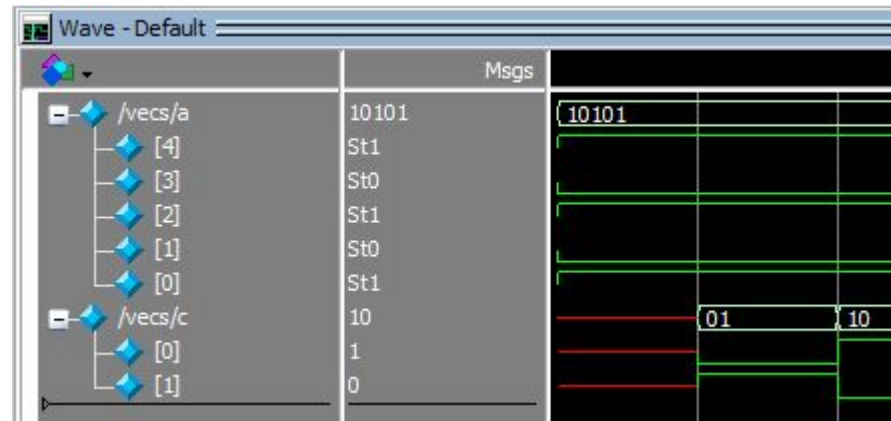
# Verilog HDL – Vektörler

- Operasyonlar
  - wire [4:0] a;
  - reg [1:0] c;
  - c = a[2:1]; (c ve a aynı yönde)

Ln#	
1	module vecs();
2	
3	wire [4:0] a;
4	reg [1:5] b;
5	reg [0:1] c;
6	
7	assign a = 5'b10101;
8	
9	initial
10	begin
11	#100
12	c = a;
13	
14	#100
15	c = a[2:1];
16	
17	#100
18	b = a;
19	end
20	
21	
22	endmodule
23	
24	

# Verilog HDL – Vektörler

- Operasyonlar
  - `c = a[2:1];`
    - `c[1] <- a[2]`
    - `c[0] <- a[1]`



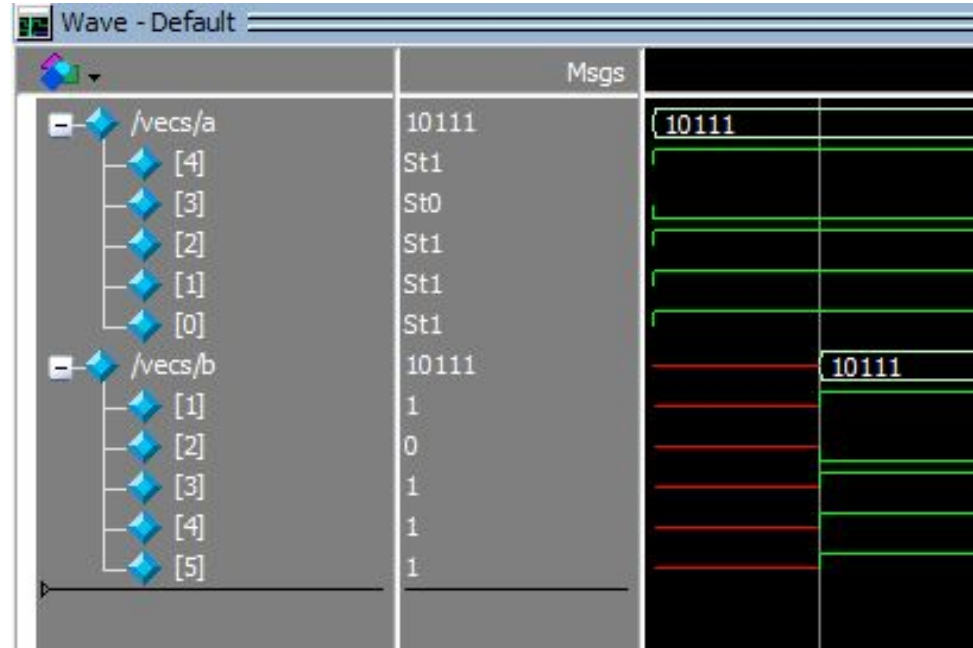
# Verilog HDL – Vektörler

- Operasyonlar
  - wire [4:0] a;
  - reg [1:5] b;
  - b = a

```
Ln# |  
1  | module vecs();  
2  |  
3  |     wire [4:0] a;  
4  |     reg [1:5] b;  
5  |     reg [0:1] c;  
6  |  
7  |     assign a = 5'b10111;  
8  |  
9  |     initial  
10 |         begin  
11 |             //#100  
12 |             //c = a;  
13 |  
14 |             //#100  
15 |             //c = a[2:1];  
16 |  
17 |             #100  
18 |             b = a;  
19 |         end  
20 |  
21 |  
22 |     endmodule
```

# Verilog HDL – Vektörler

- Operasyonlar
  - $b = a$ 
    - $b[1] \leftarrow a[4]$
    - $b[2] \leftarrow a[3]$
    - $b[3] \leftarrow a[2]$
    - $b[4] \leftarrow a[1]$
    - $b[5] \leftarrow a[0]$



# Verilog HDL – Assignment İşlemleri

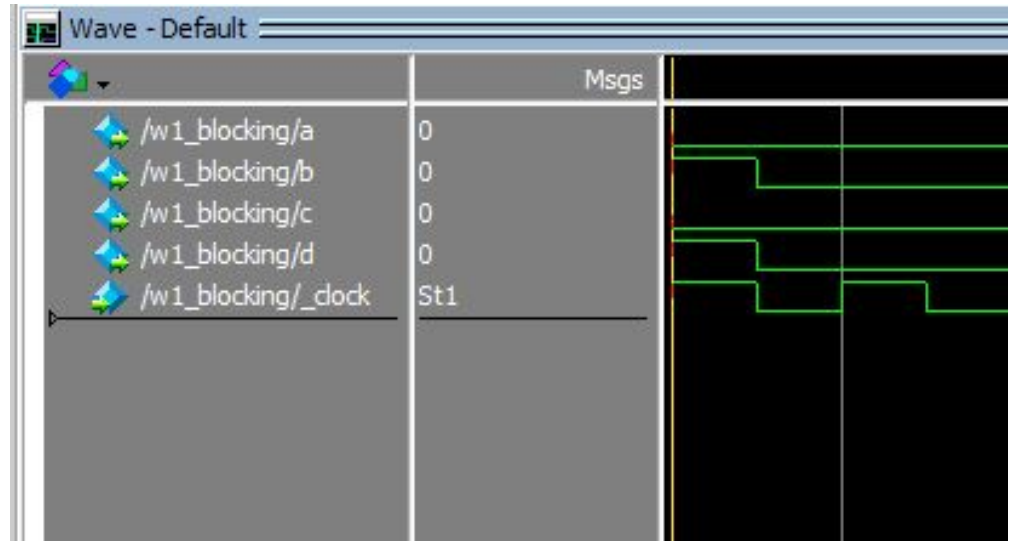


- **[Kritik]** İki tür assignment kullanılır.
  - Blocking Assignment
    - = operatörü kullanılır.
    - Sıralı çalışır.
  - Non-Blocking Assignment
    - <= operatörü kullanılır.
    - Paralel çalışır.



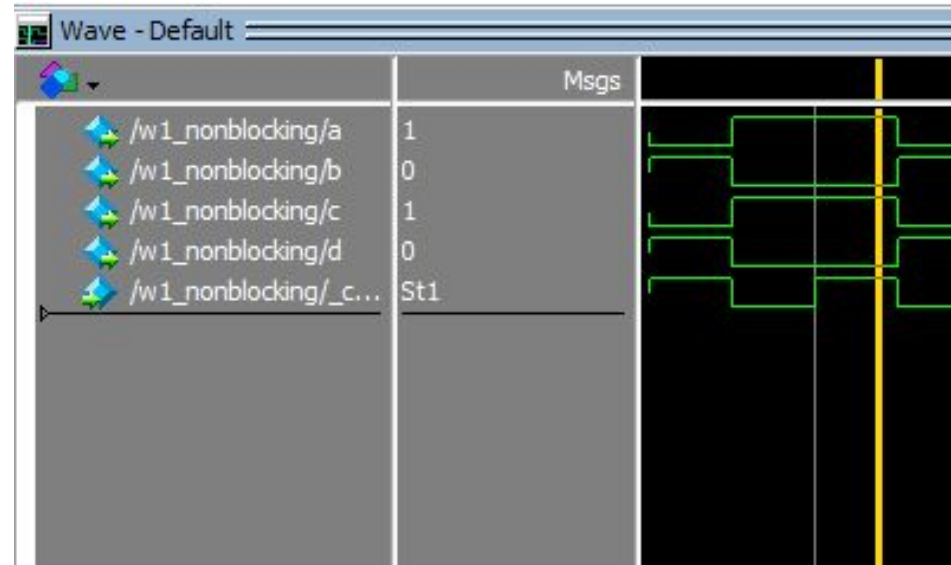
# Verilog HDL – Blocking Assignment

```
Ln# | 1 | module w1_blocking(a,b,c,d,_clock);  
    | 2 |  
    | 3 | output reg a,b,c,d;  
    | 4 | input _clock;  
    | 5 |  
    | 6 | initial  
    | 7 | begin  
    | 8 |     a = 1'b0;  
    | 9 |     b = 1'b1;  
   10 |     c = 1'b0;  
   11 |     d = 1'b1;  
   12 | end  
   13 |  
   14 | always @ (negedge _clock)  
   15 | begin  
   16 |     b = a;  
   17 |     c = b;  
   18 |     d = c;  
   19 |     a = d;  
   20 | end  
   21 |  
   22 | endmodule  
   23 |
```



# Verilog HDL – Non-Blocking Assignment

```
Ln# |  
1  | module w1_nonblocking(a,b,c,d,_clock);  
2  |  
3  |     output reg a,b,c,d;  
4  |     input _clock;  
5  |  
6  |     initial  
7  |     begin  
8  |         a = 1'b0;  
9  |         b = 1'b1;  
10 |         c = 1'b0;  
11 |         d = 1'b1;  
12 |     end  
13 |  
14 |     always @ (negedge _clock)  
15 |     begin  
16 |         b <= a;  
17 |         c <= b;  
18 |         d <= c;  
19 |         a <= d;  
20 |     end  
21 |  
22 | endmodule  
23 |
```



# Verilog HDL – Mantıksal Operatörler



- Mantıksal Operatörler
  - Operatörler
    - `&&` -> AND
    - `||` -> OR
    - `!` -> NOT
  - Operandlar ve Sonuçlar 1 Bit

# Verilog HDL – Bitisel Operatörler



- Bitisel Operatörler
  - Operatörler
    - `&` -> Bitisel AND
    - `|` -> Bitisel OR
    - `~` -> Bitisel NOT
    - `^` -> Bitisel XOR
  - Bit bit işlem yapar

# Verilog HDL – İndirgenmiş Operatörler



- Bitisel Operatörler
  - Operatörler
    - `&` -> İndirgenmiş AND
    - `|` -> İndirgenmiş OR
    - `^` -> İndirgenmiş XOR
  - Çok bitlik operandı tek bitlik sonuca çevirir.

# Verilog HDL – Birleştirme Operatörü



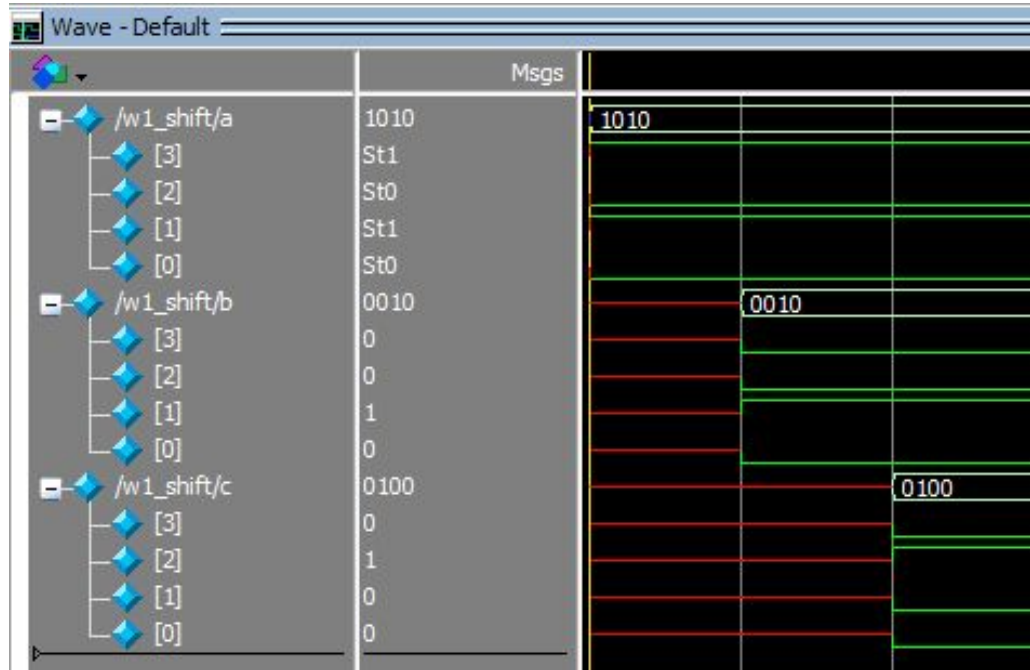
- {} Birleştirme için kullanılır.
- Assume,
  - `a = 1'b1;`
  - `b = 2'b01;`
  - `c = 3'b101;`
  - `x = {a,b,c};` // `x = 1_01_101`
  - `y = {a,c,2'b11} // y = 1_101_11`
  - `z = {{2{c},b,a} // z = 101_101_01_1`

# Verilog HDL – Kaydırma Operatörü



- Shift Operatörleri
  - Sağa kaydır >>
  - Sola kaydır <<
- Boşlukla sıfırla doldurulur.
- Örnek
  - `a = 4'b1010;`
    - `b = a >> 2;      // b = 0010`
    - `c = a << 1;      // c = 0100`

# Verilog HDL – Kaydırma Operatörü





# Verilog HDL – İlişki Operatörleri

---

- İlişki Operatörleri
  - `>`     $\rightarrow$  Büyüktür
  - `<`     $\rightarrow$  Küçüktür
  - `>=`  $\rightarrow$  Büyük eşittir
  - `<=`  $\rightarrow$  Küçük eşittir
- Sonuç bir bitliktir.
  - `1 > 0`         $\rightarrow$  1
  - `1 < 0`         $\rightarrow$  0
  - `'bx > 0`        $\rightarrow$  x
  - `'bx <= 0`       $\rightarrow$  ~x

# Verilog HDL – Eşitlik Operatörleri



- İlişki Operatörleri
  - == -> Mantıksal (Logic) Eşitlik
  - != -> Mantıksal Eşitsizlik
  - === -> Durum (State) Eşitliği
  - !== -> Durum (State) Eşitsizliği
- Sonuçlar
  - Mantıksal eşitlik -> 0,1,x döndürür
  - Durum eşitliği -> 0,1 döndürür

# Verilog HDL – Eşitlik Operatörleri



- Örnek
  - `4'b 1z0x == 4'b 1z0x -> x`
  - `4'b 1z0x != 4'b 1z0x -> x`
  - `4'b 1z0x === 4'b 1z0x -> 1`
  - `4'b 1z0x !== 4'b 1z0x -> 0`

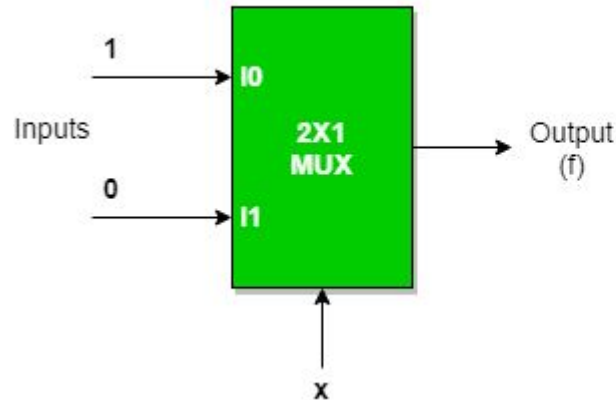
# Verilog HDL – Şart Operatörü



- `condition_expr ? true_expr : false_expr`
- Örnek I
  - Tek giriş (in) ve tek çıkış (out) olan bir devre düşünelim.
  - `out = (in) 0 : 1`

# Verilog HDL – Şart Operatörü

- Örnek II, 2x1 Multiplexer
  - $f = x ? I0 : I1$



Truth Table

x	f
0	1
1	0

# Verilog HDL – Aritmetik Operatörler



- Operatörler
  - Toplama      -> +
  - Çıkarma      -> -
  - Çarpma      -> \*
  - Bölme      -> /
  - Mod      -> %
- Herhangi bir operand **x** olursa sonuç **x** olur.

# Verilog HDL – User Defined Primitives

- UDP
- Düşük bellek gerektirir ve hızlıdır.

```
primitive mux_prim (mux_out, select, a, b)
    output mux_out;
    input select, a, b;
    table
    //      select      a      b      :      mux_out
        0      0      0      :      0;
        0      0      1      :      0;
        0      0      x      :      0;
        0      1      0      :      1;
        0      1      1      :      1;
        0      1      x      :      1;
        ...
    endtable
endprimitive
```

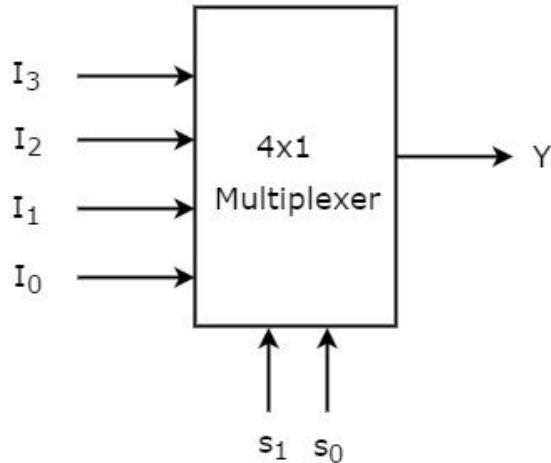
# Verilog HDL - If/Else



- Klasik if/else kullanımı
  - if (expr1)
    - true\_res\_1;
  - else if(expr2)
    - true\_res\_2;
  - else
    - def\_res



# Verilog HDL - If/Else



```
module mux4_1(out, in, sel);
    output out;
    input [3:0] in;
    input [1:0] sel;
```

```
    reg out;
    wire [3:0] in;
    wire [1:0] sel;
```

```
    always @(in or sel)
        if (sel == 0)
            out = in[0];
        else if (sel == 1)
            out = in[1];
        else if (sel == 2)
            out = in[2];
        else
            out = in[3];
```

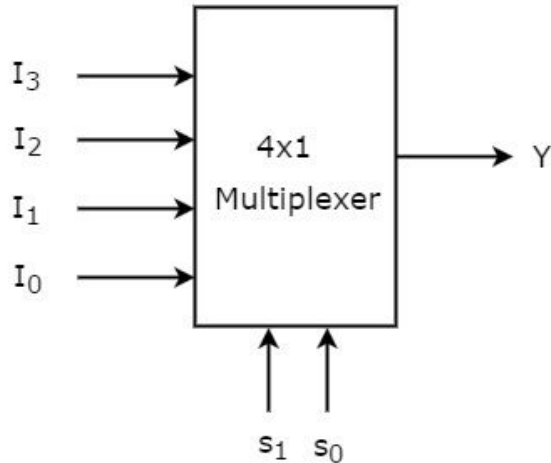
```
endmodule
```

# Verilog HDL – Case



- `case (exp)`
  - `item_1 : op_1;`
  - `item_2 : op_2;`
  - `item_3 : op_3;`
  - `item_4 : op_4;`
  - `...`
- `endcase`

# Verilog HDL – Case



```
module mux4_1(out, in, sel);
output out;
input [3:0] in;
input [1:0] sel;

reg out;
wire [3:0] in;
wire [1:0] sel;

always @(in or sel)
    case (sel)
        0: out = in[0];
        1: out = in[1];
        2: out = in[2];
        3: out = in[3];
    endcase
endmodule
```

# Verilog HDL – For



- `for (cond_init; cond; op_step)`
  - `statement`

```
module count(Y, start);
    output [3:0] Y;
    input start;

    reg [3:0] Y;
    wire start;
    integer i;

    initial
        Y = 0;

    always @(posedge start)
        for (i = 0; i < 3; i = i + 1)
            #10 Y = Y + 1;

endmodule
```

# Verilog HDL – While



- while (cond)
  - statement

```
module count(Y, start);  
    output [3:0] Y;  
    input start;  
  
    reg [3:0] Y;  
    wire start;  
    integer i;  
  
    initial  
        Y = 0;  
  
    always @(posedge start) begin  
        i = 0;  
        while (i < 3) begin  
            #10 Y = Y + 1;  
            i = i + 1;  
        end  
    end  
endmodule
```

# Verilog HDL – Repeat

- repeat (times) statement;

```
module count(Y, start);  
    output [3:0] Y;  
    input start;  
  
    reg [3:0] Y;  
    wire start;  
  
    initial  
        Y = 0;  
  
    always @(posedge start)  
        repeat (4) #10 Y = Y + 1;  
endmodule
```

# Verilog HDL – Forever

- forever statement;

```
module test;

reg clk;

initial begin
    clk = 0;
    forever #10 clk = ~clk;
end

other_module1 o1(clk, ..);
other_module2 o2(.., clk, ..);

endmodule
```

# Verilog HDL – Modellemeler



- Verilog dilince modüller 3 farklı tarzda modellenenebilir.
  - Yapısal (Structural)
  - Akışsal (Data Flow)
  - Davranışsal (Behavioral)



# Verilog HDL – Full Adder

- Full Adder 3 Biti toplar.

- Girişler

- A

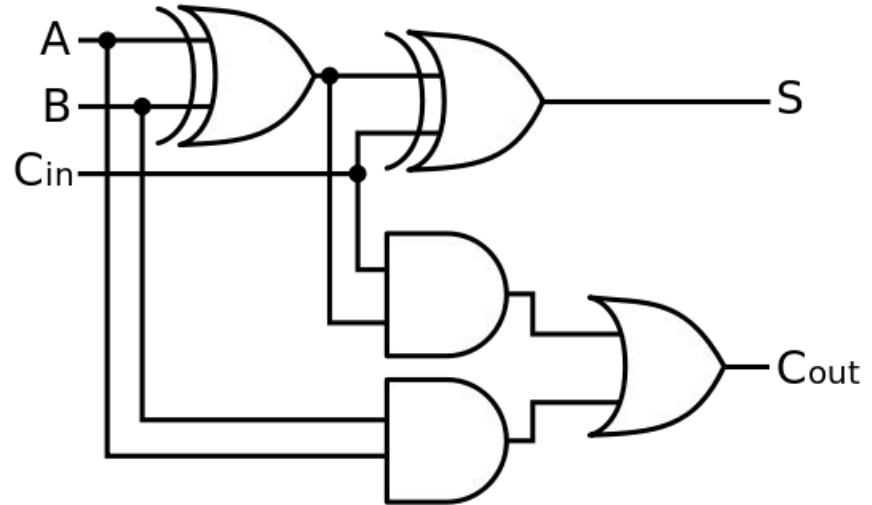
- B

- Cin

- Çıkışlar

- S

- Cout



# Verilog HDL – Modellemeler, Yapısal

- Primitif kapılar kullanılarak modelleme yapılır.

```
module fulladder_structural(a,b,c_in, s, c_out);  
    input a,b,c_in;  
    output s, c_out;  
  
    wire x1, x2, x3;  
    xor (x1, a, b);  
    and (x2, a, b);  
    and (x3, x1, c_in);  
    or (c_out, x2, x3);  
    xor (s, x1, c_in);  
endmodule
```

# Verilog HDL – Modellemeler, Akışsal

- Akışsal modellemede çıkış sinyalleri giriş sinyallerinin bir fonksiyonu olarak ifade edilir ve assign ifadesi, = operatörü kullanılır.

```
module fulladder_dataflow(a,b,c_in, s, c_out);  
    input a,b,c_in;  
    output s, c_out;  
    assign {c_out, s} = a + b + c_in;  
endmodule
```

# Verilog HDL – Modellemeler, Davranışsal



- Davranışsal modellemede always ve initial blokları kullanılarak modül modellenir.
  - Event bazlı işlemler için gereklidir.
  - Initial bloğu bir kez çalışır.
  - Always bloğu, her event gerçekleştiğinde çalışır.

# Verilog HDL – Modellemeler, Davranışsal

- Davranışsal modelleme

```
module changing_signal(signal, clock);  
    output reg signal, clock;  
  
    initial  
    begin  
        signal <= 0;  
    end  
  
    always @ (clock)  
    begin  
        signal = ~signal;  
    end  
  
endmodule
```

# Verilog HDL – Modellemeler, Davranışsal

- Always Bloğu
  - Sensivity List, bloğun içerisindeki kodun çalışması için trigger görevini görür.
  - Elements, always bloğu çalıştırıldığında uygulanacak işlemlerdir.
  - Always bloğu içerisinde reg'lere atama yapılabilir.

```
always @ ( ... sensitivity list ... )  
begin  
    ... elements ...  
end
```

# Verilog HDL – Modellemeler, Davranışsal



- Sensivity List
  - `<none> clock`
    - Sinyaldeki her değişimde trigger oluşturur.
  - `posedge clock`
    - 0'dan 1'e geçişte trigger oluşturur.
  - `negedge clock`
    - 1'den 0'a geçişte trigger oluşturur.

# Verilog HDL – Modellemeler, Davranışsal

- Blocking Assignment, = operatörü, sıralı çalışır.

```
module assignment_blocking(a,b,c,d,_clock);
output reg a,b,c,d,_clock;

initial
begin
    a = 0;
    b = 1;
    c = 0;
    d = 1;
end

always @ (_clock)
begin
    b = a;
    c = b;
    d = c;
    a = d;
end

endmodule
```



# Verilog HDL – Modellerler, Davranışsal

- Non-Blocking Assignment, <= operatörü, paralel çalışır.

```
module assignment_non_blocking(a,b,c,d,_clock);  
output reg a,b,c,d,_clock;  
  
initial  
begin  
    a = 0;  
    b = 1;  
    c = 0;  
    d = 1;  
end  
  
always @ (_clock)  
begin  
    b <= a;  
    c <= b;  
    d <= c;  
    a <= d;  
end  
  
endmodule
```

# Verilog HDL – Modellemeler, Davranışsal

- Full Adder, davranışsal modelleme.

```
module fulladder_behavioral_0(a,b,c_in, s, c_out);  
    input wire a,b,c_in;  
    output reg s, c_out;  
  
    always @(a or b or c_in)  
        begin  
            {c_out, s} = a + b + c_in;  
        end  
  
endmodule
```

# Verilog HDL – Modellemeler, Davranışsal



- Davranışsal modellemede üst düzey kodlama fonksiyonları kullanılabilir.
  - Case
  - If/Else
  - Döngüler

# Verilog HDL – Modellerler, Davranışsal

- Full Adder
  - Davranışsal
    - Case

```
module fulladder_behavioral_1(a,b,c_in, s, c_out);  
input wire a,b,c_in;  
output reg s, c_out;  
  
always @(a or b or c_in)  
begin  
    case ({a,b,c_in})  
        3'b000: begin s = 0; c_out = 0; end  
        3'b001: begin s = 1; c_out = 0; end  
        3'b010: begin s = 1; c_out = 0; end  
        3'b011: begin s = 0; c_out = 1; end  
        3'b100: begin s = 1; c_out = 0; end  
        3'b101: begin s = 0; c_out = 1; end  
        3'b110: begin s = 0; c_out = 1; end  
        3'b111: begin s = 1; c_out = 1; end  
    endcase  
end  
  
endmodule
```

# Verilog HDL – Modellemeler, Davranışsal

- Full Adder, Davranışsal, If/Else

```
module fulladder_behavioral_2(a,b,c_in, s, c_out);
input wire a,b,c_in;
output reg s, c_out;
always @(a or b or c_in)
begin
    if(a==0 && b==0 && c_in==0)
    begin
        s=0;
        c_out=0;
    end
    else if(a==0 && b==0 && c_in==1)
    begin
        s=1;
        c_out=0;
    end
    else if(a==0 && b==1 && c_in==0)
    begin
        s=1;
        c_out=0;
    end
    else if(a==0 && b==1 && c_in==1)
    begin
        s=0;
        c_out=1;
    end
    else if(a==1 && b==0 && c_in==0)
    begin
        s=1;
        c_out=0;
    end
    else if(a==1 && b==0 && c_in==1)
    begin
        s=0;
        c_out=1;
    end
    else if(a==1 && b==1 && c_in==0)
    begin
        s=0;
        c_out=1;
    end
    else if(a==1 && b==1 && c_in==1)
    begin
        s=1;
        c_out=0;
    end
end
end
```

```
else if(a==0 && b==1 && c_in==1)
begin
    s=0;
    c_out=1;
end
else if(a==1 && b==0 && c_in==0)
begin
    s=1;
    c_out=0;
end
else if(a==1 && b==0 && c_in==1)
begin
    s=0;
    c_out=1;
end
else if(a==1 && b==1 && c_in==0)
begin
    s=0;
    c_out=1;
end
else if(a==1 && b==1 && c_in==1)
begin
    s=1;
    c_out=1;
end
end
endmodule
```

