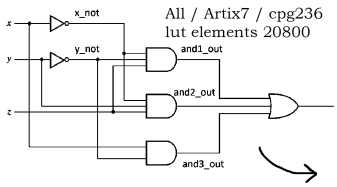
Theorem		Dual		Name
T6	$B \bullet C = C \bullet B$	T6′	$B + C = C + B$	Commutativity
T7	$(B \bullet C) \bullet D = B \bullet (C \bullet D)$	T7′	$(B + C) + D = B + (C + D)$	Associativity
T8	$(B \bullet C) + (B \bullet D) = B \bullet (C + D)$	T8′	$(B + C) \bullet (B + D) = B + (C \bullet D)$	Distributivity
T9	$B \bullet (B + C) = B$	T9′	$B + (B \bullet C) = B$	Covering
T10	$(B \bullet C) + (B \bullet \overline{C}) = B$	T10′	$(B + C) \bullet (B + \overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D) = (B \bullet C) + (\overline{B} \bullet D)$	T11′	$(B + C) \bullet (\overline{B} + D) \bullet (C + D) = (B + C) \bullet (\overline{B} + D)$	Consensus
T12	$\overline{B_0 \bullet B_1 \ldots} = (\overline{B_0} + \overline{B_1} + \overline{B_2} \ldots)$	T12′	$\overline{B_0 + B_1 + B_2 \ldots} = (\overline{B_0} \bullet \overline{B_1} \bullet \overline{B_2} \ldots)$	De Morgan's Theorem

Table 2.3
Minterms and Maxterms for Three Binary Variables

x	y	z	Minterms Term	Minterms Designation	Maxterms Term	Maxterms Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

{} {{}}	Concatenation, replication
unary ÷ unary -	Unary operators
+ - * / **	Arithmetic
%	Modulus
> >= < <=	Relational
!	Logical negation
&&	Logical and
||	Logical or
==	Logical equality
!=	Logical inequality
===	Case equality
!==	Case inequality
~	Bitwise negation
&	Bitwise and
|	Bitwise inclusive or
^	Bitwise exclusive or
^~ or ~^	Bitwise equivalence
&	Reduction and
~&	Reduction nand
|	Reduction or
~|	Reduction nor
^	Reduction xor
~^ or ^~	Reduction xnor
<<	Logical left shift
>>	Logical right shift
<<<	Arithmetic left shift
>>>	Arithmetic right shift
?:	Conditional

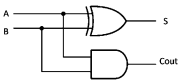
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F: 0 0 | 0; 0 1 | 0; 1 0 | 0; 1 1 | 1
OR		$F = x + y$	x y | F: 0 0 | 0; 0 1 | 1; 1 0 | 1; 1 1 | 1
Inverter		$F = x'$	x | F: 0 | 1; 1 | 0
Buffer		$F = x$	x | F: 0 | 0; 1 | 1
NAND		$F = (xy)'$	x y | F: 0 0 | 1; 0 1 | 1; 1 0 | 1; 1 1 | 0
NOR		$F = (x + y)'$	x y | F: 0 0 | 1; 0 1 | 0; 1 0 | 0; 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F: 0 0 | 0; 0 1 | 1; 1 0 | 1; 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x y | F: 0 0 | 1; 0 1 | 0; 1 0 | 0; 1 1 | 1



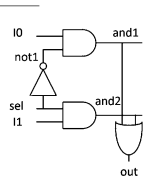

```
module ornek2
(
input x,
input y,
input z,
output F2
);

wire x_not;
wire y_not;
wire and1_out;
wire and2_out;
wire and3_out;

not G1 (x_not,x);
not G2 (y_not,y);
and G3 (and1_out,x_not,y_not,z);
and G4 (and2_out,x_not,y,z);
and G5 (and3_out,x,y_not);
or G6 (F2,and1_out,and2_out,and3_out);

endmodule
```

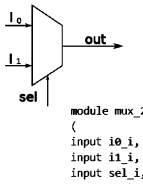
Giriş		Çıkış	
A	B	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S = A'B+AB' → S = A xor B
Cout = AB




```
module half_adder
(
input a_i,
input b_i,
output s_o,
output cout_o
);

xor G1 (s_o,a_i,b_i);
and G2 (cout_o,a_i,b_i);

endmodule
```





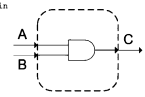

```
module mux_2_1
(
input i0_i,
input i1_i,
input sel_i,
output out_o
);

wire not1, and1, and2;

not G1 (not1,sel_i);
and G2 (and1,i0_i,not1);
and G3 (and2,i1_i,sel_i);
or G4 (out_o,and1,and2);

endmodule
```

```
initial begin
    a_i    = 2'b00;
    b_i    = 2'b00;
    cin_i  = 1'b0;
    #10;
    a_i    = 2'b01;
    b_i    = 2'b10;
    cin_i  = 1'b1;
    #10;
    a_i    = 2'b11;
    b_i    = 2'b11;
    cin_i  = 1'b1;
    #10;
    a_i    = 2'b11;
    b_i    = 2'b11;
    cin_i  = 1'b0;
    #10;
    a_i    = 2'b01;
    b_i    = 2'b01;
    cin_i  = 1'b1;
    #10;
    $finish;
end
```

```
module Adder3Bit_behavioral (
    input[2:0] num1,
    input[2:0] num2,
    output reg[3:0] sum);

    always@(num1 or num2)
begin
        sum = num1 + num2;
    end
endmodule
```

Hassasiyet Listesi

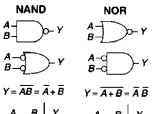
```
always@( A or B ) begin
    C = A & B;
end

always@( * ) begin
    C = A & B;
end
```




```
always@( A ) begin
    C = A & B;
end

always@( B ) begin
    C = A & B;
end
```

```
module Adder3Bit_behavioral (
    input[2:0] num1,
    input[2:0] num2,
    output reg[3:0] sum
    );
    always@( * )begin
        sum = num1 + num2;
    end
endmodule
```

```
module Adder3Bit_behavioral (
    input[2:0] num1,
    input[2:0] num2,
    output reg[3:0] sum
    );
    always@(num1 or num2)begin
        sum = num1 + num2;
    end
endmodule
```



A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

```
module half_adder_behav
(
input wire a_i,
input wire b_i,
output reg s_o,
output reg cout_o
);

always @(a_i,b_i) begin
    {cout_o,s_o} = a_i + b_i;
end

endmodule
```

```
module half_adder_behav
(
input a_i,
input b_i,
output s_o,
output cout_o
);

reg s,cout;

always @(a_i,b_i) begin
    {cout,s} = a_i + b_i;
end

assign cout_o  = cout;
assign s_o     = s;

endmodule
```

```
module add_sub_8bit
(
input [7:0] in0_i,
input [7:0] in1_i,
input op_i,
output [7:0] result_o
);

reg [7:0] result;

//always @(in0_i, in1_i, op_i) begin
always @(*) begin
    if (op_i == 1'b0) begin
        result = in0_i + in1_i;
    end
    else begin
        result = in0_i - in1_i;
    end
end

assign result_o = result;

endmodule
```

always@ Bloğu

Nasıl Tanımlanır?

```
always@( hassasiyet_listesi ) begin
   //İşlemler
   //.
   //.
   //.
end
```

- always@* bloğu yapılan işlemlerin sağında kullanılan değişkenlerin(num1, num2) herbangi birinin değişmesi sonucu çalışır.
- Bu nedenle düzgün bir devre için atama yapılan değerlerin(sum) kullanılmamasına(hem sağda hem de solda olma durumu) dikkat edilmelidir.
- Böyle bir durum sonucunda blok tekrardan tetiklenecek ve yaptığı işlemi bırakıp baştan başlayacaktır.

Wire

wire değişkenine yapılacak atamalar always ve initial blokları dışında yapılmak zorundadır.

a. Önceki haftalarda kullanılan kapı seviyesindeki atamalar

```
and kapi_atamasi(C,A,B);
```

b. "assign" ile yapılan atamalar

```
assign C = A & B;
```

wire Verilog dilinde yol(bus) atamaları için kullanılır. Amacı gelen veriyi aktarmak ve aradaki bağlantıyı sağlamaktır. Modüllere giriş ve çıkış olarak tanımladığınız değişkenler varsayılan olarak wire türündedir. Sadece output belirtildiği takdirde reg tipi olarak kullanılabilir.

Dışarıdan modül çağırdığımızda bağlanacak sinyal girişler için **wire** veya reg, çıkışlar için sadece **wire** tipinde olmalıdır.

```
wire A,B;
wire C;
modul_ismi  wire_reg_farki(.In1( A ),
                           .In2( B ),
                           .Out( C ),
) ;
```

```
reg A,B;
wire C;
modul_ismi  wire_reg_farki(.In1( A ),
                           .In2( B ),
                           .Out( C ),
                           ) ;
```

```
wire [3:0] bitwisenot_A;
reg [3:0] A = 4'b1010;
assign bitwisenot_A = ~A;
```

~1	~0	~1	~0	⇨	0	1	0	1

Cicikuş

```
wire [3:0] A_bitwiseand_B;
reg [3:0] A = 4'b1010;
reg [3:0] B = 4'b1101;
assign A_bitwiseand_B = A & B;
```

1&1	0&1	1&0	0&1	⇨	1	0	0	0

eklenen_modül_ismi tasarımda_kullanılacak_isim (**.input1**(wire1),
.input2(wire2),
.input3(wire3),
.output1(wire4),
.output2(wire5));

eklenen_modül_ismi: Önceden tasarlamış olduğunuz modülün ismi. (Sınıf ismi gibi düşünülebilir)

tasarımda_kullanılacak_isim: Tasarımınıza eklediğiniz modüle vereceğiniz isim (Bir sınıftan oluşturduğunuz değişkenin ismi gibi düşünülebilir)

input1-4: Eklediğiniz modülün giriş pinlerinin ismi

output1-2: Eklediğiniz modülün çıkış pinlerinin ismi

Giriş/çıkış pinlerinin başına nokta ve aralarına virgül konulmalıdır.

wire1-5: Eklenen modüle bağlanacak kabloların belirtir. Modül giriş ve çıkışlarına sadece kablo bağlanmak zorunda değildir fakat bu konu sonraki laboratuvarlarda anlatılacaktır.

Eklenecek modül aynı zamanda aşağıdaki gibi de tanımlanabilir

```
eklenen_modül_ismi  tasarımda_kullanılacak_isim ( .girisler  (wire/reg) , .cikislar  (wire)  );
```

reg A;
wire B;
wire C;
wire D;

```
lab3_oncesi test1( .giris1(A),
                   .giris2(B),
                   .cikis1(C),
                   .cikis2(D)
                   );
```

```
module lab3_oncesi(
        input giris1,
        input giris2,
        output cikis1,
        output cikis2
);
```

kasır

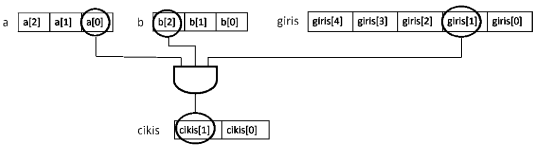
- Giriş-Çıkışlar veya wire/reg değişkenler

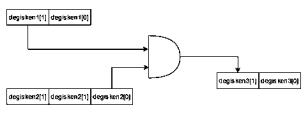
```
module f (
    input [4:0] giris,
    output [1:0] cikis
);

wire [2:0] a,b;

and a0(cikis[1], a[0], b[2], giris[1]);
xor a1(cikis[0], a[1], b[1], giris[4]);

endmodule
```

giris: giris[4] | giris[3] | giris[2] | giris[1] | giris[0]
cikis: cikis[1] | cikis[0]
a: a[2] | a[1] | a[0]
b: b[2] | b[1] | b[0]

- Çok bit genişliğindeki değerler ile işlemler

Bit seçimi : and a0(cikis[1], a[0], b[2], giris[1]);

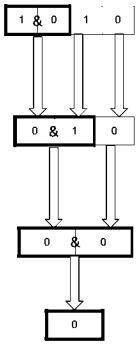




Reduction İşlemleri

Reduction	
&a	AND
~&a	NAND
|a	OR
~|a	NOR
^a	XOR
~^a ^~a	XNOR

```
wire [3:0] reductionand_A;
reg [3:0] A = 4'b1010;
assign reductionand_A = &A;
```



- Reduction işlemleri sonucu oluşan sonuç her zaman tek bittir.
- Reduction işlemleri sadece tek değişkene uygulanır.

Çok bitli değişkenlerin birden fazla bit seçimi aşağıda verilen örnekteki gibi yapılır.

```
reg [2:0] tb_giris1;
reg [3:0] tb_giris2;

initial begin
tb_giris1= 3'b110;
tb_giris2[1:0]=2'b10;
tb_giris2[3:2]=2'b01;
end
```

Burada tb_giris1 direkt olarak *3'b110* değeri, tb_giris2 sinyaline ise *4'b0110* değeri atanmıştır.

tb_giris2[1:0] = *2'b10* satırından sonra oluşan **tb_giris2** sinyali aşağıdaki gibidir:

X	X	1	0

tb_giris2[3:2] = *2'b01* satırından sonra oluşan **tb_giris2** sinyali aşağıdaki gibidir:

0	1	1	0

```
module lab4(
    input A,B,
    output reg C_out
    );

    reg C_yanlis;
    reg C_dogru;
    initial begin
        C_yanlis = A & B;
    end
    initial begin
        C_dogru = 1'b1 & 1'b0;
    end

    always@* begin
        C_out = A & B;
    end

endmodule
```

```
module tb_sorula();
 reg Atb,Btb,Ctb;
 wire Ftb,Qtb;
 sorula uut(
    .A(Atb),
    .B(Btb),
    .C(Ctb),
    .F(Ftb),
    .Q(Qtb)
 );
 initial begin
     Atb = 0;
     Btb = 0;
     Ctb = 0;
     #10;
     Atb = 0;
     Btb = 0;
     Ctb = 1;
     #10;
     Atb = 0;
     Btb = 1;
     Ctb = 0;
     #10;
     Atb = 0;
     Btb = 1;
     Ctb = 1;
     #10;
  end
endmodule
```

```
wire Trigger , Pass;
reg A, C;
always @( * ) begin
    if(Trigger) begin
        A = Pass;
        C = Pass;
    end
    else begin
        A=1'b0;
        C=1'b1;
    end
end
```

```
wire [3:0] bitwisenot_A;
reg [3:0] A = 4'b1010;
assign bitwisenot_A = ~A;
always @( * ) begin
case(Trigger)
    1'b0: begin
            A = 1'b0;
            C = 1'b1;
          end
    1'b1: begin
            A = Pass;
            C = Pass;
          end
endcase
end
```

```
module agalar_ders(
input [3:0]sayi1,sayi2,

output [2:0]F

    );
    wire A,B,C,D;
    xor (A,sayi1[3],sayi2[3]);
    xor (B,sayi1[2],sayi2[2]);
    xor (C,sayi1[1],sayi2[1]);
    xor (D,sayi1[0],sayi2[0]);

    wire nA,nB,nC,nD;
    not (nA,A);
    not (nB,B);
    not (nC,C);
    not (nD,D);

    wire w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11,w12,w13,w14;

    and (w1,A,nB,nD,nC);
    and (w2,A,D,B,nC);
    and (w3,A,nB,D,C);
    and (w4,A,B,C,nD);
    and (w5,nA,B,nC,D);
    and (w6,nA,nB,nC,D);
    and (w7,nA,B,C,D);
    and (w8,nA,nB,C,nD);

    or (F[0],w1,w2,w3,w4,w5,w6,w7,w8);

    and (w9,A,D,nC);
    and (w10,A,B,nC);
    and (w11,nB,C,A);
    and (w12,B,nD,C);
    and (w13,C,D,nA);
    and (w14,B,D,nA);

    or (F[1],w9,w10,w11,w12,w13,w14);

    and (F[2],A,B,C,D);

endmodule
```

```
    module belki_olur(
    input[9:0]bolunen,
    input [5:0]bolen,
    output reg [3:0]sonuc1,
    output reg [3:0]sonuc2,
    output reg [3:0]sonuc3

        );
    reg [31:0] bolunenc;

    always@* begin
    bolunenc=(bolunen[9:2]100)+(bolunen[1]50)+(bolunen[0]*25);

    bolunenc=(bolunenc/bolen);

    sonuc3=(bolunenc%10);
    bolunenc=((bolunenc-sonuc3)/10);
    sonuc2=(bolunenc%10);
    bolunenc=(bolunenc-sonuc2)/10;
    sonuc1=bolunenc%10;

    end

    endmodule
```

```
   module belkiolur_tb(
    );
       reg [9:0]a;
       reg [5:0]b;

       wire [3:0]c;
       wire [3:0]d;
       wire [3:0]e;
       belki_olur sahin(.bolunen(a),.bolen(b),.sonuc1(c),.sonuc2(d),.sonuc3(e));

       initial begin
       a=10'b0111110100;b=4;#10;//125/4=31.25 1 2 5
       a=10'b0000011001;b=3;#10;//6.25/3=2.0833 2 0 8
       end
   endmodule
```

```
    module maske(
    input [7:0]deger,
    input [3:0] ilk_maske,
    input [3:0] isle_maske,
    input islem,
    output reg[7:0] maskelenmis_deger

        );
        reg [3:0]a;
        reg [3:0]b;
        reg [3:0]c;
        always@(*)begin
         if(islem==1)begin
              a=ilk_maske^isle_maske;
         end
         else begin
              a=ilk_maske~^isle_maske;
         end
         if(islem==1)begin
              b=deger[7:4]^a;
              c=deger[3:0]^a;
          end
          else begin
              b=deger[7:4] ~^a;
              c=deger[3:0] ~^a;
         end
              maskelenmis_deger[7:4]=b;
              maskelenmis_deger[3:0]=c;
        end

    endmodule
```

```
module sahin_soru_tb(
    );
    reg[7:0]deger;
    reg[3:0]ilk_maske;
    reg[3:0]isle_maske;
    reg islem;
    wire [7:0]maskelenmis_deger;

    maske sahin(.deger(deger),
     .ilk_maske(ilk_maske),
     .isle_maske(isle_maske),
     .islem(islem),
     .maskelenmis_deger(maskelenmis_deger));

    initial begin

    deger=8'b01011011;
    ilk_maske=4'b1011;
    isle_maske=4'b0110;
    islem=1'b1;

    end

endmodule
```