Verilog HDL'e Giriş

Bilg. Yük. Müh. Selçuk BAŞAK

SelSistem Bilgi ve İletişim Teknolojileri
www.selsistem.com.tr

Donanım Tanımlama Dilleri - HDL

- İlk olarak 1977 yılında,
  - ISP(Instruction Set Processor) - Carnegie Mellon Universitesi
  - KARL- Kaiserslautern Üniversitesi
- İlk modern HDL olan Verilog, 1985, Gateway Design Automation.
- VHDL, 1987, Amerika Birleşik Devletleri Savunma Bakanlığı Sponsorluğunda.
- HDL şematik devre tasarımlarının dökümantasyonu ve simulasyonu amacıyla geliştirilmişti.
- HDL'den Lojik sentezleme yapılması ile HDL Dijital Tasarımda ön plana geçti.

Verilog - Giriş

- Verilog HDL, 1984-1985 Philip Morby, Gateway Design Automation
- Amaç : Dijital devreleri, modelleme, simülasyon ve analiz amacıyla kolay, basit ve etkili bir şekilde ifade etmek.
- IEEE tarafından ilk olarak1995 yılında standardlaştırıldı.
- 2001 yılında standard Verilog 2001 olarak güncellendi.
- C programlama dilini andırır. Öğrenilmesi nispeten kolaydır

Verilog - Syntax

- Case sensitive bir dildir
  - Anahtar sözcükler küçük karakter ile yazılırlar.
- İfadeler “;” ile sonlandırılır
- Yorumlar;
  - // dan sonraki tüm karakterler
  - /* -- */ arasındaki tüm karakterler yorum olarak algılanırlar

Verilog - Syntax

- Değişkenlerin yaşam alanları vardır.
  - Aynı alanda geçerli değişkenler için farklı isimler kullanılmalıdır
  - Maksimum 1024 karakter olabilir
  - İlk karakteri sayı olamaz,büyük ve küçük alfabetik, nümerik ve alt çizgi karakterlerinden oluşabilir
- Boşluklar ve boş satırlar istenildiği gibi bırakılabilir

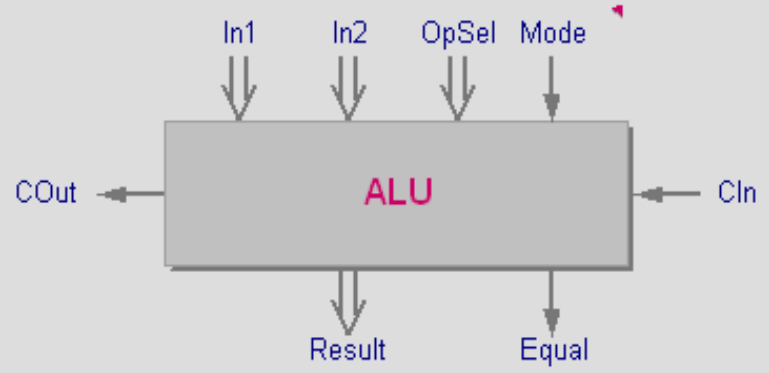
Verilog - Modül

- Verilog ile dijital sistemler hiyerarşik modüllerden ve bu modüllerin arasındaki bağlantılar ile tanımlanır.
- Modülün içeriği,
  - Yapısal (Düşük seviyeli lojik ve alt moduller ile),
  - Veri akışı ile (Çıkışları giriş sinyallerinin dönüşümü ile)
  - Davranışsal olarak (Devreden beklenen davranışın programsal ifadesi şeklinde)

  üç değişik tarzda ifade edilebilir.
- Bu üç stil, üst seviyeli modüllerde soyutlamaya gidilebilmesini ve alt seviyeli modüllerinde de donanıma daha yakın ve verimli olarak ifade etme imkanı sağlar.
- Modül içinde birden fazla tarz ifade beraber kullanılabilir.

Verilog - Modül

- Modüller, giriş/çıkış portları ve bu giriş sinyaleri ile çıkış sinyallerini belirleyen içerik kısmından oluşur
- Modülde, port listesi modül başlığında adları tanımlanıp altta yönleri ve boyutları belirtilir(Verilog-1995)
- Verilog 2001 ile portların yön ve boyuları başlık içinde de tanımlanabilir.
- “endmodule” anahtar sözcüğü ile tasarım tamamlanmış olur

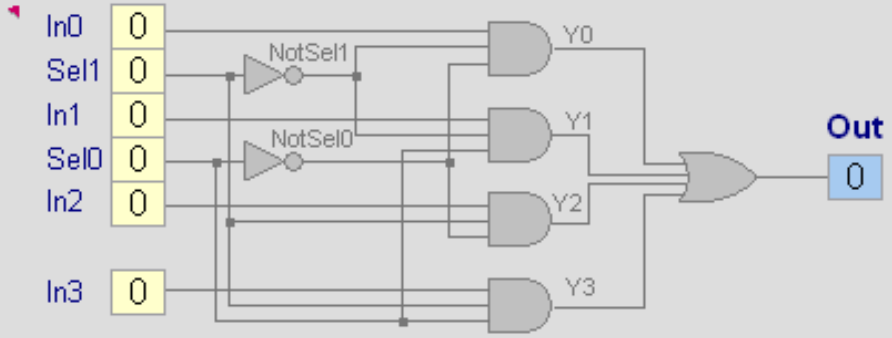



```
module ALU (Result, COut, Equal, In1, In2,
            OpSel, CIn, Mode);

output [3:0] Result;   // operation result
output       COut;     // carry out
output       Equal;    // when 1, In1 = In2
input  [3:0] In1;      // first operand
input  [3:0] In2;      // second operand
input  [3:0] OpSel;    // operation select
input        CIn;      // carry in
input        Mode;     // mode arithm/logic;
                       // arithm when 0
 . . .

endmodule
```

```
module  ALU(output [3:0] Result,
output Cout, output equal,
input [3:0] In1, ... );
...
endmodule
```

Verilog – Structural Style (Yapısal)

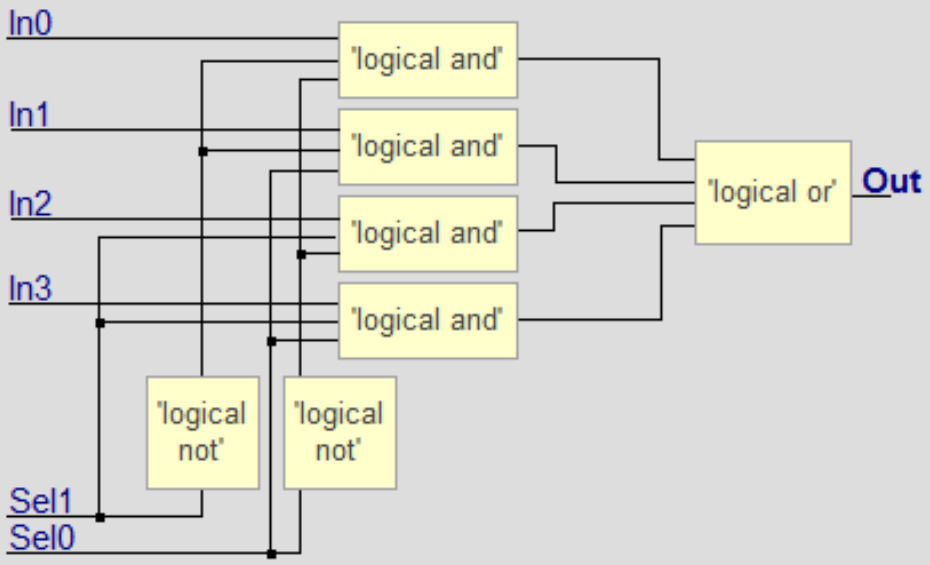
- Yapısal tarzda ifade primitive kapı türünden veya alt modüller ile yapılılır.
- Ön tanımlı primitive yapılar kullanılırken genellikle ilk eleman çıkış olur.
- Kullanıcı tanımlı alt modullerde giriş ve çıkışların sırası alt modülün tasarımına bağlıdır.




```
module mux_4_to_1 (Out,In0,In1,In2,In3,Sel1,Sel0);
output Out;
input In0, In1, In2, In3, Sel0, Sel1;

wire NotSel0, NotSel1;
wire Y0, Y1, Y2, Y3;

not (NotSel0, Sel0);
not (NotSel1, Sel1);
and (Y0, In0, NotSel1, NotSel0);
and (Y1, In1, NotSel1, Sel0);
and (Y2, In2, Sel1, NotSel0);
and (Y3, In3, Sel1, Sel0);
or (Out, Y0, Y1, Y2, Y3);

endmodule
```

Verilog – Dataflow Style (Veri Akışı)

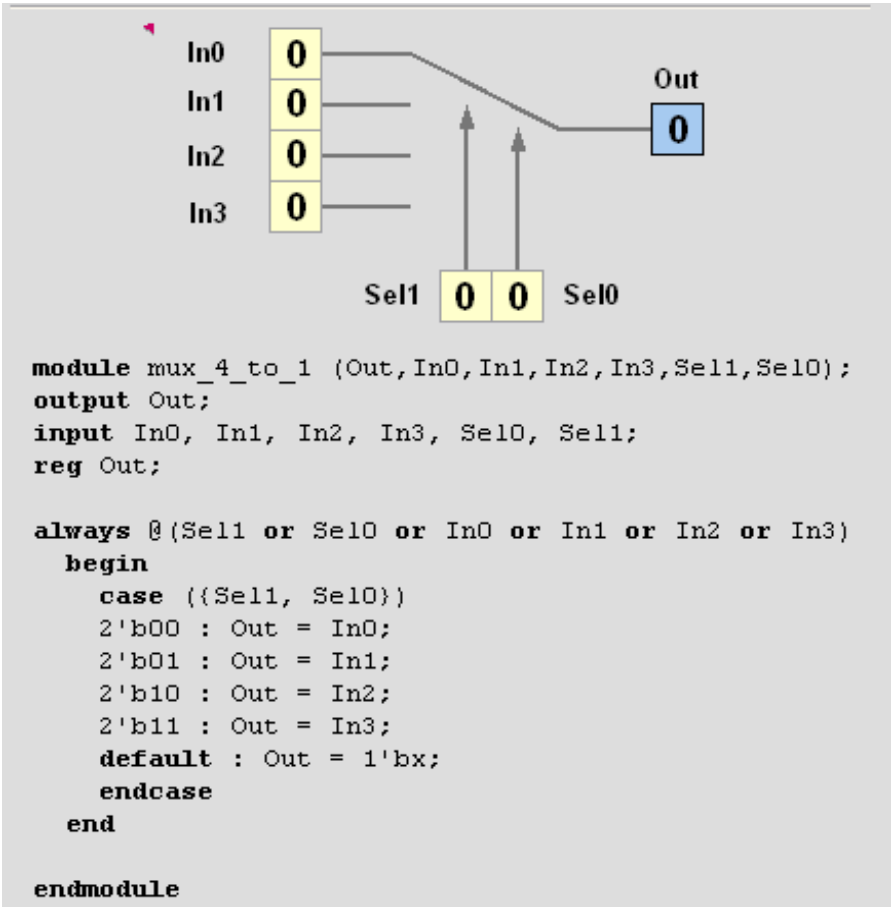
- Veri akışı tarzında, çıkış sinyalleri giriş sinyallerinin bir fonksiyonu olarak ifadesi edilir.




```
module mux_4_to_1 (Out, In0, In1, In2, In3, Sel1, Sel0);

output Out;
input In0, In1, In2, In3, Sel0, Sel1;

assign Out = (~Sel1 & ~Sel0 & In0) | (~Sel1 & Sel0 & In1)
             | (Sel1 & ~Sel0 & In2)| (Sel1 & Sel0 & In3);

endmodule
```

Verilog – Behaviral Style (Davranışsal)

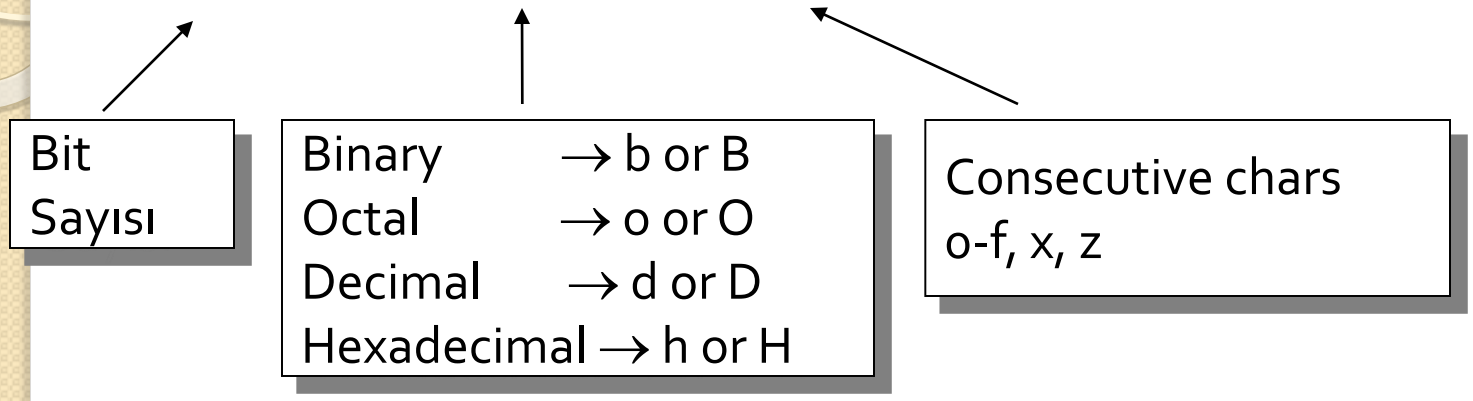
- Davranışsal tarzda, tasarlanmak istenen sistemin davranışı «always» blokları içinde programlama dillerinde de kullanılan «if/else» veya «case» gibi yapılar ile ifade edilerek yapılır.
- Always içerisinde sadece reg tipinde sinyallere atama yapılabilir.
- Örnekte «Out» portuna atama yapılabilmesi için reg olarak tanımlanmıştır.




```
module mux_4_to_1 (Out,In0,In1,In2,In3,Sel1,Sel0);
output Out;
input In0, In1, In2, In3, Sel0, Sel1;
reg Out;

always @(Sel1 or Sel0 or In0 or In1 or In2 or In3)
  begin
    case ({Sel1, Sel0})
    2'b00 : Out = In0;
    2'b01 : Out = In1;
    2'b10 : Out = In2;
    2'b11 : Out = In3;
    default : Out = 1'bx;
    endcase
  end

endmodule
```

Verilog – Sayılar

- <size>'<radix> <value>



- 3'b001
- 8'hAB
- 8'd4
- 124

Verilog – Veri Türleri

- Verilog 4 değerli mantık sistemine sahiptir.

(0, 1, z, x)

- Net Tipi: Donanım elemanlarını birleştiren fiziksel bağlantı, «kablo gibi»
  - **wire** ile tanımlanan sinyaller
- Register tipi: Atama yapılmadığında mevcut değerini değerini saklar, «değişken gibi»
  - **reg** ile tanımlanan sinyaller

Verilog – Bus/Vektör

type [MSB:LSB] varname;

Örnek:

```
reg [31:0] a; // 32 bit wide reg type bus
wire [7:0] b; // 8 bit wide wire type bus
wire temp; // 1 bit net type signal
```

Bus elemanlarına erişim örnekleri

- a[2] = 1‘b1;
- temp = a[0];
- b[7] = temp;
- a[7:0] = b;
- a[7:5] = b[2:0];

Verilog - Operatörler

- Arithmetic Operations (binary: +, -,*,/,%,**); (unary: +, -)
- Bitwise (~, &,|,^, ~&,~|,~^)
- Reduction (&,~&,|,~|,^,~^)
- Logical (!,&&,||,==,!=,===,!==)
- Relational (<,<=,>,>=)
- Shift (>>,<<)
- Arithmetic Shift (>>>,<<<)
- Conditional ? :
- Concatenation {A,B}
- Replication {4{B}}

Verilog – Primitives - Structured

- Yapısal tarzda ifadelerler aşağıda ön tanımlı primitive'ler kullanılabilir
- Primitive modullerin genel olarak ilk portları output port'udur.
- and, or, xor, not, nor, nand,xnor, ...
- Ör:
- primitive instance_name(result,in1,in2,in3...);
- and a1(a,b,c);

Verilog – Continuous Assignment-Dataflow

- Yazım kuralı:

  assign *signal_name* = <*expr*>;

- Bunarı nereye yazmalıyız:
  - Modülün içerisinde
  - Prosedürlerin dışarısına
- Özellikler:
  - Hepsi paralel işlenir
  - Birbirinden bağımsız, aynı zamanda aktif olurlar

Verilog – Procedure - Behavioral

- Modüller istenen sayıda procedure içerebilir
- Procedure ler iki tür blok ile ifade edilirler:
  - initial → Sadece bir kez işlenir. Sentezlenebilir değildir. Simulasyon için kullanılır.
  - always → Sonsuza kadar işlenir. Sentezlenebilir.
- Bu bloklarda yalnızca reg tipi sinyallere değer atanabilir

Verilog – Always

- Bir duyarlılık listesine sahiptir
- Duyarlılık listesindeki herhangi bir değişken değiştiğinde her zaman işlenir. Verilog 2001'de «@(*)» ile blok içindeki referans edilen tüm sinyaller duyarllık listesine dahil edilebilir.
- Sadece reg tipi sinyallere atama yapılabilir!
- wire türlerine atama yapılamaz!
- İçerisinde "=" seri(blocking) ve "<=" paralel(blocking) atamaları yapılabilir.
- İçerisinde if/else, case gibi procedural yapılar kullanılabilir.

Verilog – Always (örnek)

```
module fa_beh (input A, B, Cin, output reg Sum, Cout) ;

alyaws@(*)
begin
    {Cout,Sum}  = A + B + Cin;
end

endmodule
```

Verilog - Atama kuralları

- **reg tipi sinyaller** modülün her yerinde referans gösterilebilirler
- **reg tipi sinyaller** sadece prosedürel ifadelerde yani always ve initial bloklarının içerisinde, task ya da functionlarda atama yapılabilir
- **reg tipi sinyaller modülün input** ya da **inout portları** olamazlar
- **reg tipi sinyaller** alt modüllerin output veya inout portlarına bağlanamaz
- Net değişkenler modülün her yerinde referans gösterilebilirler
- Net değişkenleri behavior, task or function içerisinde atanamazlar. Net değişkenleri modül içerisinde primitive, alt modül output portu, continuous assignment veya module input portu tarafından sürülmelidir

Verilog – Bloklu ve Bloksuz Atamalar

- Bloksuz Atamalar
  - "<= " ile atama yapılır
  - İşlemler paralel bir şekilde aynı anda yapılır

Örnek:*

```
always @(*)
   begin
      b <= 0; c <= 0;
      b <= a + a;
      c <= b + a;
      d <= c  + a;
end
```

- * Bu örnek sentezlemede hata verir, aynı sinyal(b,c) birden fazla sürücüye sahip

- Bloklu Atamalar
  - "=" ile gerçekleştirilir
  - Bloklu atamalar sıralı gerçekleştirilir

Örnek:

```
always @(*)
begin
      b = 0; c = 0;
      b = a + a;
      c = b + a;
      d = c  + a;
end
```

Verilog – (if…else)

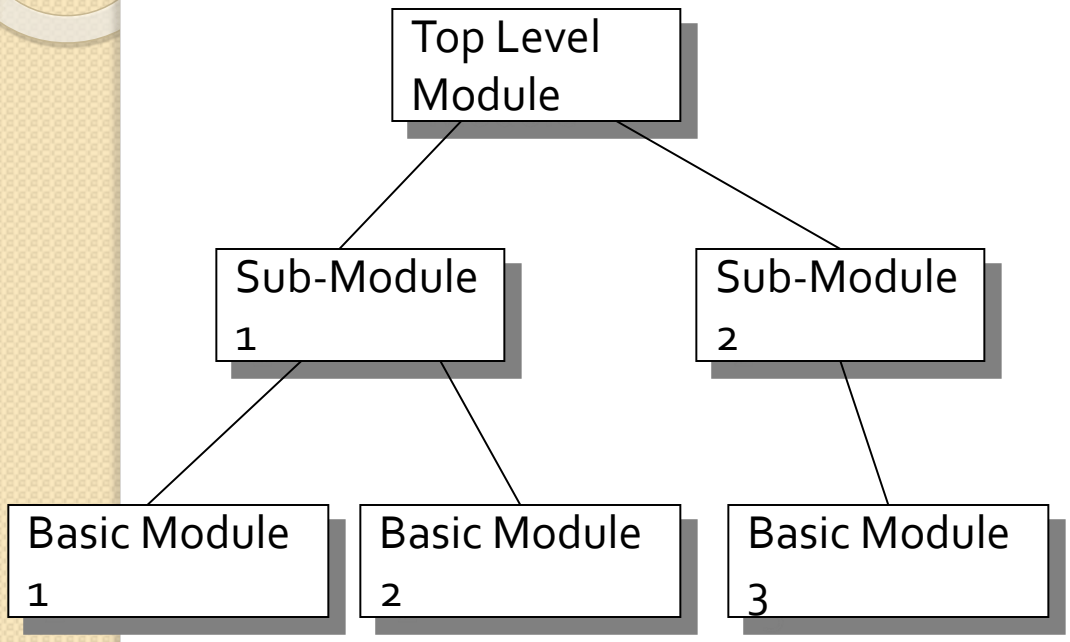
```
always@(*)
  begin
     if (a == b)
     begin
                  q <= data;
                  stop <= 1'b1;
     end
     else   if (a > b)
     begin
                  q <= a;
                  stop <= 1'b0;
     end
     else
     begin
                  q <= b;
                  stop <= 1'b0;
     end
 end
```

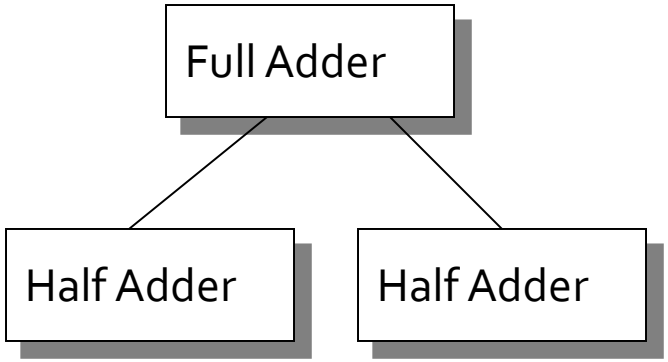
- Tüm if/else her hangi bir dalında atama yapılan bir değişkene diğer tüm dallarda da atama yapılmalıdır yoksa combinational lojik tasarımda istenmeyen latch oluşturabilirsiniz
- **Else** en yakın **if** ile ilişkilidir. **Begin** ve **end** kullanarak okunabilirlği arttırabilirsiniz.
- **if** … **else if** … **else** kullanımı seri ya da öncelikli devre oluşturur.

Verilog - Case

```
always@(*)
begin
        case (state)
          2'b00: next_state <= s1;
          2'b01: next_state <= s2;
          2'b10: begin
                    if (x) next_state <= s0;
                       else next_state <= s1;
                    end
         default next_state <= 1'bxx;
        endcase
 end
```

Verilog – Hiyerarşik Dizayn

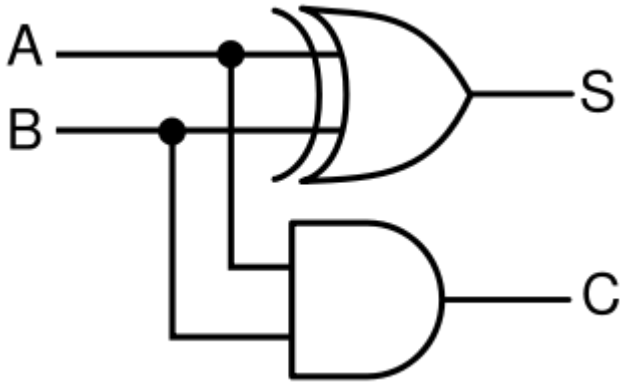




Hiyerarşik tasarım yapısal tarzda ifade edilir.

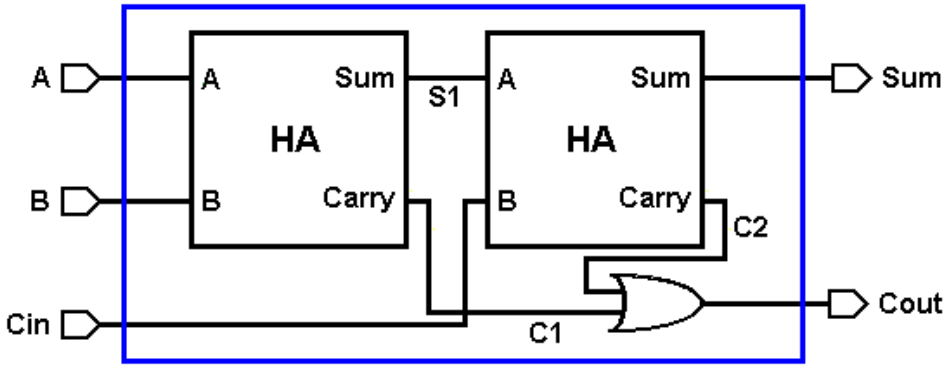
Verilog –Structural Style (Yapısal)

```
module half_add ( input A, B,
                  output S, C);
    xor  u1 (S, A, B) ;
    and u2 (C, A, B) ;

endmodule
```



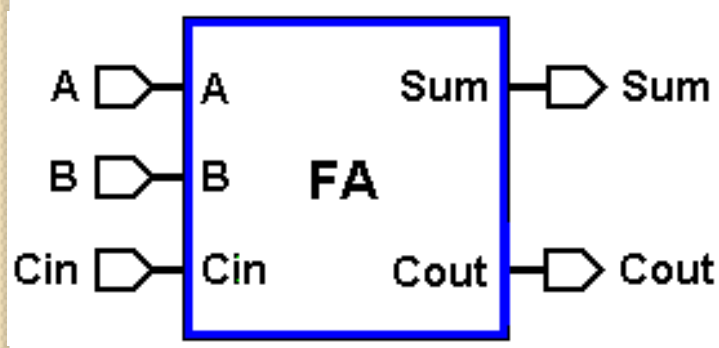
Verilog – Structural Style - Yapısal

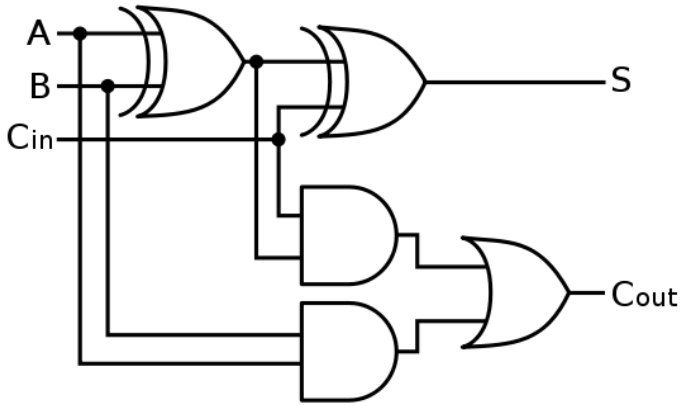
```
module full_adder (input  A, B, CI,
                   output Sum, Cout) ;
wire S1, C1, C2;

half_add HA1 (A, B, S1, C1);
half_add HA2 (S1, CI, Sum, C2);

or P1 (Cout, C1, C2);

endmodule
```



Verilog – Dataflow Style/ Veri Akışı

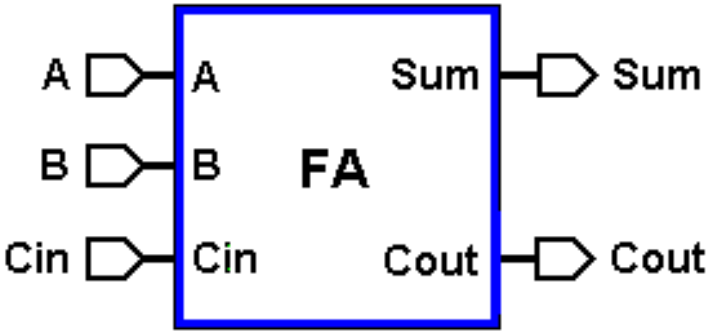
```
module fa_rtl (input A, B, Cin, output Sum, Cout) ;

assign Sum = A ^  B ^ Cin;                    //continuous assignment
assign Cout = A & B | (A ^ B )& Cin;          //continuous assignment

endmodule
```





Verilog – Behavioral Style

```
module fa_beh (input A, B, Cin, output reg Sum, Cout) ;

alyaws@(*)
begin
    {Cout,Sum}  = A + B + Cin;
end

endmodule
```



Örnek Uygulamalar

Verilog – Synchronous Logic

Saat sinyalinin yükselen kenarı:posedge

düşen kenarı: negedge

- Örnek: D-Flip Flop

```
module DFF (input d,clk, output reg q);

always @(posedge clk)
begin
        q <= d;
end
endmodule
```