SOC Design

Week 2: RTL Design

Fenerbahçe University

Professor & TAs

Prof: Dr. Vecdi Emre Levent

Office: 311

Email: emre.levent@fbu.edu.tr

TA: Arş. Gör. Ezgi Çakmak

Office: 311

Email: ezgi.cakmak@fbu.edu.tr

RTL Design

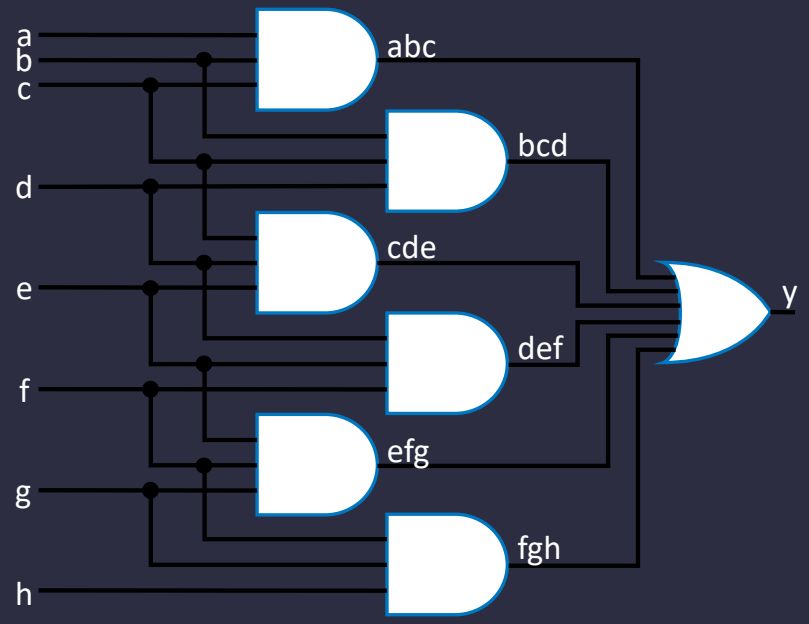
- RTL Design

Kombinasyonel Devre Tasarım Süreci

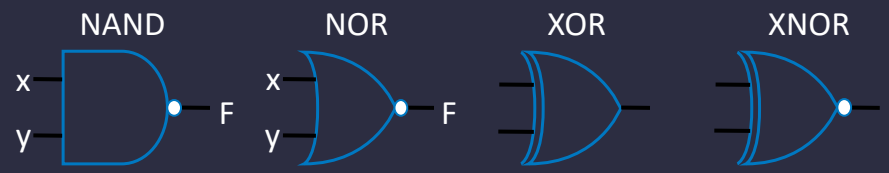
	Adım	Açıklama
Adım 1	**Fonksiyonu Oluştur**	Doğruluk tablosu veya denklem oluşturulur
Adım 2	**Denkleme 'e Dönüştür**	Doğruluk tablosun'dan denkleme dönüştürülür. Denklem, doğruluk tablosunun 1 olan satırlarının birlikte kullanılması ile oluşturulur.
Adım 3	**Mantık Kapısı Devresi Oluştur**	Oluşturulan denkleme karşılık gelen devre çizilir

Örnek: Ardışık 3 tane 1 bulan devre

- Örnek: 8 bitlik sayının içinden ardışık olaran 3 tane 1 olan devre: abcdefgh
  - 000**111**01 → 1 10101011 → 0
    11110000 → 1
  - **Adım 1: Fonksiyonu oluştur**
    - Doğruluk tablosu veya denklem?
      - Doğruluk tablosu 2^8=256 satır olacağı için çok uzun olacaktır.
      - Denklem: Sonucun 1 çıkacağı denklemleri tespit edip birleştirin
    - **y = abc + bcd + cde + def + efg + fgh**
  - **Adım 2: Denkleme dönüştür** -- tamamlandı
  - **Adım 3: Devreyi oluştur**



Mantık Kapıları

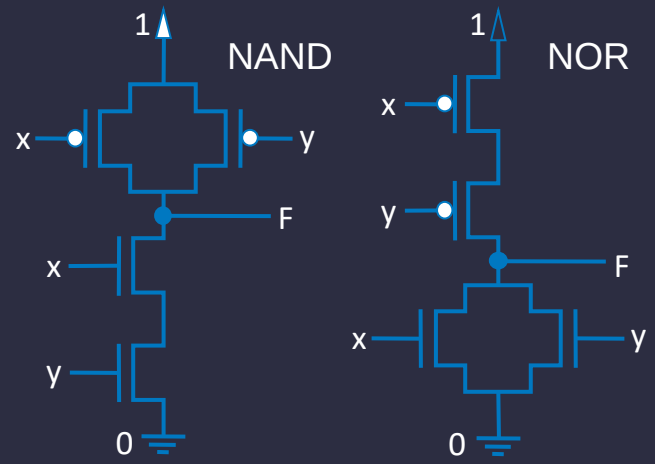


x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

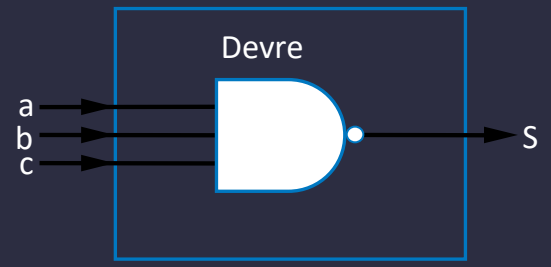
x	y	F
0	0	1
0	1	0
1	0	0
1	1	1



- NAND: NOT AND
- NOR: NOT OR
- XOR: Exclusive OR
- XNOR: NOT XOR

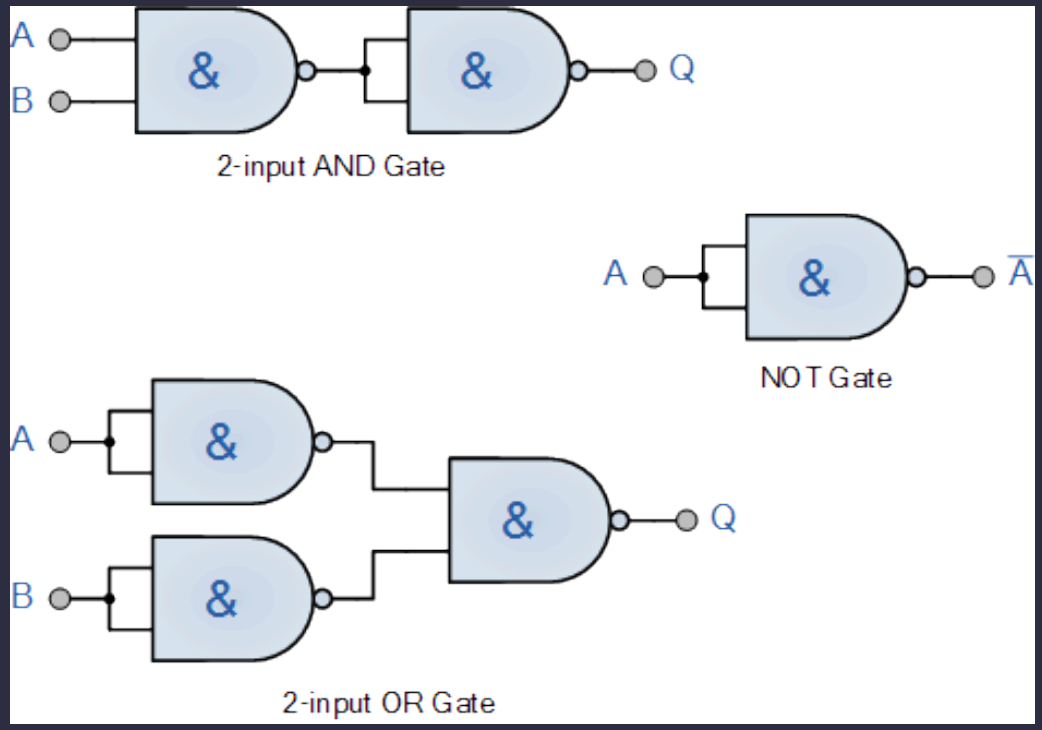
Mantık Kapıları

- Uçak lavabo kullanım lambası
  - S = (abc)’



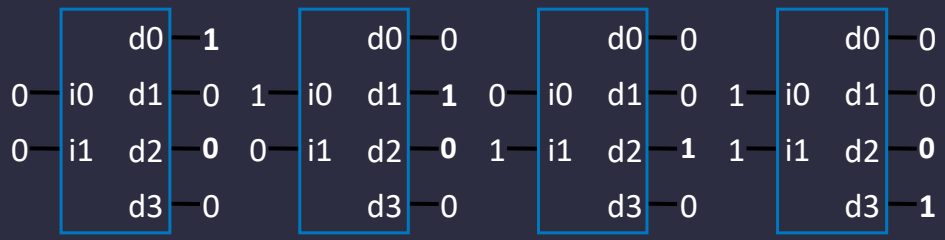
NAND Kapısı

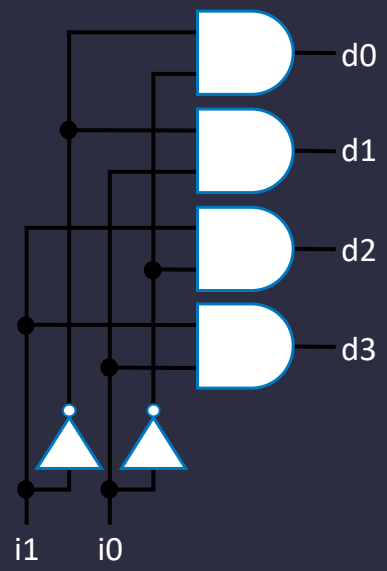
- Genellikle dijital tasarım için NAND kapısı çok kullanılır.
- Bunun nedeni, NAND Kapısı ile tüm devrelerin yapılabiliyor olmasıdır.
  - NOT
  - AND
  - OR

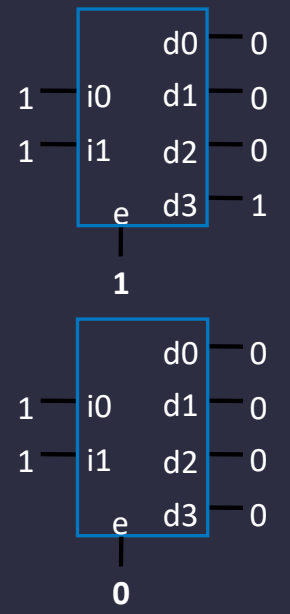


Decoder ve MUX

- **Decoder**: Çıkış pinlerinden giriş'teki aldığı sayıya karşılık gelen pini aktif eder.
- 2 girişli decoder: 4 olası giriş vardır.
  - 4 çıkışı bulunur
- Enable pin'li Decoder
  - Eğer e=0, tüm çıkışlar 0 olur
  - Eğer e=1, normal davranır
- N giriş decoder: 2^n çıkış

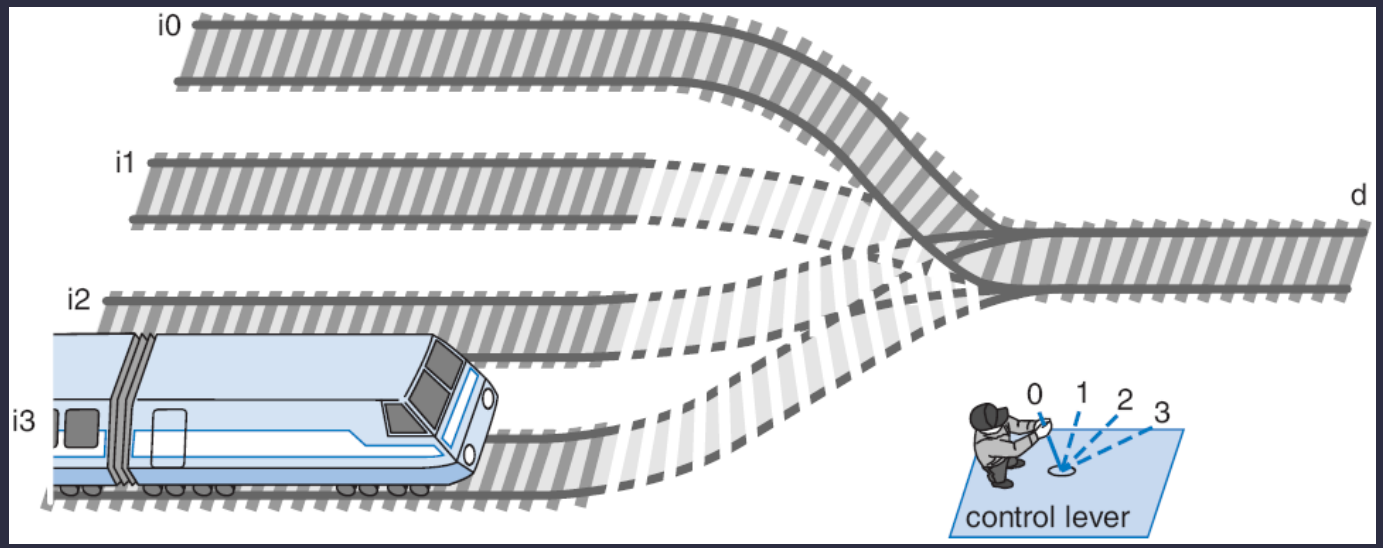




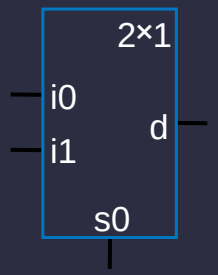


Multiplexer (Mux)

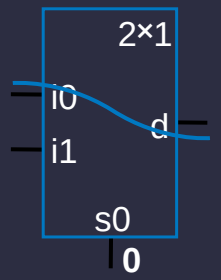
- Mux: Kombinasyonel bir devredir. Kendisine gelen girişleri, seçme bitine göre dışarı çıkartır.
  - 4 giriş mux → 2 seçme girişi
  - 8 giriş mux → 3 seçme girişi
  - N giriş → $\log_2(N)$ seçme girişi

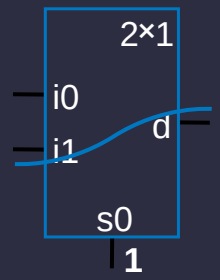


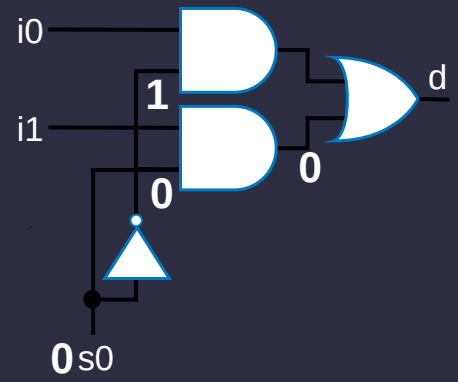
Mux İç Yapısı

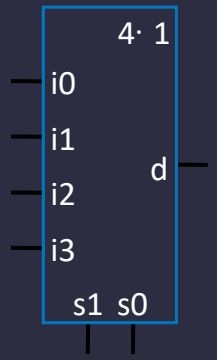


2x1 mux

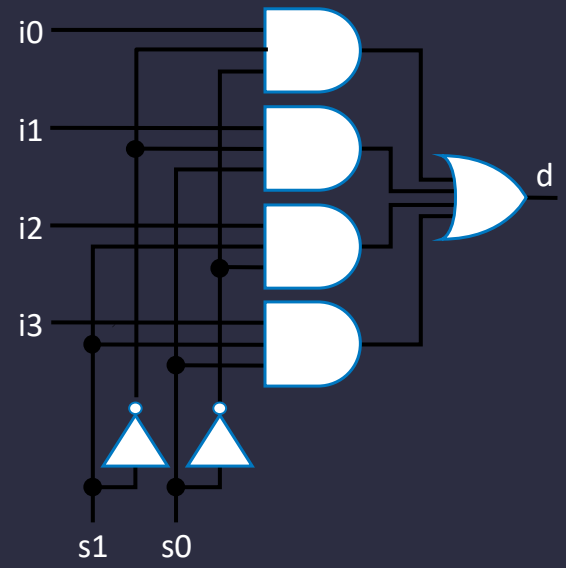




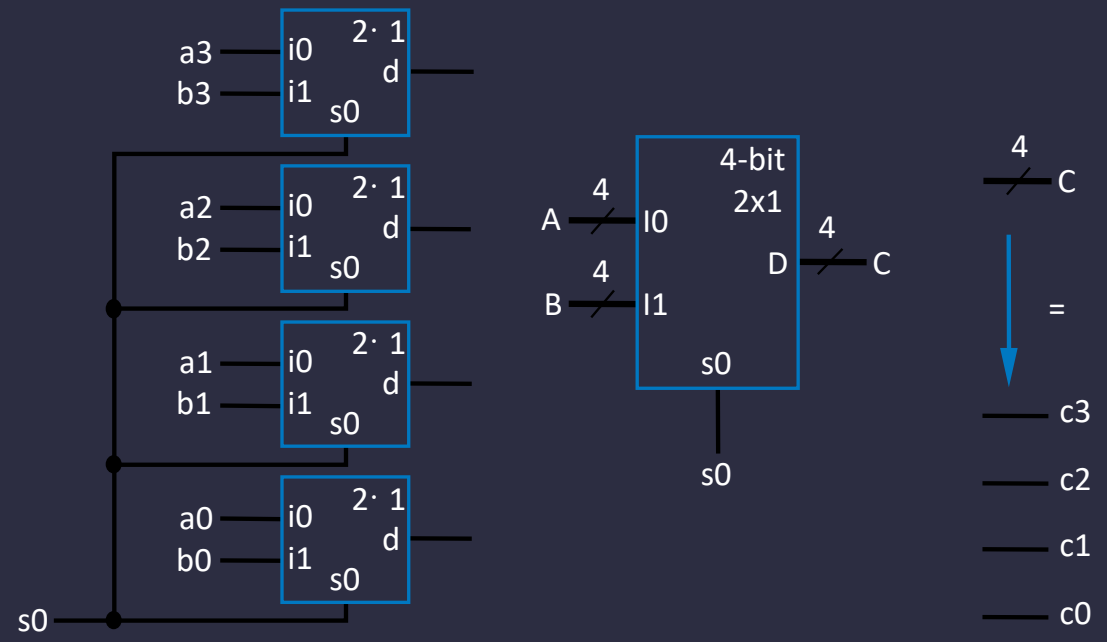




4x1 mux

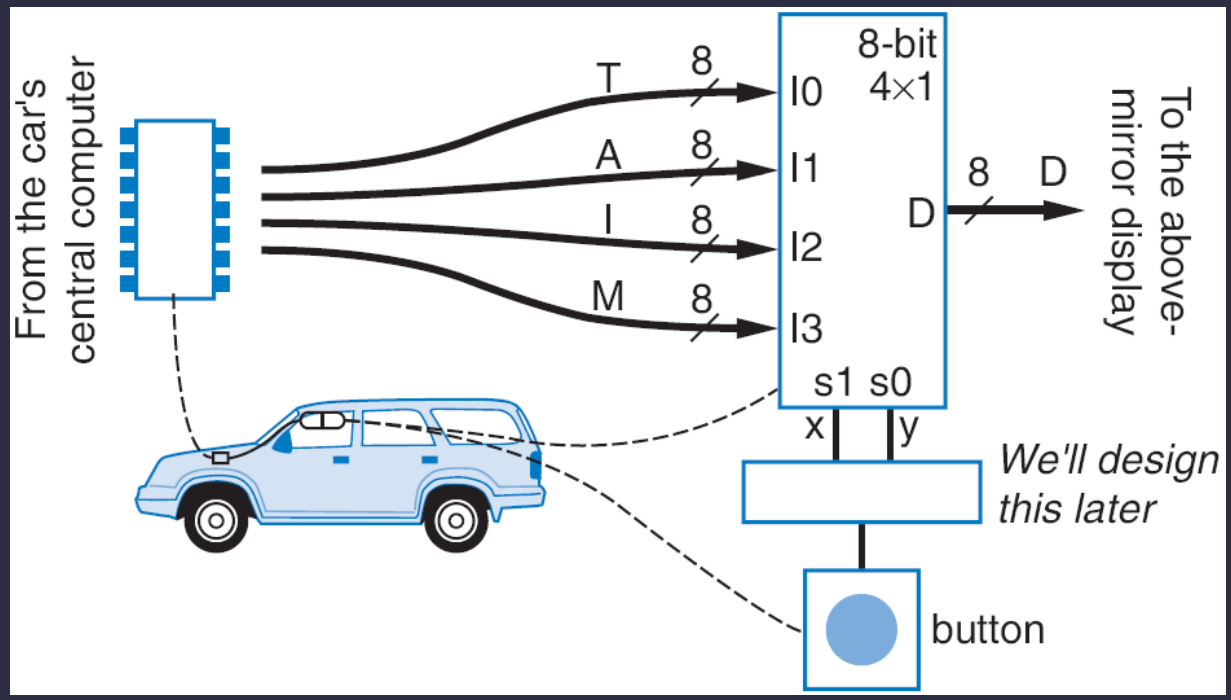


MUX Birleştirme



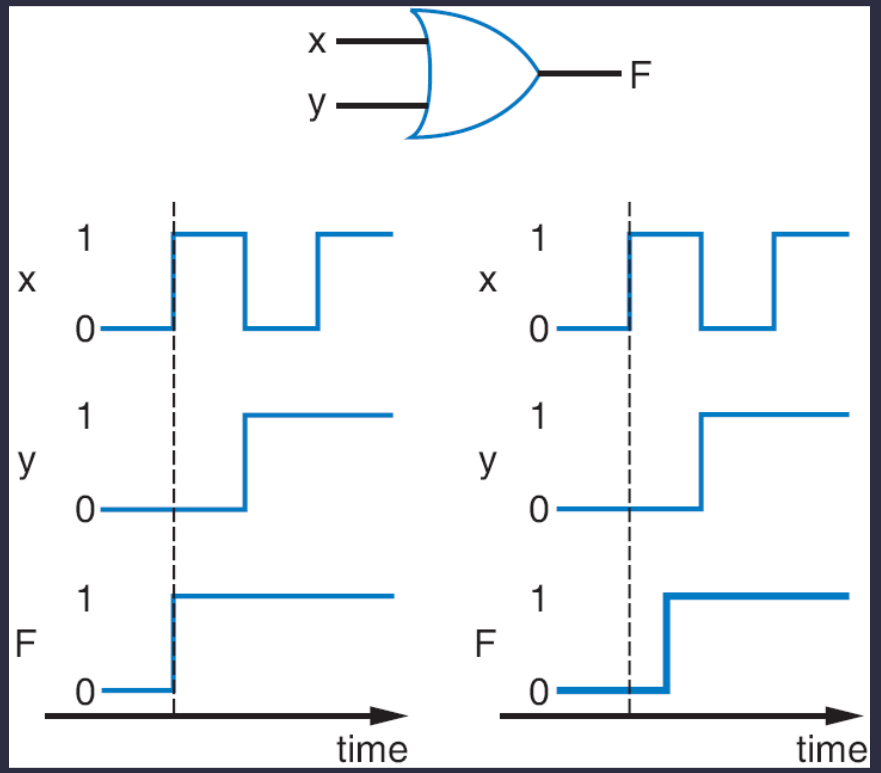
- Örnek: İki 4 bitlik giriş, A (a3 a2 a1 a0) ve B (b3 b2 b1 b0)
  - 4-bit 2x1 MUX 4 tane 1 bit, 2x1 MUX kullanarak yapılabilir

N-bit MUX Örneği



- 4 olası gösterilecek metin bulunmaktadır
  - Sıcaklık, Ortalama Yakıt Kullanımı, Ortalama Hız, Kalan KM – hepsi 8 bitlik genlişlikte olsun
  - Hangisinin ekranda gözükeceği x ve y bitleri ile seçiliyor
  - 8-bit 4x1 MUX kullanılabilir.

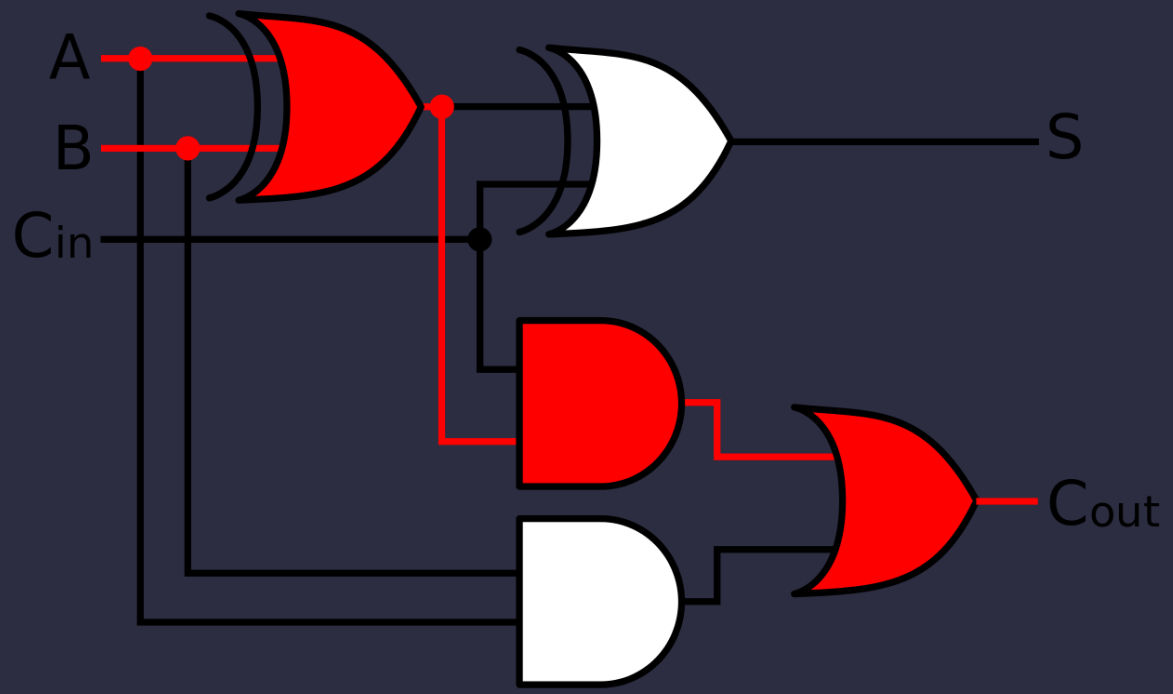
Kapı Gecikmeleri



- Tüm devrelerin gecikmesi vardır.
  - Çıkışlar anında değişmez

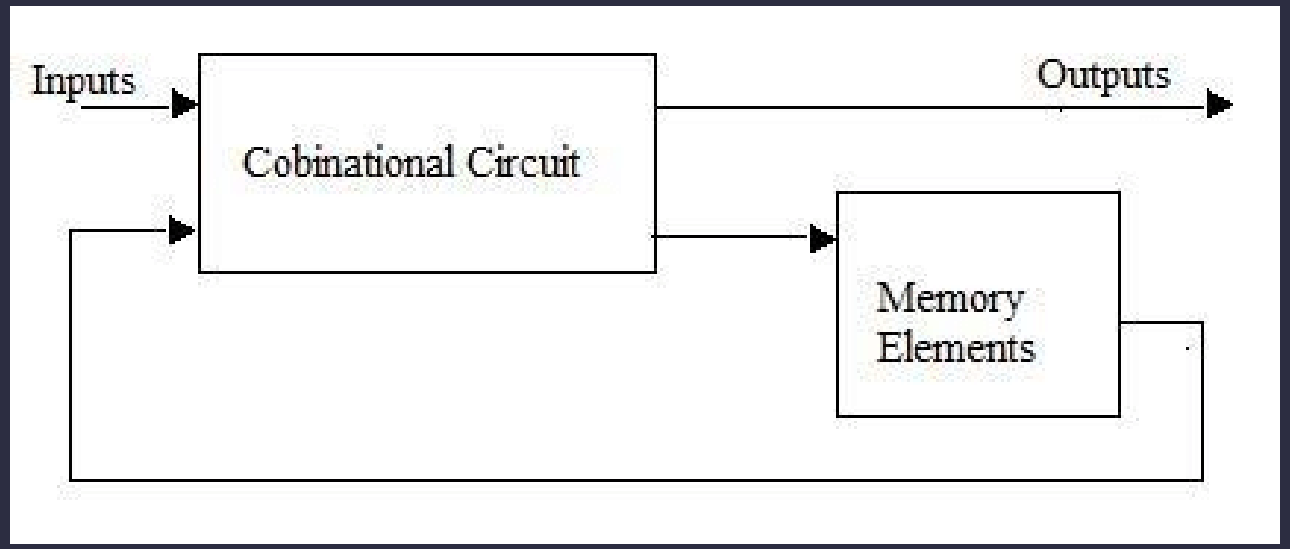
Mantık Kapılarından, Kontrol Ünitelerine

- *Kombinasyonel Devreler*
  - Devrenin çıktısı, o anki girişe bağlıdır.
  - Devrenin çıktı verme gecikmesi, devredeki en uzun yol'a bağlıdır.



Mantık Kapılarından, Kontrol Ünitelerine

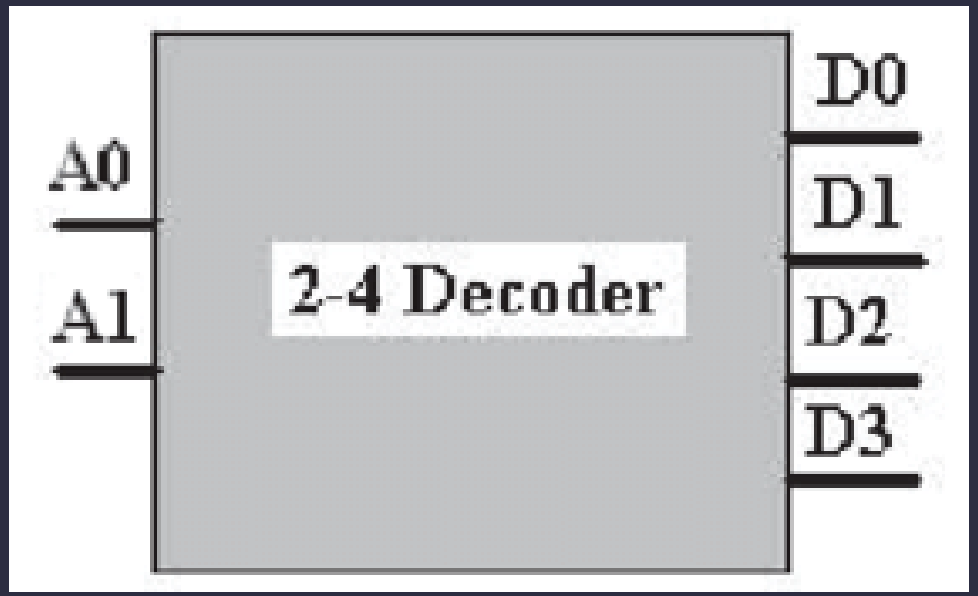
- *Ardışık Devreler*
  - Çıkış hem şu anki girişe ve bellekteki değerlere bağlıdır.
  - Devrenin bazı çıktıları bellekte saklanarak yeniden kullanılır.
  - Detaylarına önümüzdeki hafta gireceğiz.



Çözücü (Decoder)

- *n* giriş, 2^n çıkış
  - Her bir çıkış sadece tek bir durumda 1 olur

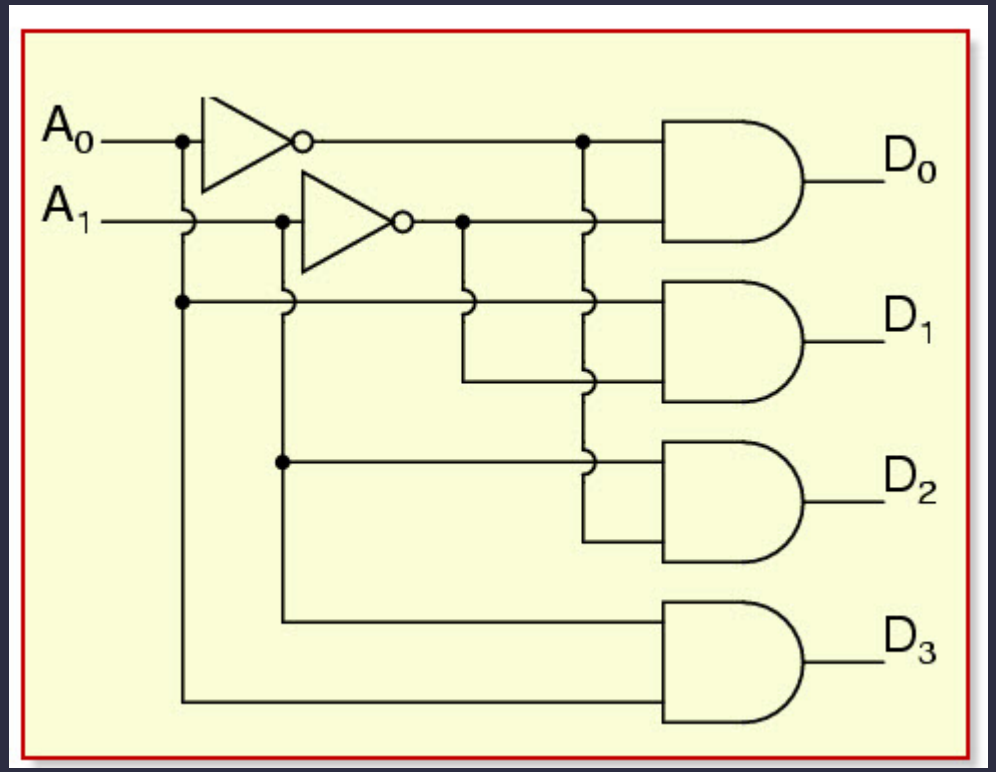
2-bit çözücü



Çözücü (Decoder)

- *n* giriş, 2^n çıkış
  - Her bir çıkış sadece tek bir durumda 1 olur

2-bit çözücü

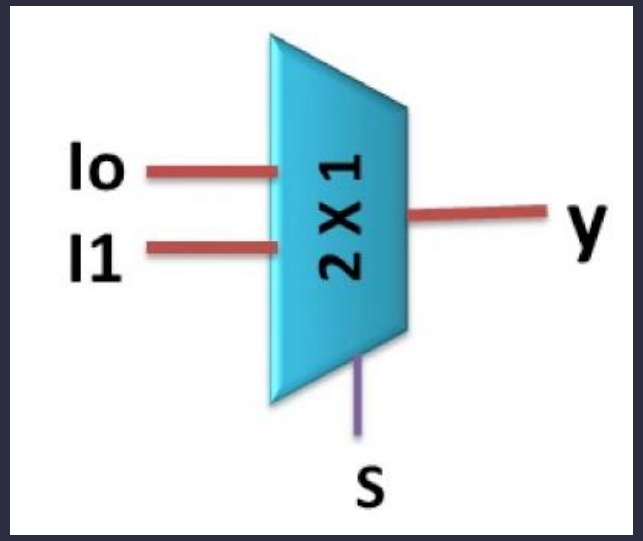


Truth Table

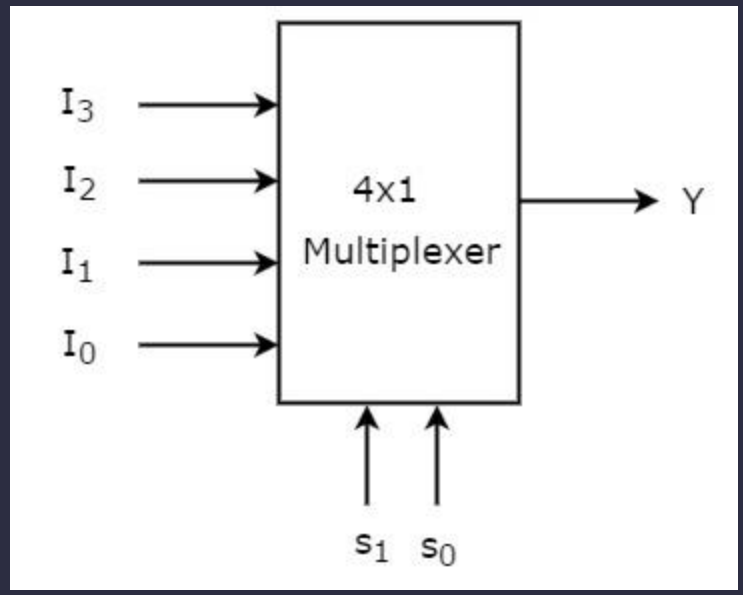
A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Seçici (Multiplexer - MUX)

- *n*-bit seçme, 2^n giriş ve tek çıkışı bulunmaktadır.
  - Seçme bitine göre, girişteki değer çıkışa aktarılmaktadır.



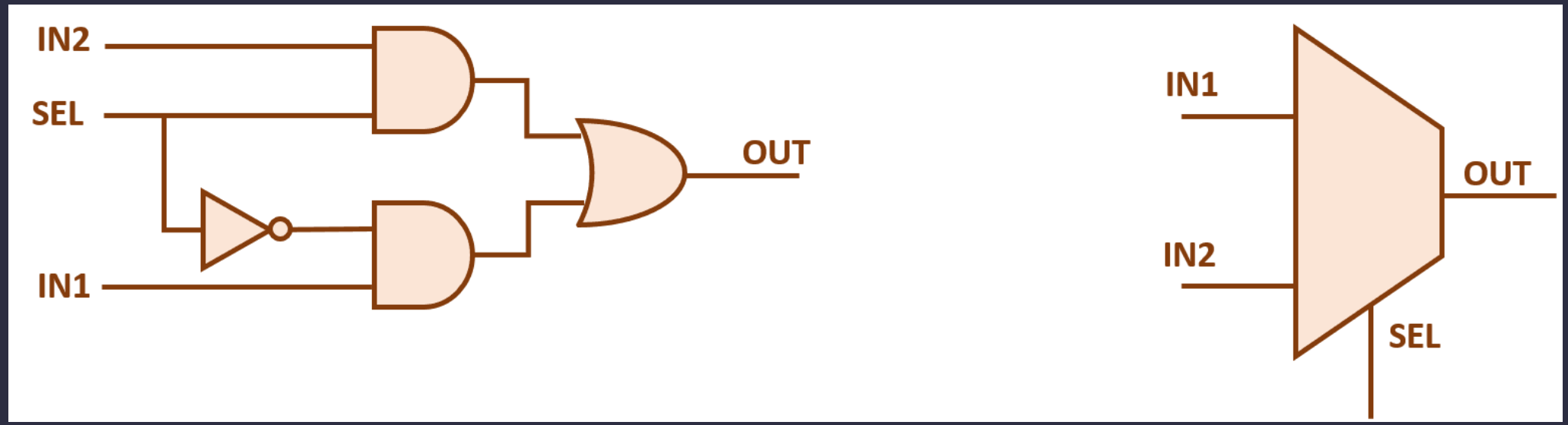
2-1 MUX



4-1 MUX

Seçici (Multiplexer - MUX)

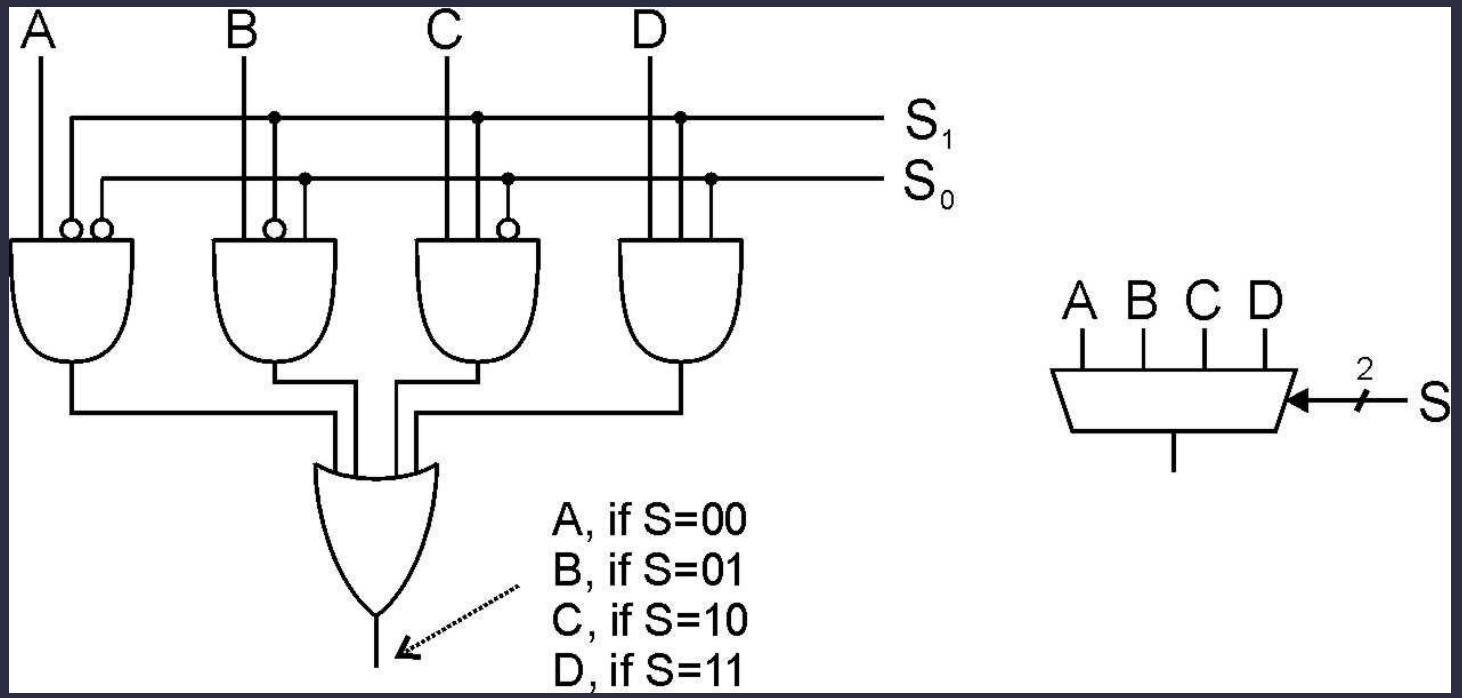
- *n*-bit seçme, 2^n giriş ve tek çıkışı bulunmaktadır.
  - Seçme bitine göre, girişteki değer çıkışa aktarılmaktadır.



2-1 MUX

Seçici (Multiplexer - MUX)

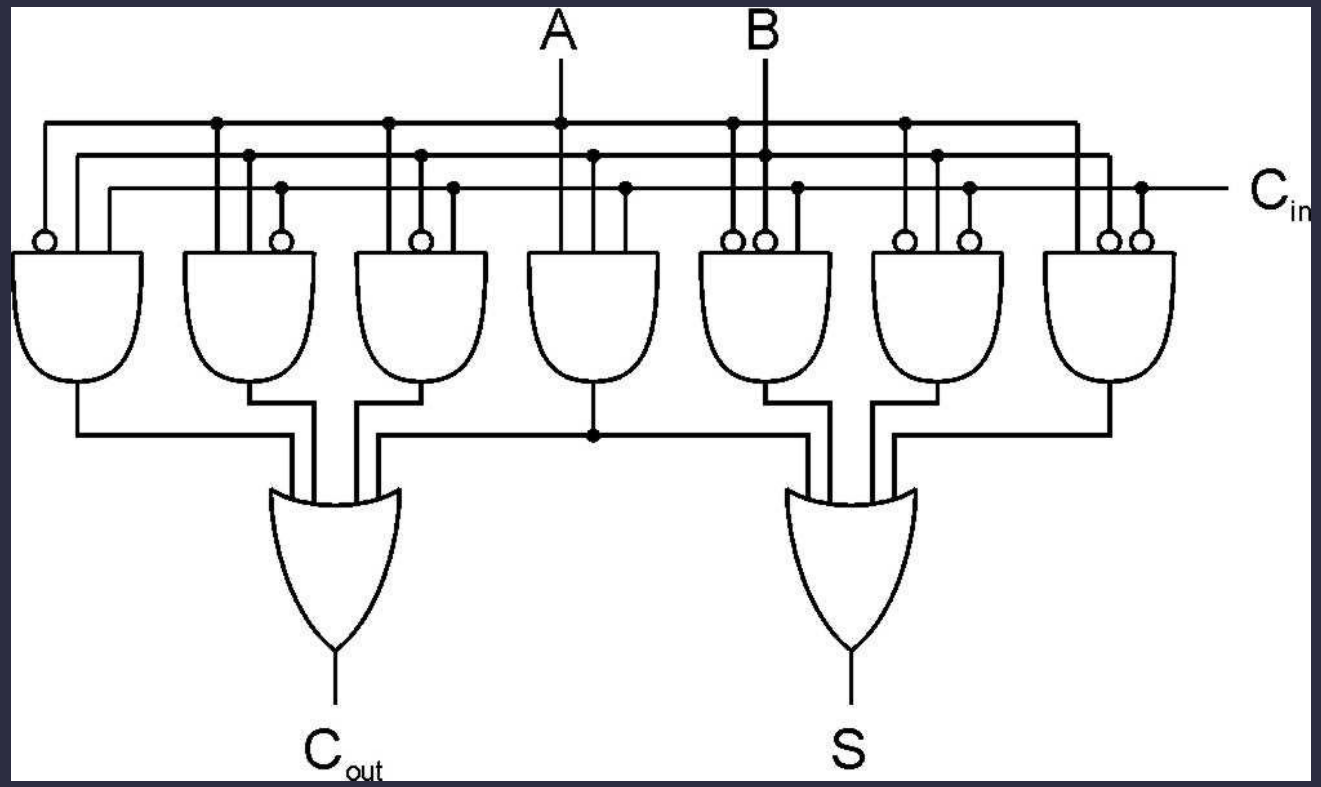
- *n*-bit seçme, 2^n giriş ve tek çıkışı bulunmaktadır.
  - Seçme bitine göre, girişteki değer çıkışa aktarılmaktadır.



4-1 MUX

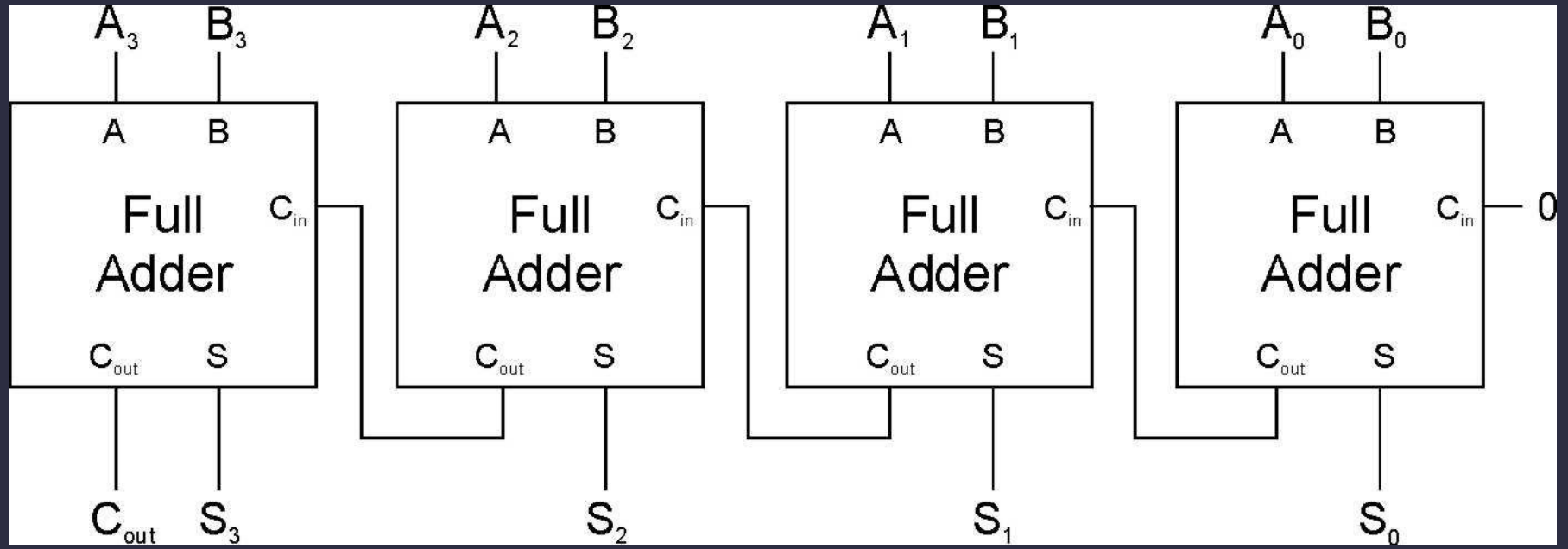
Tam Toplayıcı (Full Adder)

- İki bit (A ve B) ve bir elde girişi (Cin) alarak, bir bitlik toplam (S) ve elde (Cout) üretir.



A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

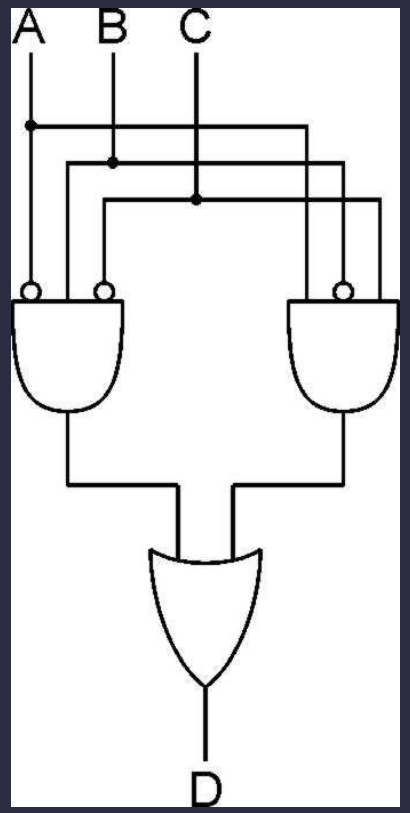
4 bitlik Toplayıcı



Diğer Devreler

- Herhangi bir devre, Ve, Veya ve Değil kapıları ile ifade edilebilir.

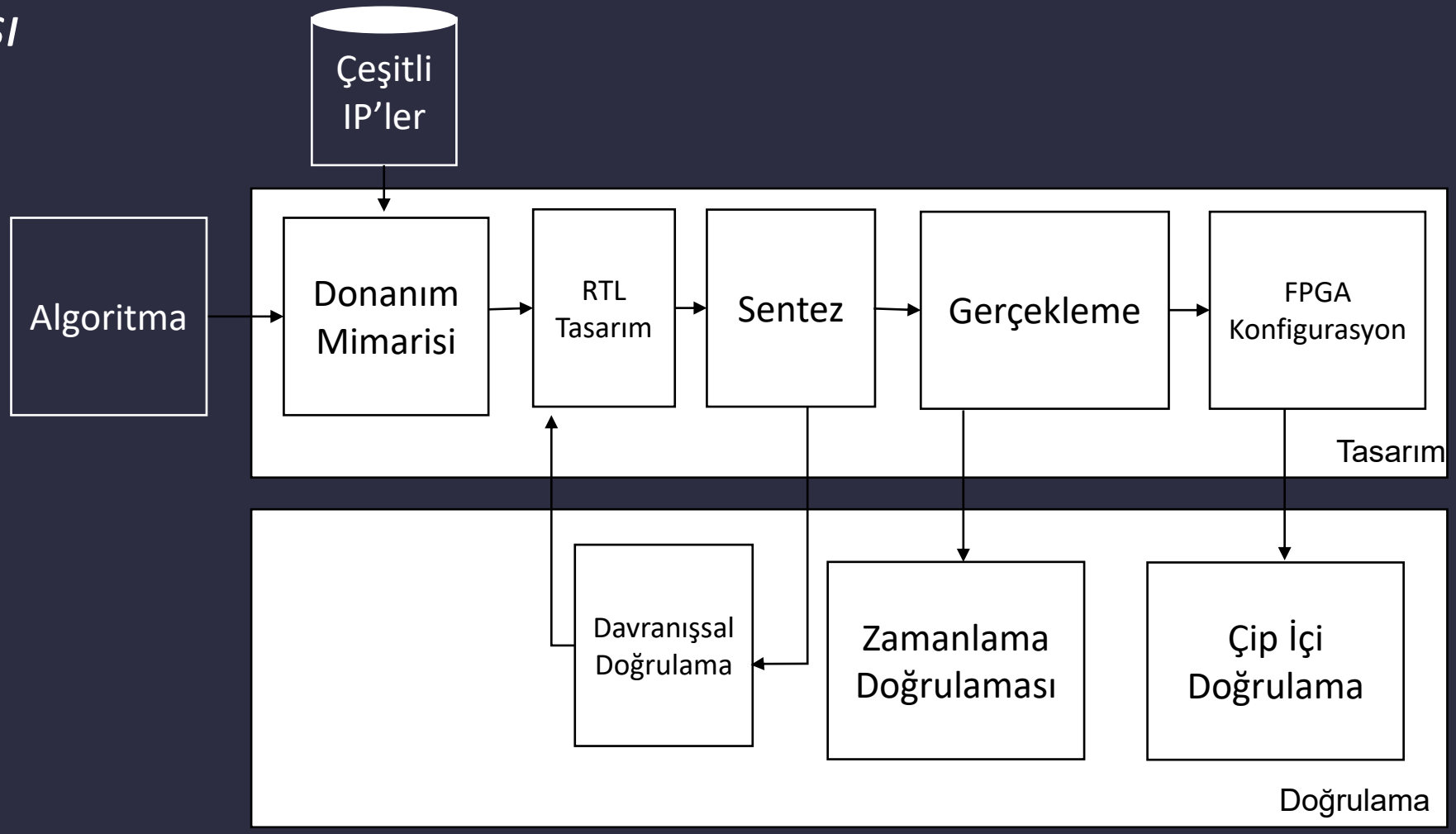
A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



1. Doğruluk tablosunda (truth table) 1 ile ifade edilen bölgeleri, Ve kapısı ile ifade edin.
2. Ve kapılarını veya kapısı ile birleştirin

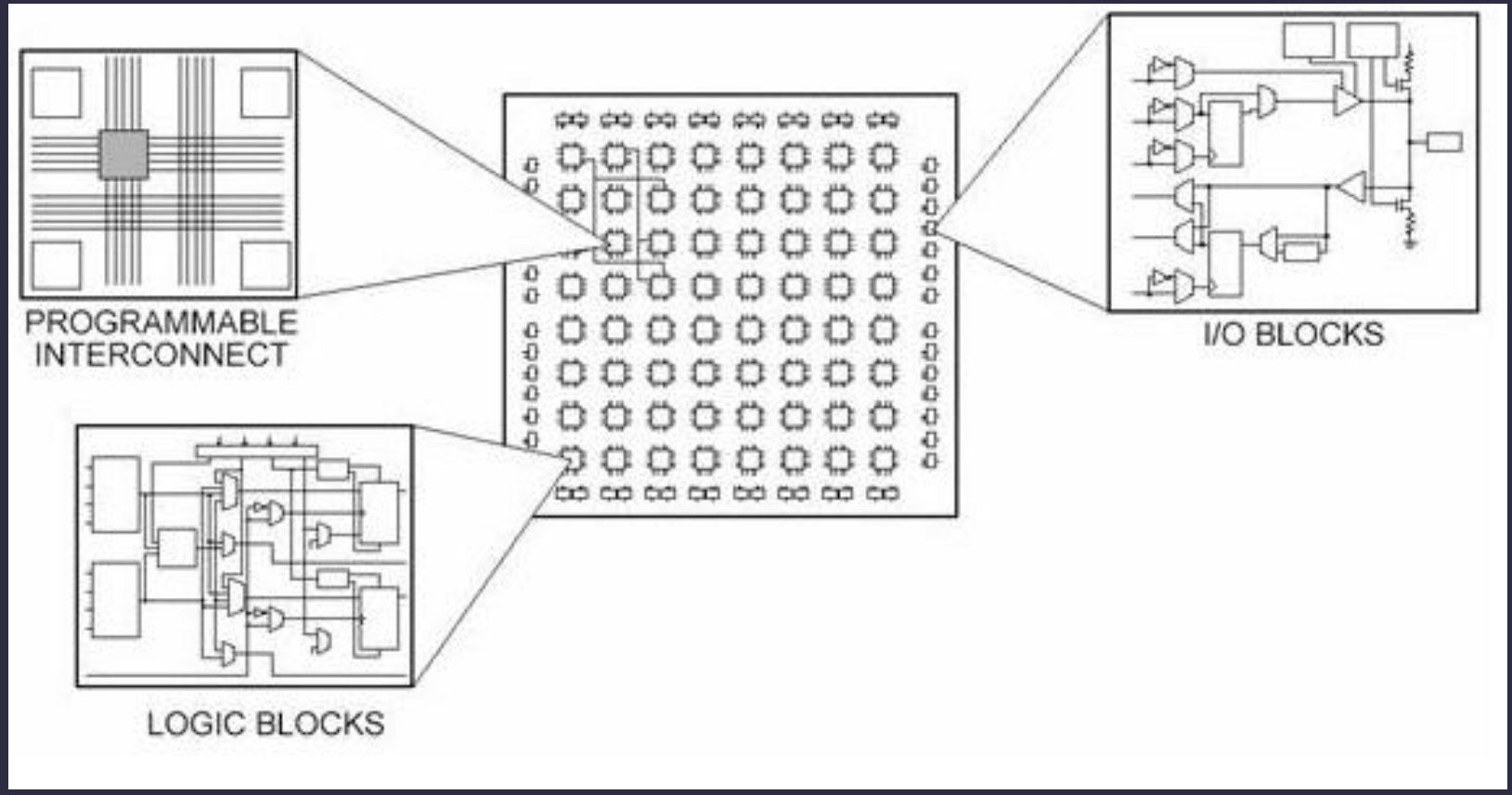
Vivado Tasarım Aracı

- *Tasarım Akışı*



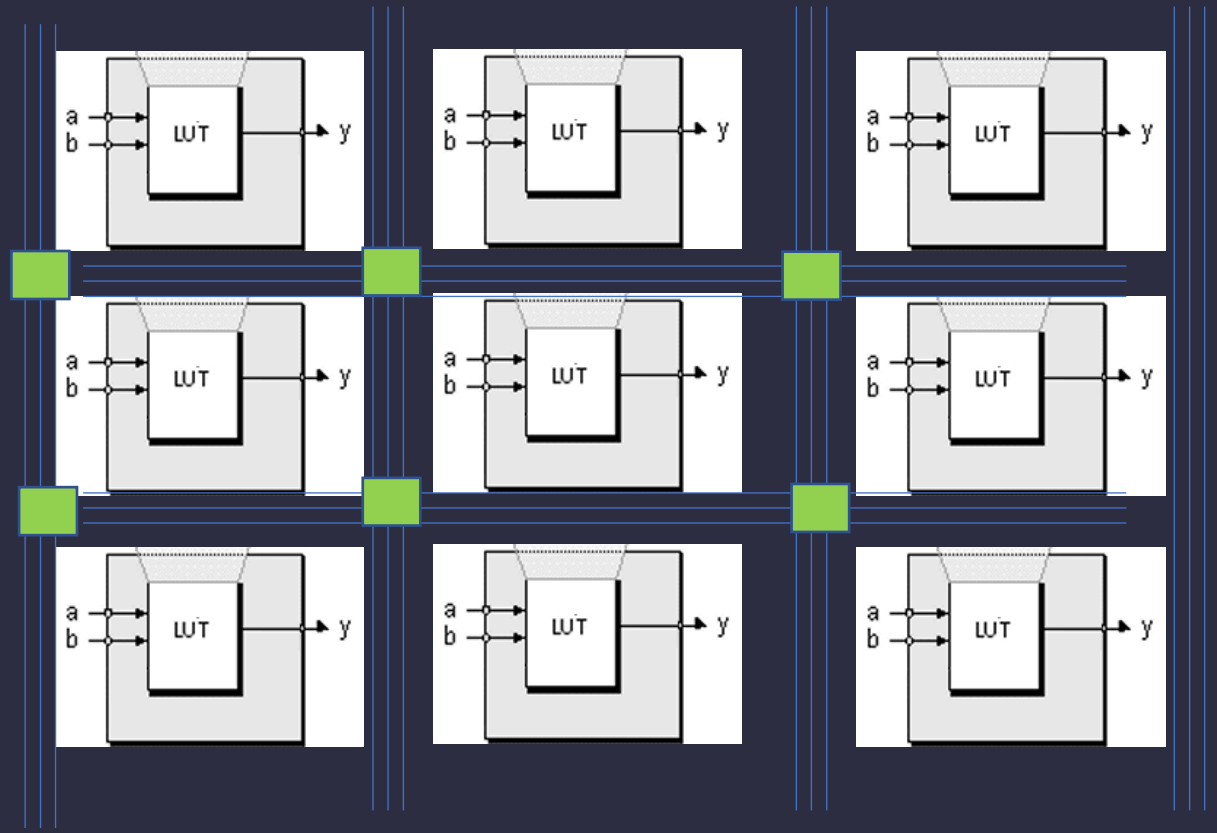
Verilog – Kombinasyonel Devreler

FPGA Nedir?



Verilog – Kombinasyonel Devreler

FPGA Nedir?



FPGA Nedir?

- FPGA (Field Programmable Gate Arrays) entegre bir çiptir.
- İçerisinde programlanabilir bloklar ve bu bloklar arasında konfigüre edilebilir bağlantılar bulunmaktadır.
- Bu bloklar ve bağlantılar programlanarak, istenilen devre FPGA'in içerisinde gerçeklenebilmektedir.

Örnek FPGA Çip'i

FPGA Nedir?

- Bazı FPGA'ler sadece tek seferlik programlanabilmektedir.
- Bu FPGA'lere one-time programmable (OTP) denmektedir.

FPGA Nedir?

- FPGA’in açılımındaki "Field Programmable" kısmı, alanda programlanabilir, yani hangi iş için kullanılacak ise, o alan ile alakalı mantıksal tasarımın yüklenebileceğini ifade etmektedir.

- Bunun anlamı, FPGA çipleri fabrikada üretildikten sonra, sonradan istenilen tasarımın içerisine atılabilmesidir.

Neden FPGA önemlidir?

Çok çeşitli entegre devreler (IC – Integrated Circuits) bulunmaktadır.

- Bellek
- Mikroprosesörler
- Programlanabilir mantık cihazları (Programmable logic devices)
  - SPLD (Simple)
  - CPLD (Complex)
- Uygulama spesifik entegre devreler (ASIC – Application specific integrated circutis)
- Ve FPGA'ler …

PLD'ler

- İç mimarisi PLD üreticisi tarafından önceden belirlenmiştir.
- FPGA'ler gibi alanda programlanabilir yapılardır
- Ancak FPGA'lere kıyasla çok daha basit ve küçüktürler.
- Küçük olması sebebiyle FPGA'lerde kurulabilen büyük devreler, PLD'lerde kurulması daha zordur.

ASIC'ler

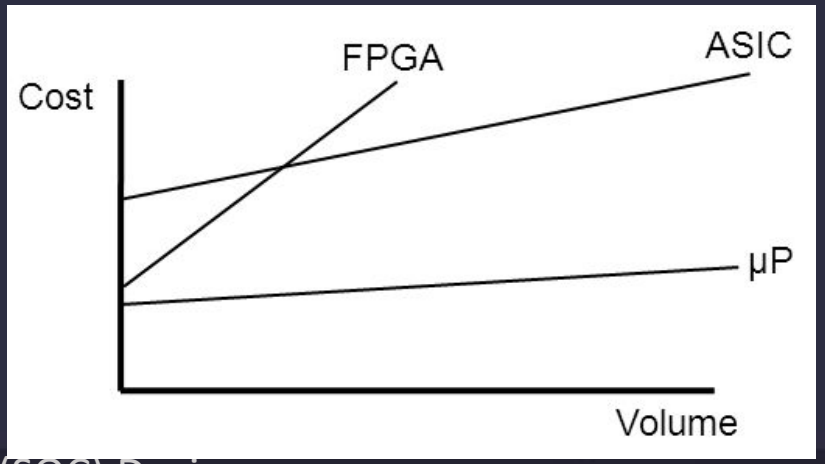
- Bu yapılar yüz milyonlarca mantık kapısını içerebilmektedirler.
- Devasa ve karmaşıklığı yüksek devrelerin gerçeklemesinde faydalıdırlar.
- ASIC'ler sadece belirli bir işlemi yapmak üzere üretilmiş entegre devrelerdir.
- ASIC'ler çok yüksek performans (yüksek frekanslarda çalışma) getirsede, ASIC tasarımı karmaşıklığı ve tasarım zamanı ve maliyetleri oldukça yüksektir.
- Ve üretilen çipler modifiye edilemez. Çipte bir hata olması durumunda üretilmiş bütün çipler çöpe atılacaktır.

FPGA'ler

- Bu nedenle FPGA'ler PLD ve ASIC'ler arasındaki boşluğu doldurmaktadır.
- FPGA'Ler PLD'ler gibi programlanabilir yapıdadır.
- Ancak PLD'lere göre çok daha fazla programlanabilir alanı vardır.
- ASIC'lerin aksine; ortaya çıkan tasarımda bir hata olması durumunda, tasarım tekrar tekrar modifiye edilerek test edilebilir.

FPGA'ler

- FPGA'ler ASIC'lere göre çok daha ucuzdurlar. (ASIC'ler ancak milyonlarca adet üretildiğinde birim maliyeti ucuz olmaktadır)
- FPGA'ler üzerinde tasarım geliştirmek ASIC bir tasarım yaratmaya göre çok daha kolaydır.
- Tasarım süresinin az olması nedeniyle, bir ürün üretileceğinde, piyasaya çıkma süresi daha kısadır.
- Tasarım ekipleri ASIC'te yatırılması gereken NRE (Non recurring engineering) yatırımını yapmadan, FPGA üzerinde tasarımlarını geliştirmeleri mümkün olmaktadır.



FPGA'in ağırlıklı olarak kullanıldığı sektörler ve uygulama alanları

- Telekomünikasyon
- Ağ haberleşmesi
- Otomotiv
- Sağlık
- Çeşitli endüstriyel uygulamalar
- ASIC tasarımlarının prototipleri

- DSP (Digital signal processor) uygulamaları
- SoC (System on Chip), tüm gerekli elektronik sistemin bir araya toplandığı tek bir IC

FPGA'lerin girdiği pazarlar

- ASIC: Bir çok ASIC uygulaması artık FPGA'ler ile yapılmaktadır.
- DSP: Günümüz FPGA'leri ile sinyal işleme uygulamaları, DSP cihazlarına göre çok daha yüksek hızlarda yapılabilmektedir.
- Mikrokontrolörler: Basit kontrol fonksiyonları için kullanılan mekanizmalardır. FPGA fiyatlarının düşmesi ile bu basit görevleri görmesi için FPGA'in içerisine soft işlemciler gömülerek aynı mekanizmayı görmesi sağlanmaktadır.

FPGA Kullanım Nedenleri

- Hesaplama Gücü
- Nanosaniye mertebesinde giriş alma ve çıktı verebilme

FPGA Kullanım Nedenleri

İşlemciler

```
c[0]=a[0]+b[0];
c[1]=a[1]+b[1];
c[2]=a[2]+b[2];
c[3]=a[3]+b[3];
...
c[1000]=a[1000]+b[1000];
```

FPGA

```
c[0]<=a[0]+b[0];
c[1]<=a[1]+b[1];
c[2]<=a[2]+b[2];
c[3]<=a[3]+b[3];
...
c[1000]<=a[1000]+b[1000];
```

FPGA Kullanım Nedenleri

İşlemciler

```
c[0]=a[0]+b[0];
c[1]=a[1]+b[1];
c[2]=a[2]+b[2];
c[3]=a[3]+b[3];
...
c[1000]=a[1000]+b[1000];
```

FPGA

```
c[0]<=a[0]+b[0];
c[1]<=a[1]+b[1];
c[2]<=a[2]+b[2];
c[3]<=a[3]+b[3];
...
c[1000]<=a[1000]+b[1000];
```

FPGA Kullanım Nedenleri

İşlemciler	FPGA
c[0]=a[0]+b[0];	c[0]<=a[0]+b[0];
c[1]=a[1]+b[1];	c[1]<=a[1]+b[1];
c[2]=a[2]+b[2];	c[2]<=a[2]+b[2];
c[3]=a[3]+b[3];	c[3]<=a[3]+b[3];
…	…
c[1000]=a[1000]+b[1000];	c[1000]<=a[1000]+b[1000];

FPGA Kullanım Nedenleri

İşlemciler	FPGA
c[0]=a[0]+b[0];	c[0]<=a[0]+b[0];
c[1]=a[1]+b[1];	c[1]<=a[1]+b[1];
c[2]=a[2]+b[2]; ←	c[2]<=a[2]+b[2];
c[3]=a[3]+b[3];	c[3]<=a[3]+b[3];
…	…
c[1000]=a[1000]+b[1000];	c[1000]<=a[1000]+b[1000];

FPGA Kullanım Nedenleri

İşlemciler	FPGA
c[0]=a[0]+b[0];	c[0]<=a[0]+b[0];
c[1]=a[1]+b[1];	c[1]<=a[1]+b[1];
c[2]=a[2]+b[2];	c[2]<=a[2]+b[2];
c[3]=a[3]+b[3]; ←	c[3]<=a[3]+b[3];
…	…
c[1000]=a[1000]+b[1000];	c[1000]<=a[1000]+b[1000];

FPGA Kullanım Nedenleri

İşlemciler

```
c[0]=a[0]+b[0];
c[1]=a[1]+b[1];
c[2]=a[2]+b[2];
c[3]=a[3]+b[3];
...
c[1000]=a[1000]+b[1000];
```

FPGA

```
c[0]<=a[0]+b[0];
c[1]<=a[1]+b[1];
c[2]<=a[2]+b[2];
c[3]<=a[3]+b[3];
...
c[1000]<=a[1000]+b[1000];
```

FPGA Kullanım Nedenleri

İşlemciler

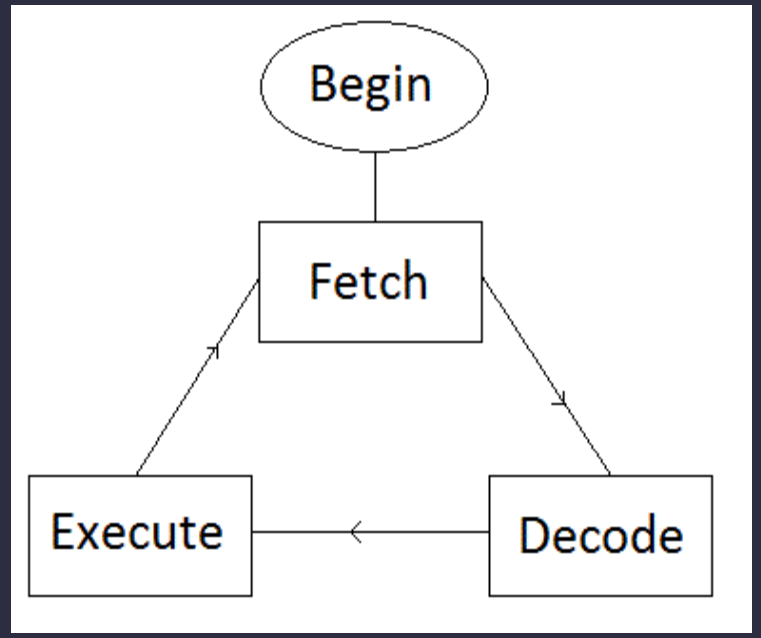
```
c[0]=a[0]+b[0];
c[1]=a[1]+b[1];
c[2]=a[2]+b[2];
c[3]=a[3]+b[3];
...
c[1000]=a[1000]+b[1000];
```

FPGA

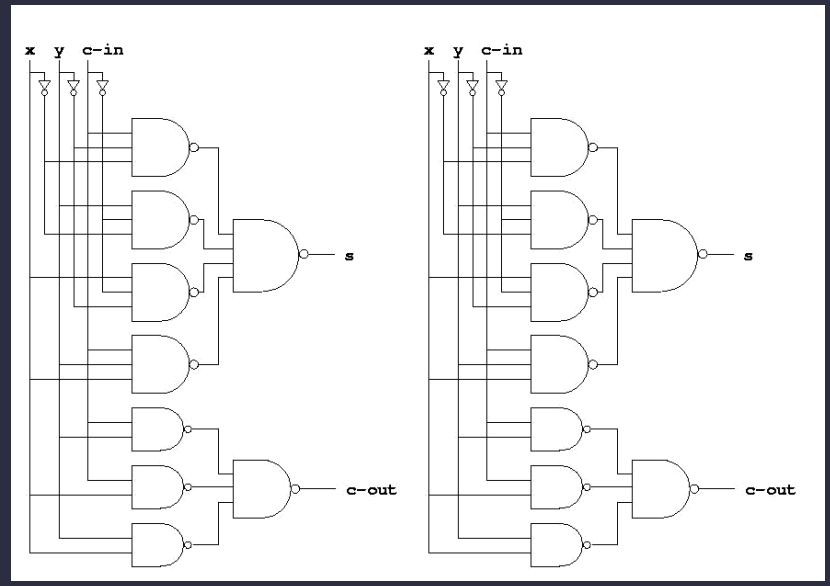
```
c[0]<=a[0]+b[0];
c[1]<=a[1]+b[1];
c[2]<=a[2]+b[2];
c[3]<=a[3]+b[3];
...
c[1000]<=a[1000]+b[1000];
```

FPGA Kullanım Nedenleri

İşlemciler



FPGA



FPGA Kullanım Nedenleri

CPU

1000 adet işlem için(her işlemin 1 clock cycle sürdüğü varsayılmıştır),

Ortalama Mikro işlemci hızı: 3 ghz için,
İşletim sistemi yok düşünüldüğünde

Gerekli zaman = İşlem sayısı X (1/Frekans)

= 1000 X 1/3milyar
= 333 nano saniye

FPGA

1000 adet işlem için(her işlemin 1 clock cycle sürdüğü varsayılmıştır),

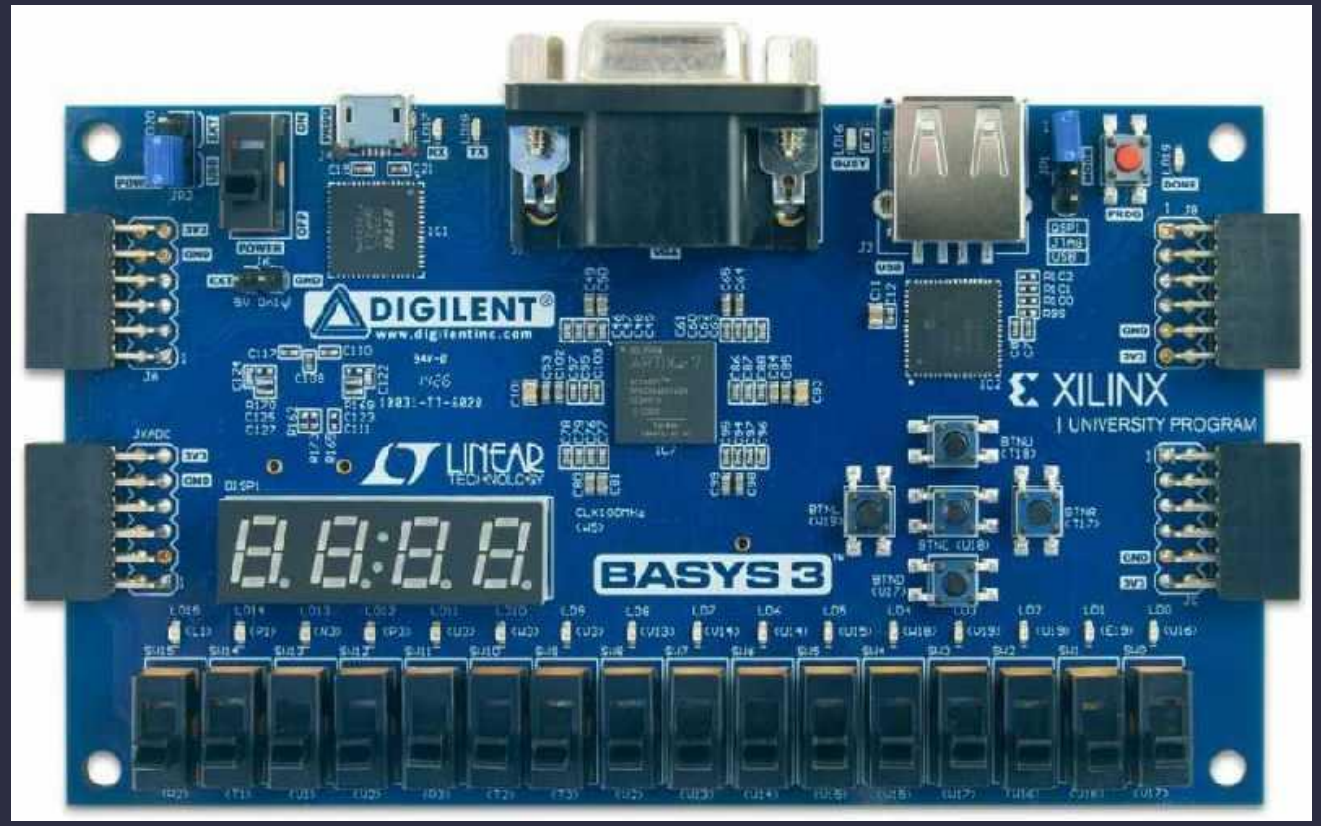
Ortalama fpga frekansı: 100 mhz için

Gerekli zaman = 1 x (1/Frekans)

= 1 X 1/100m
= 10 nano saniye

Verilog – Kombinasyonel Devreler

LAB'larda Xilinx Artix 7 FPGA Barındıran Basys3 FPGA'leri kullanılacaktır.



Verilog – Kombinasyonel Devreler

Xilinx'in FPGA'lerini programlamak için Vivado isimli bir geliştirme ortamı bulunmaktadır.

Vivado yazılımı ile FPGA tasarımları yapılmaktadır.

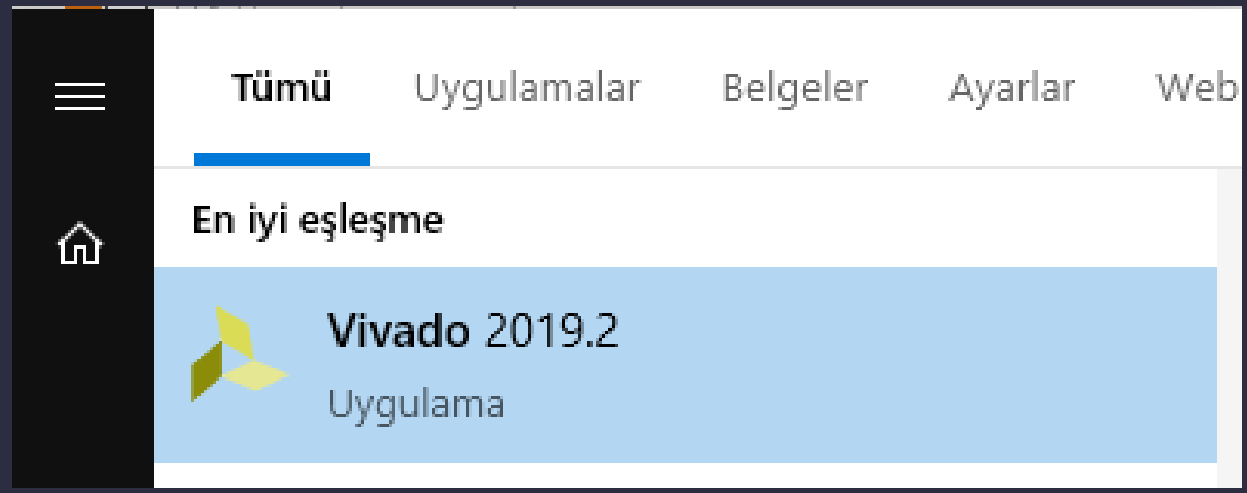
Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

- İndirme Adresi: https://www.xilinx.com/support/download.html
- Kurulum ve Lisanslama Video'su: https://www.youtube.com/watch?v=yW1bJbXnbRU

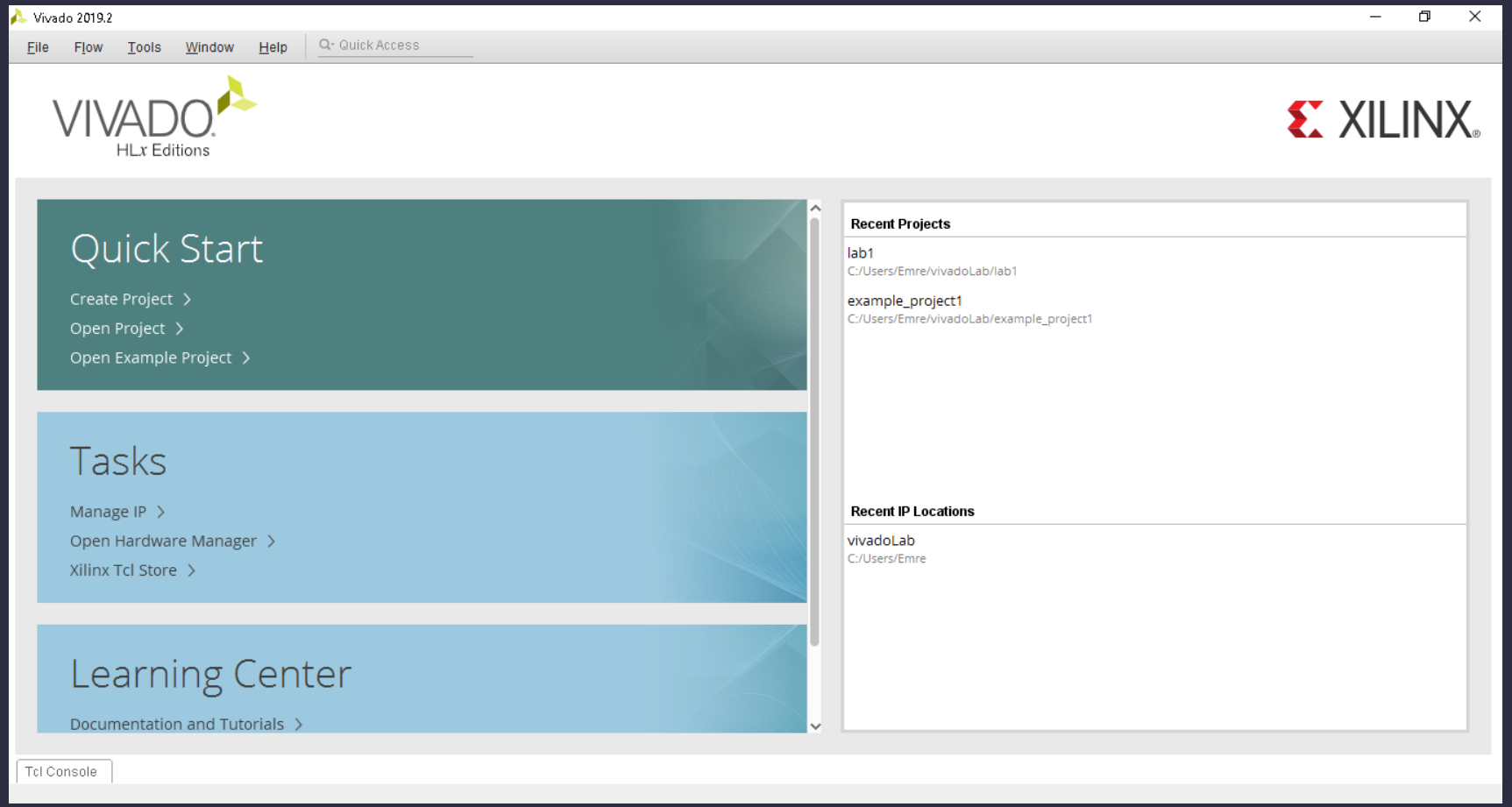
Vivado Tasarım Aracı

Vivado Tasarım Aracı, Xilinx'in 7 ve daha yeni jenarasyon FPGA'leri için kullanılabilen bir geliştirme ortamıdır. Bu ortam Xilinx'in sunduğu çeşitli geliştirme ve doğrulama araçlarını barındırır. Başlat'ın içerisinde 'Vivado' kelimesi ile arama yaparak uygulamanın kısa yoluna erişilebilir.



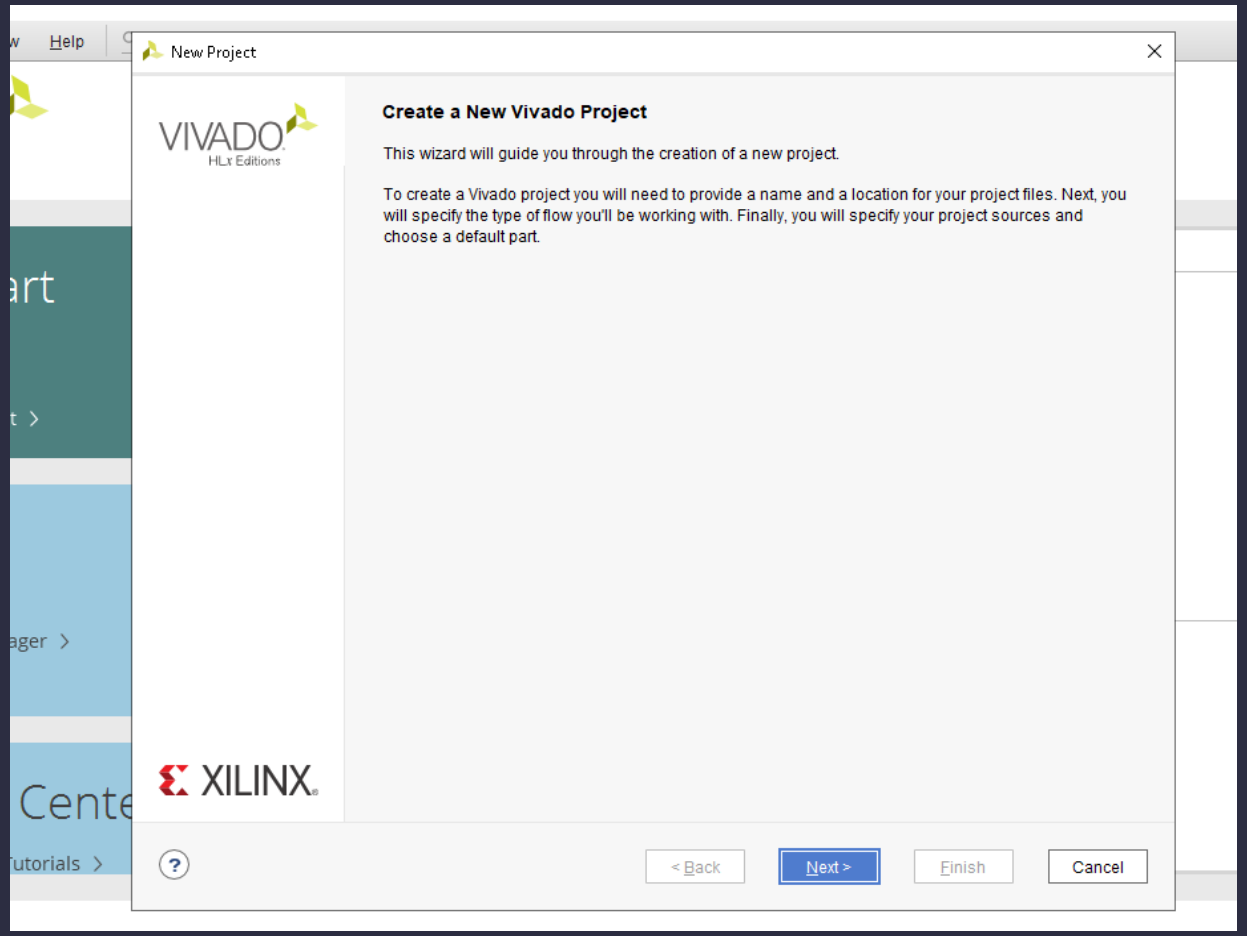
Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı



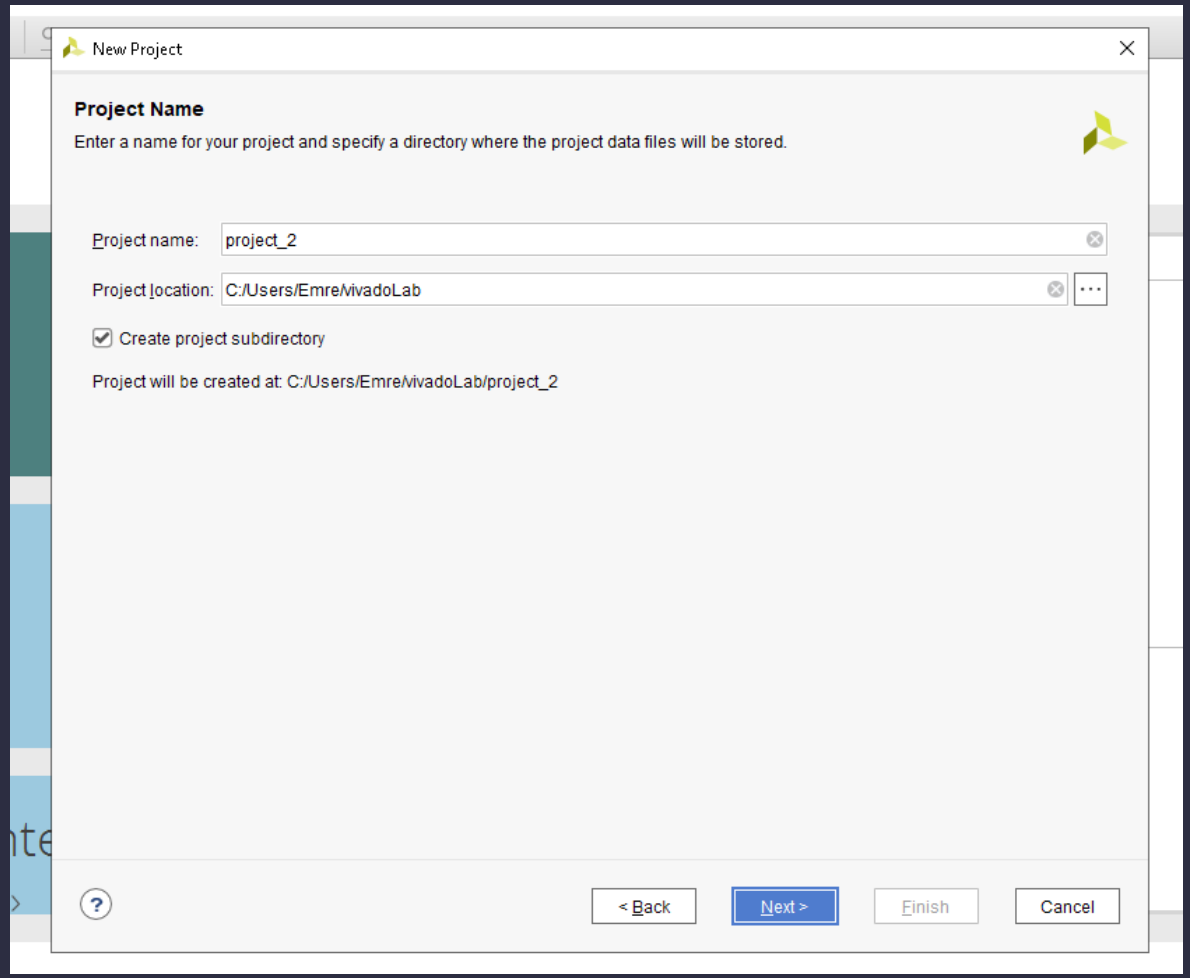
Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı



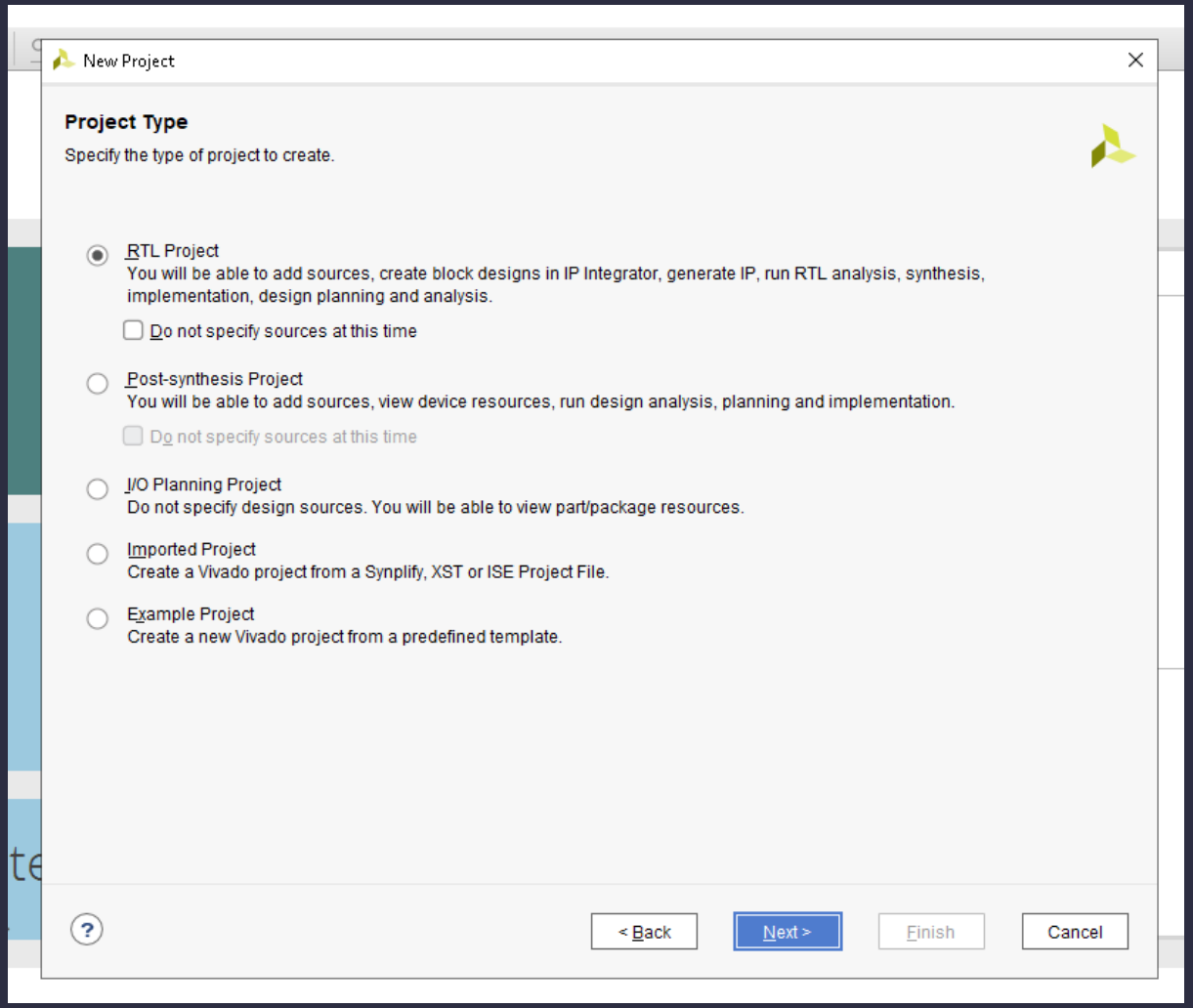
Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı



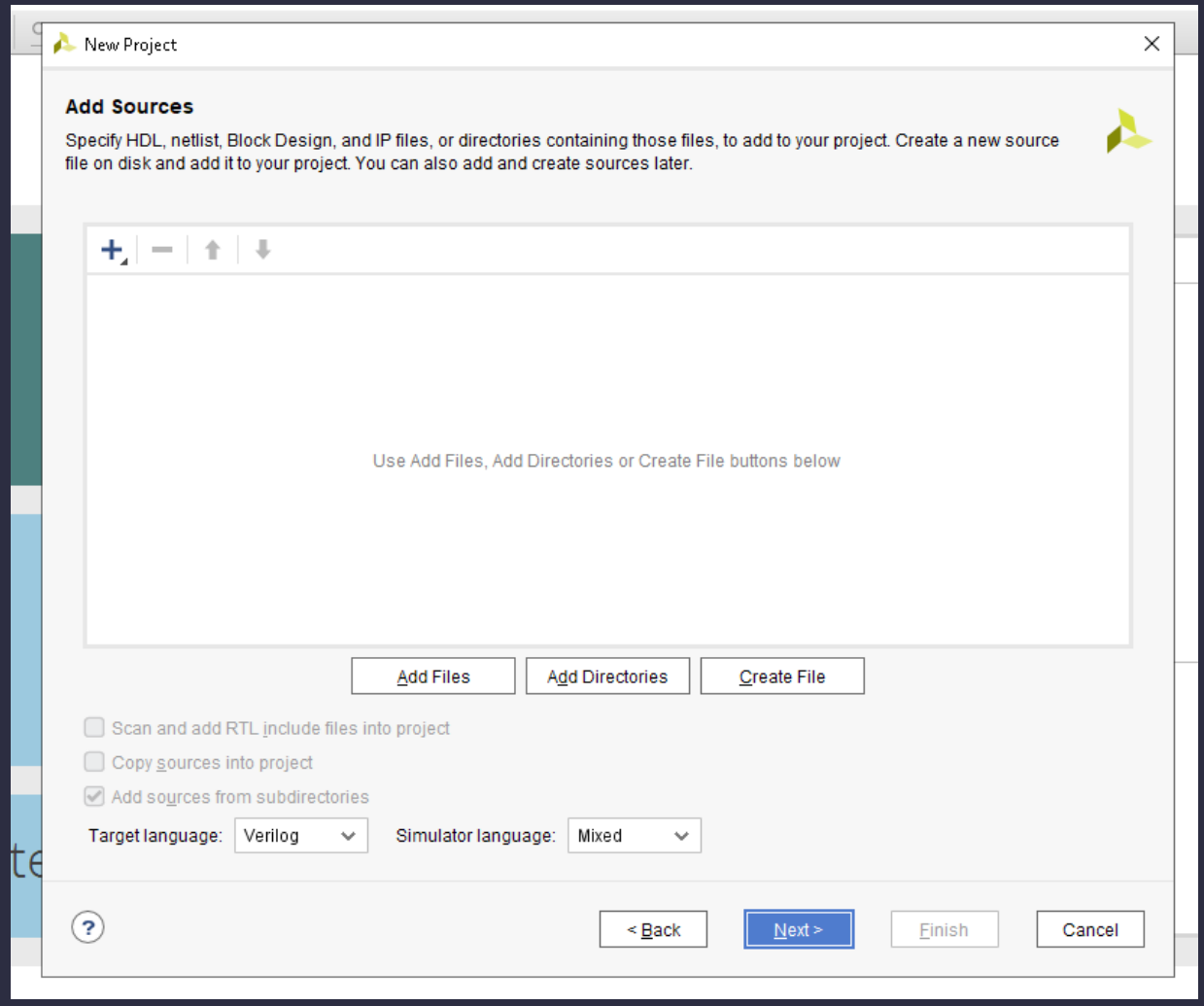
Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı



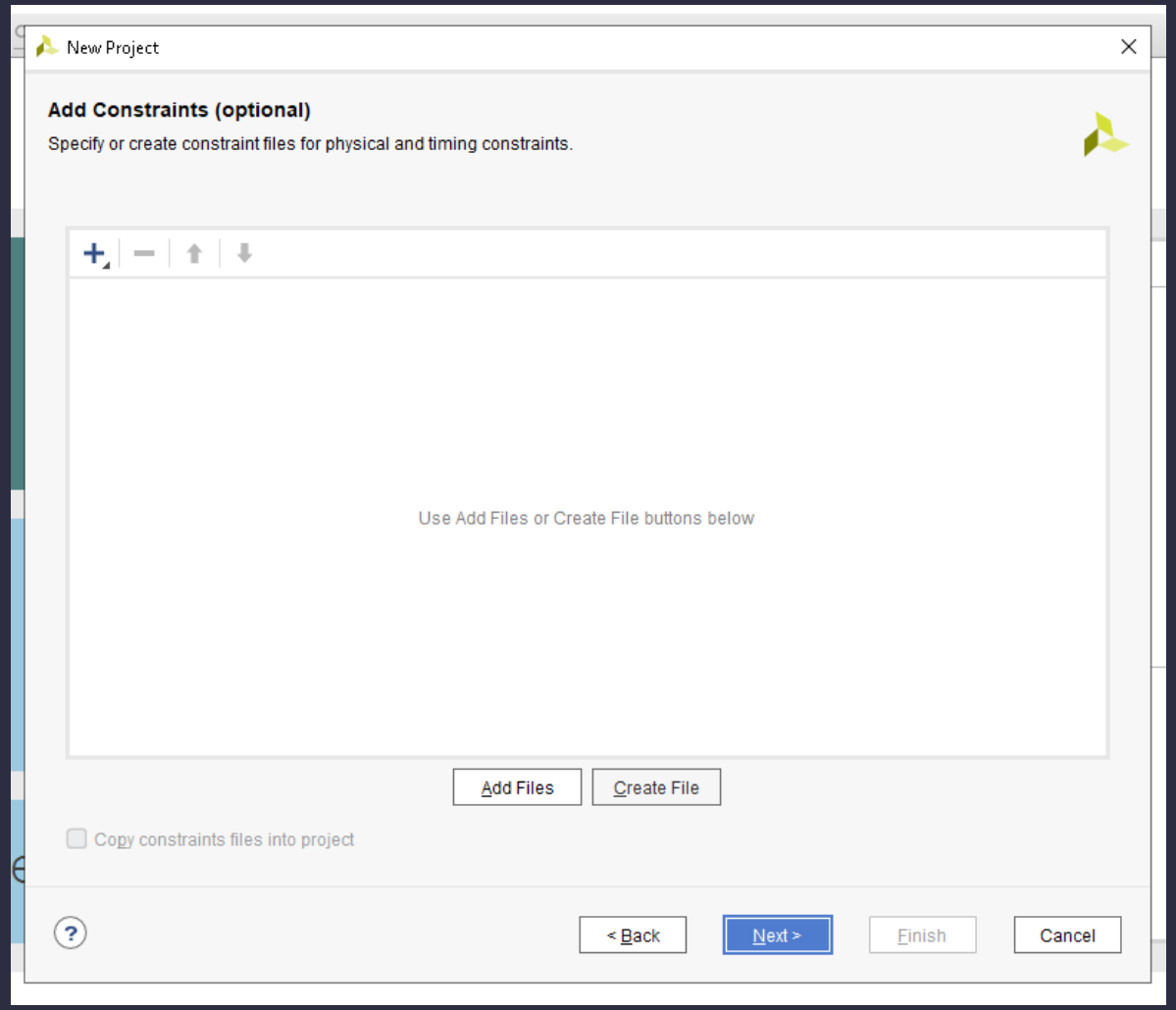
Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı



Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı



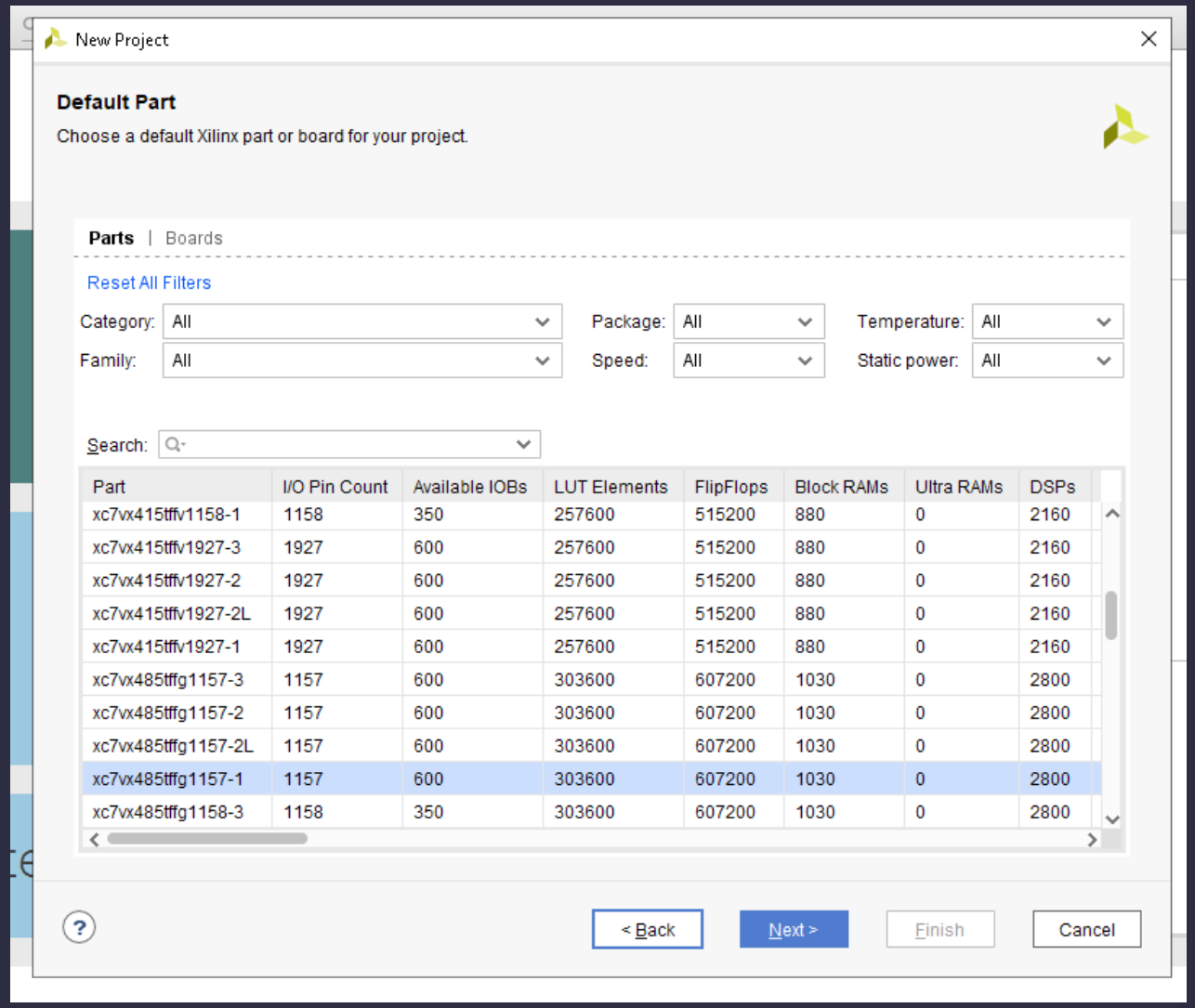
Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

LAB'larda kullanacağımız FPGA olan

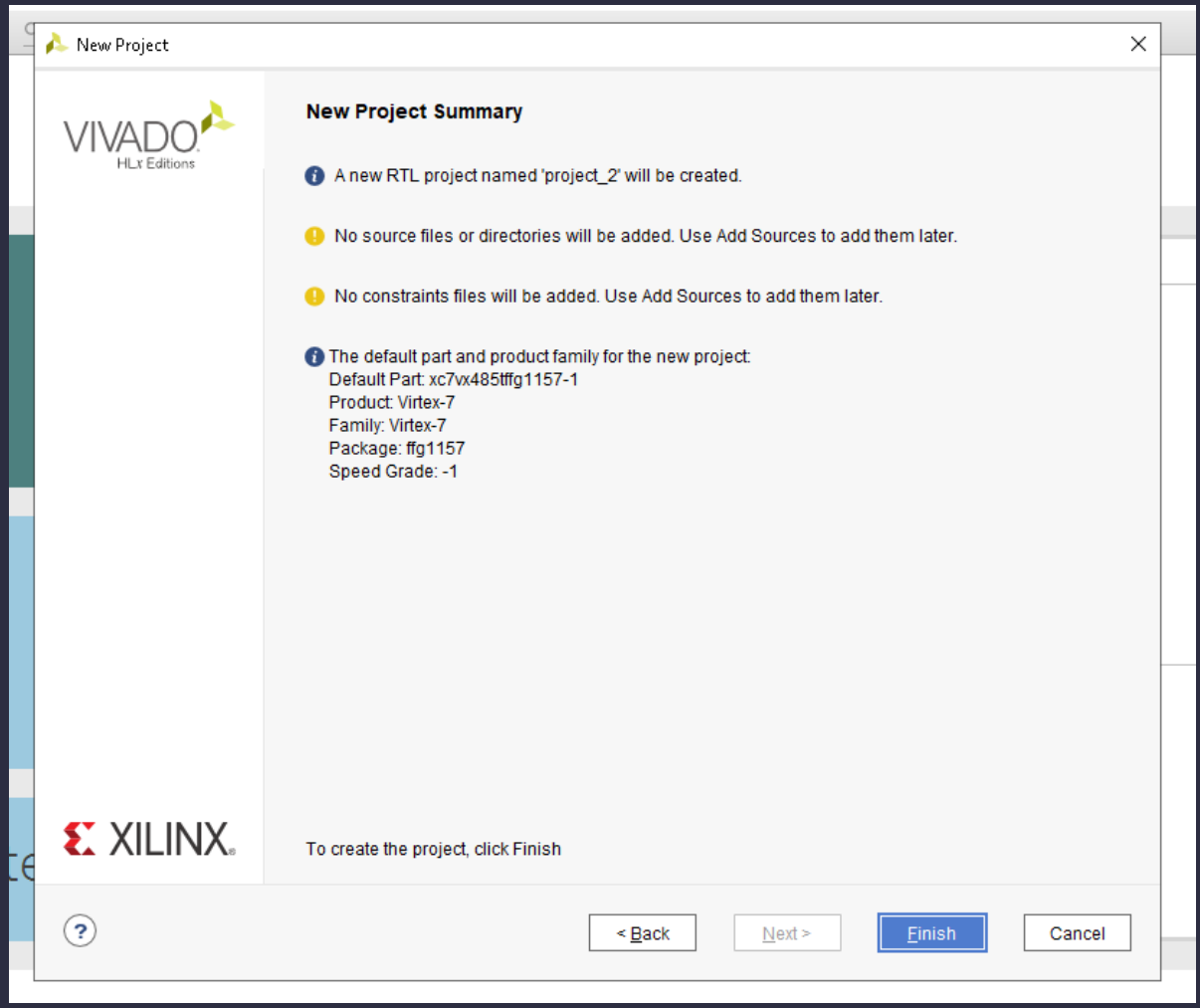
XC7A35Tcpg236-1

Model seçilmelidir.



Verilog – Kombinasyonel Devreler

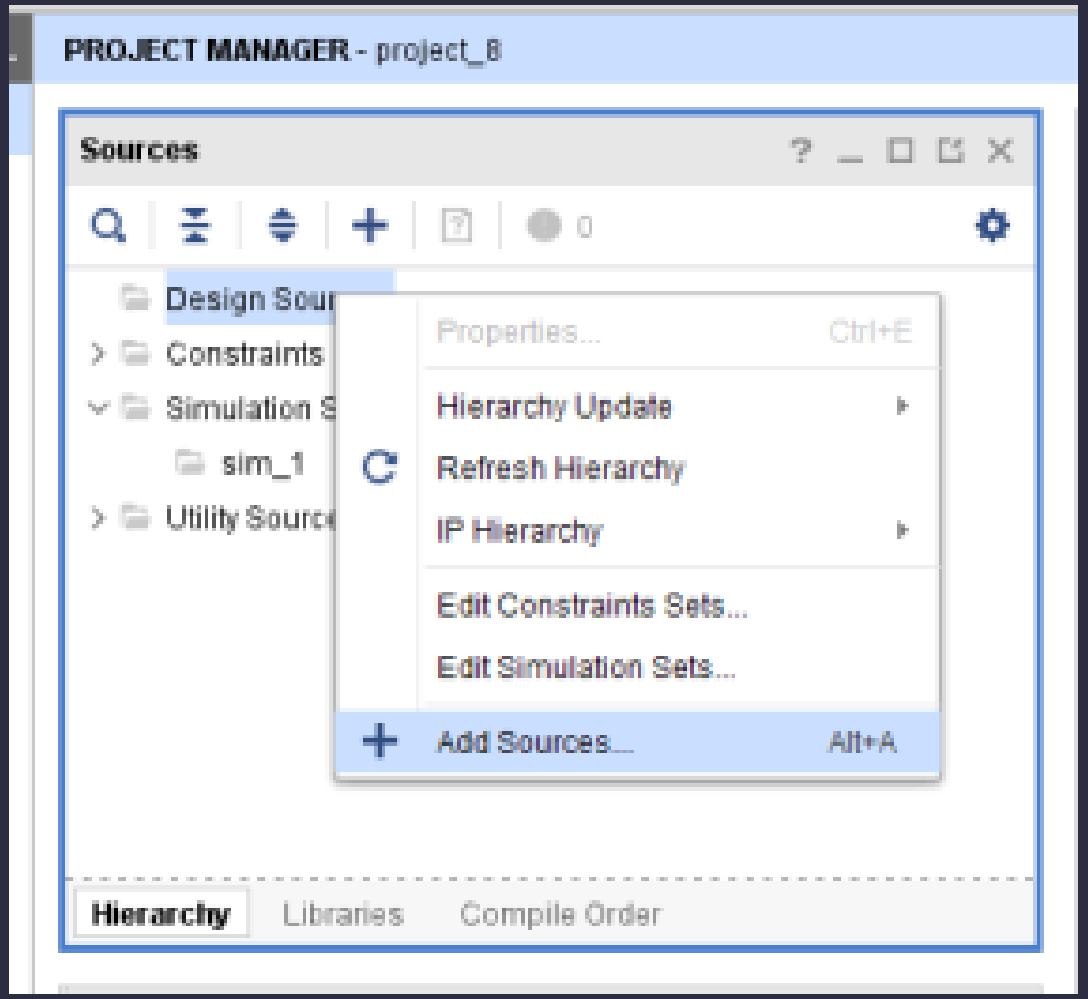
Vivado Tasarım Aracı



Verilog – Kombinasyonel Devreler

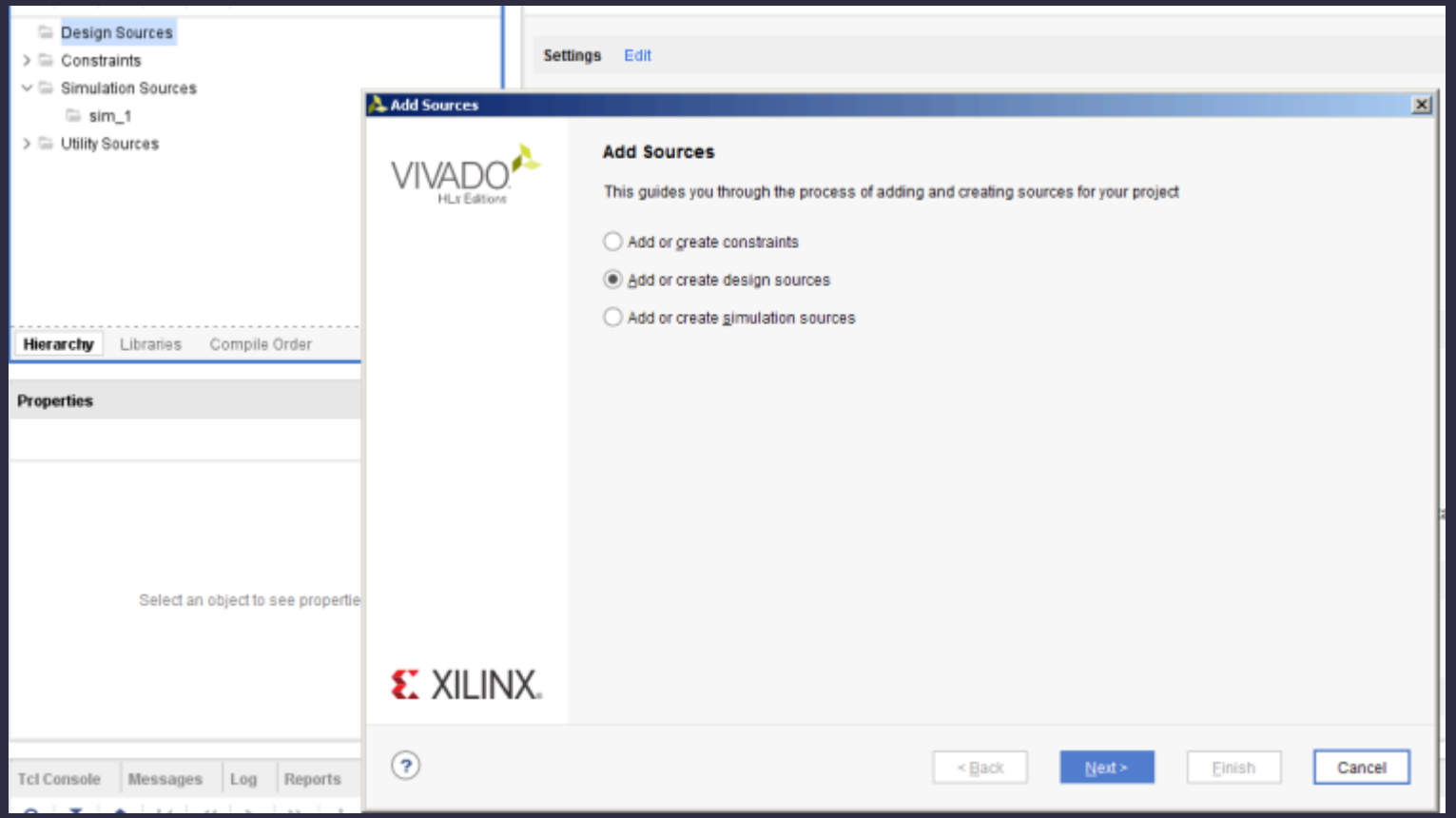
Vivado Tasarım Aracı

Yeni tasarım kaynağı eklemek



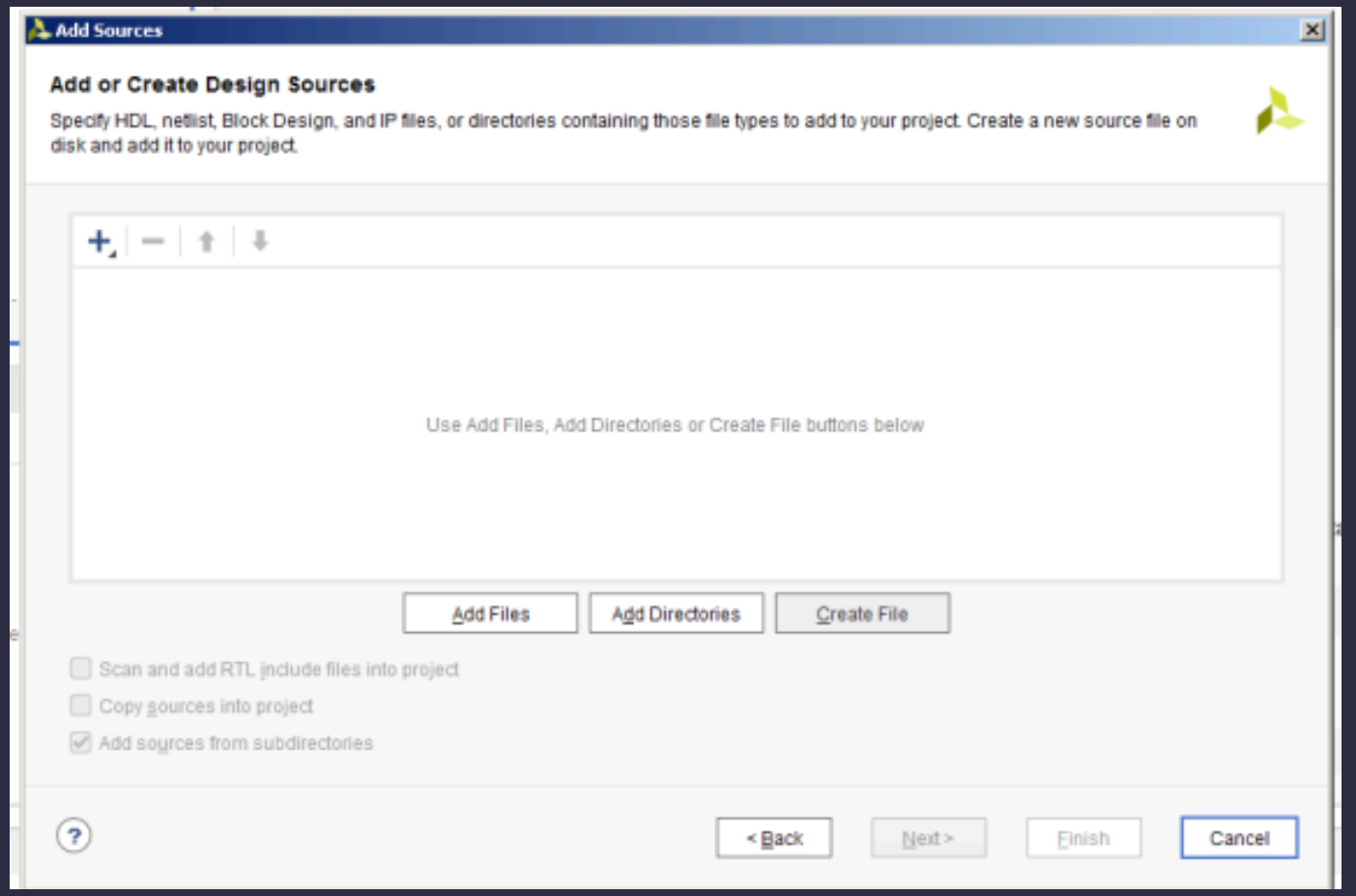
Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı



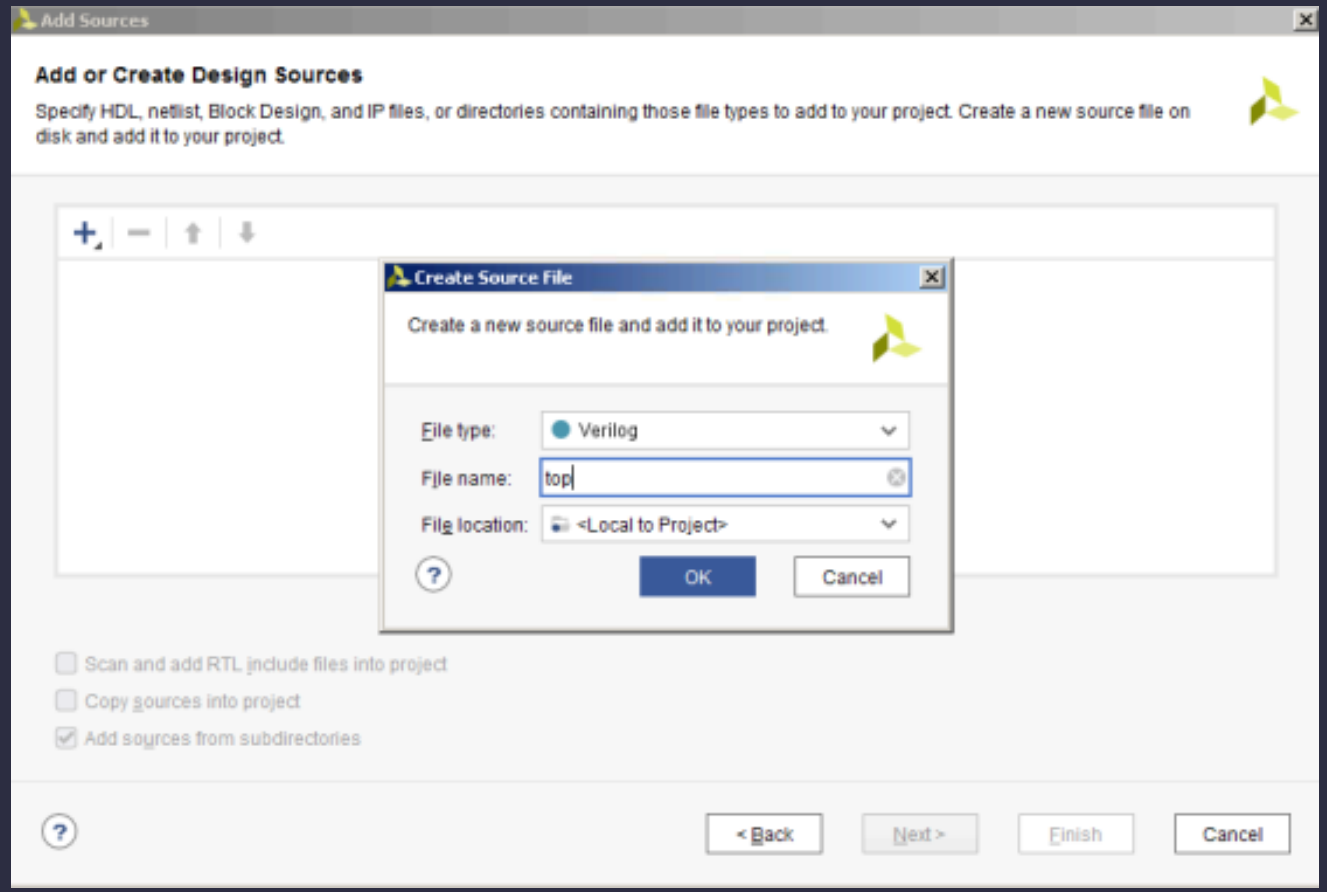
Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı



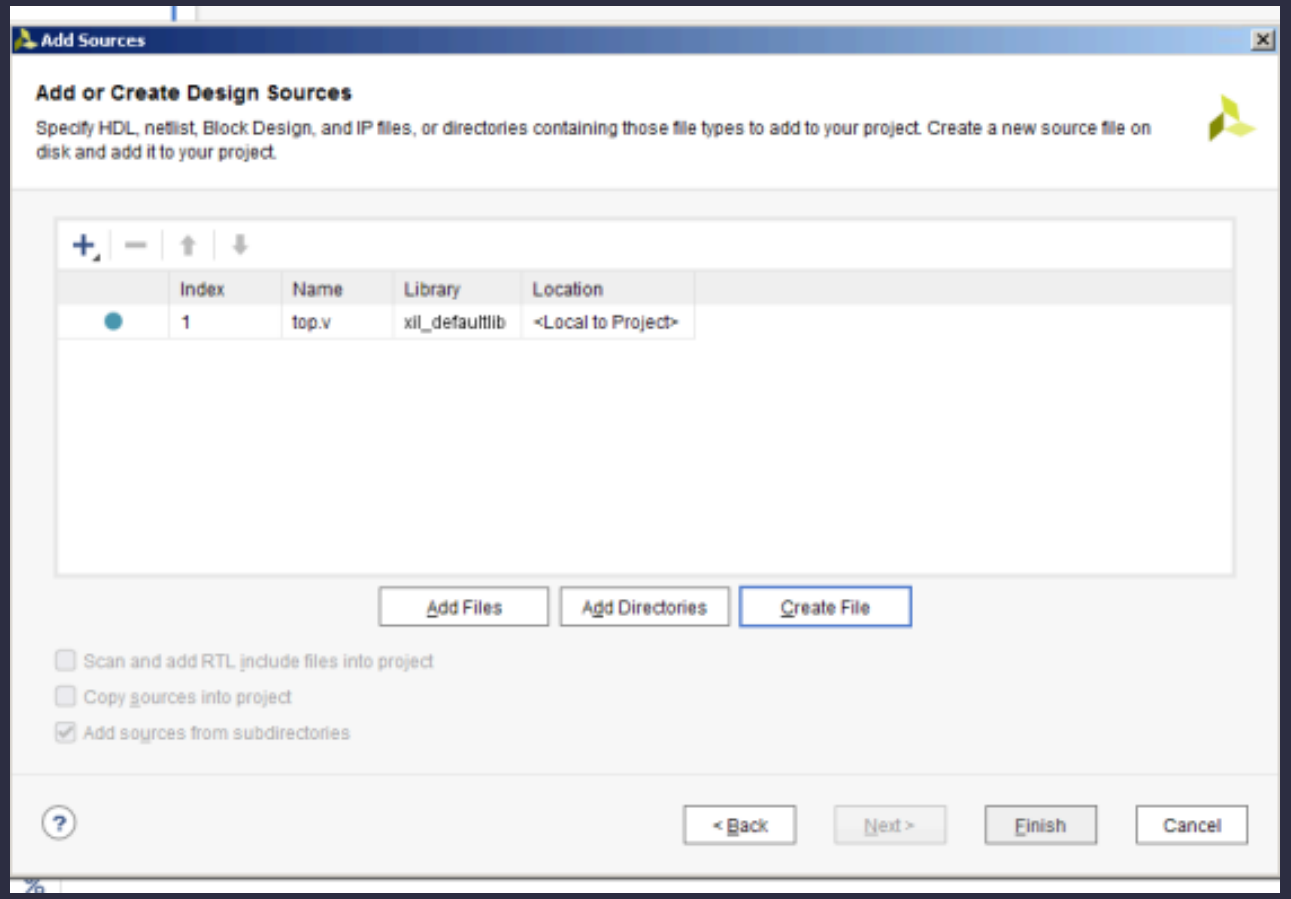
Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı



Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

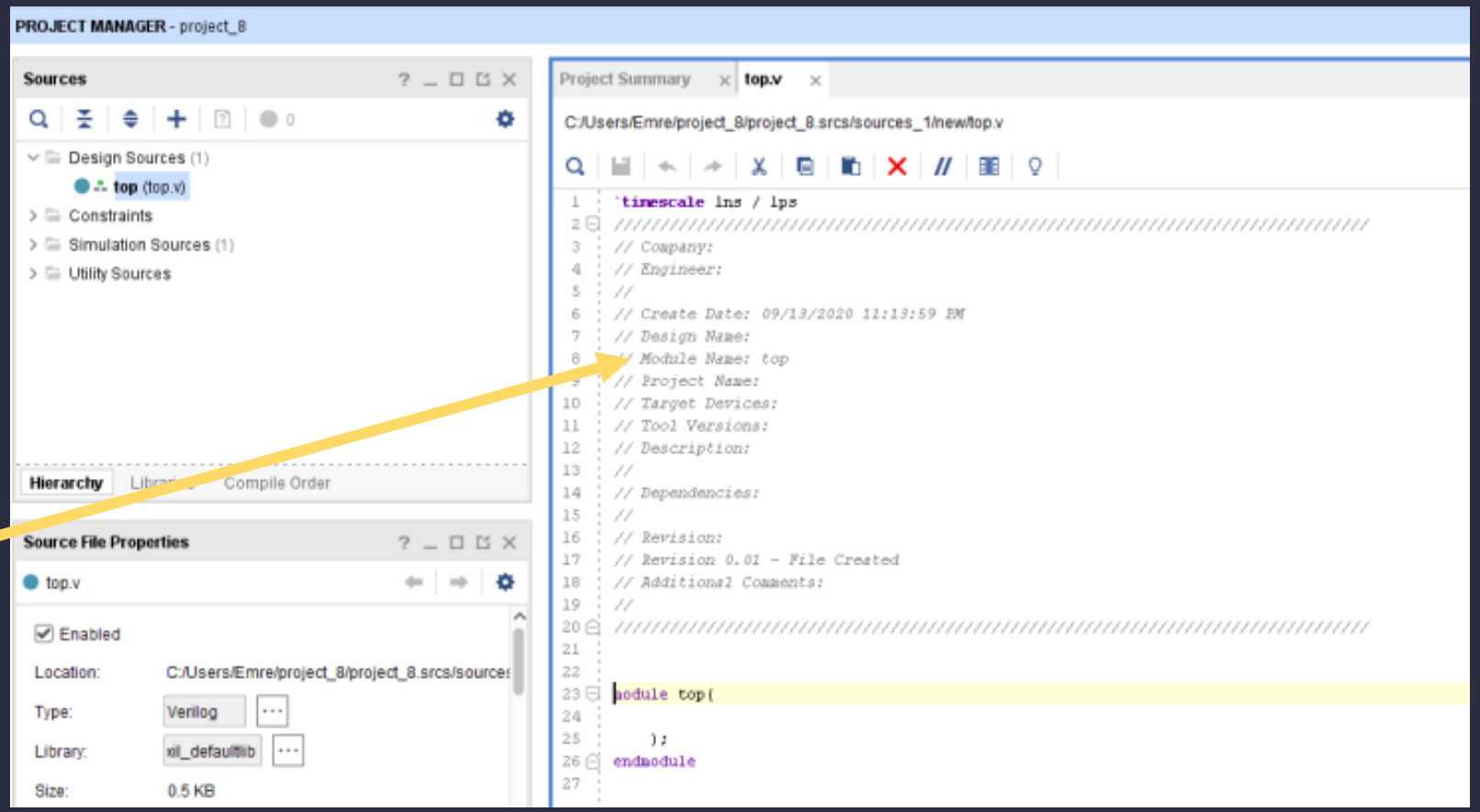


Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

RTL Tasarımı

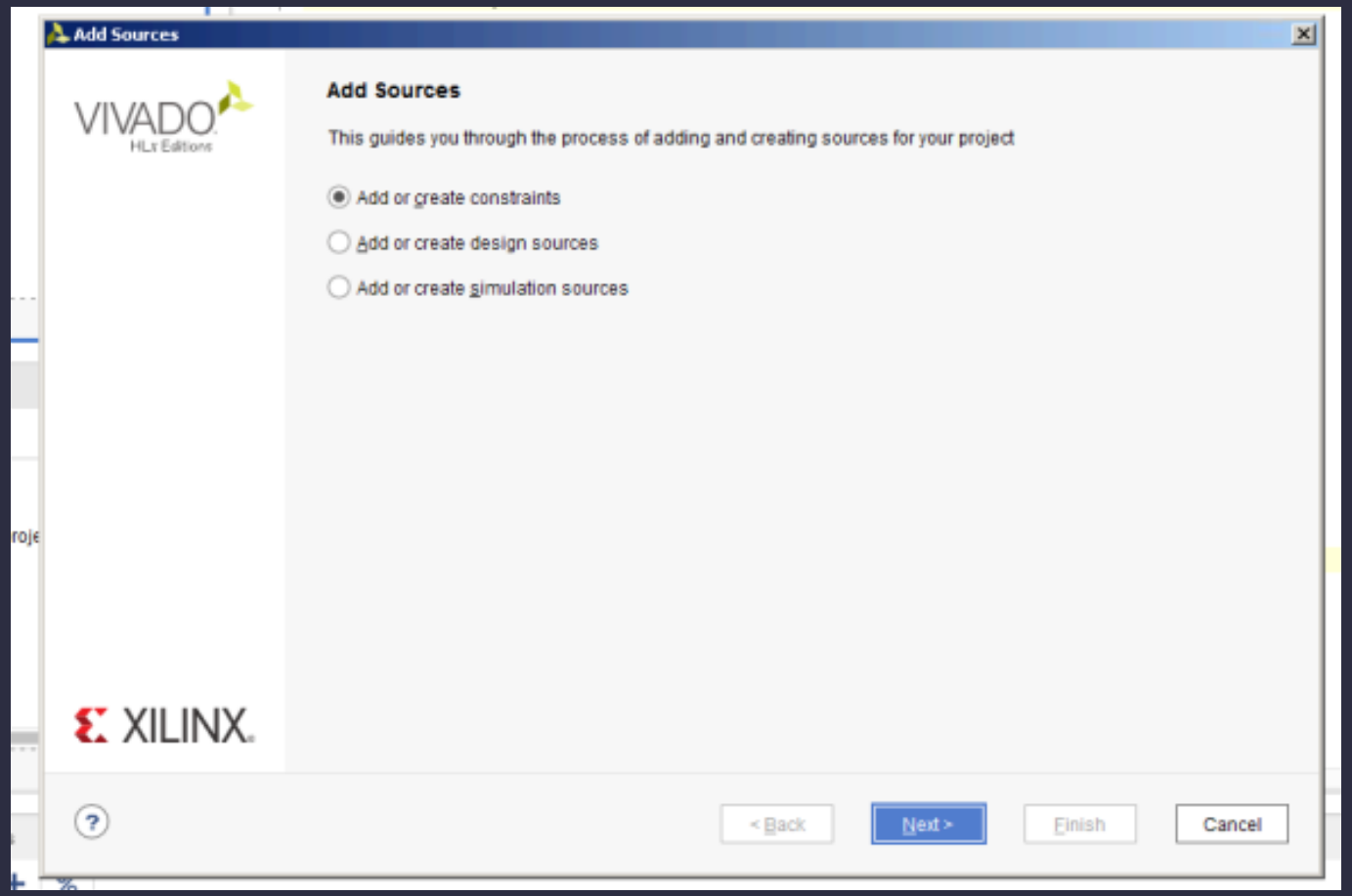
Yapılacak Dosya



Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

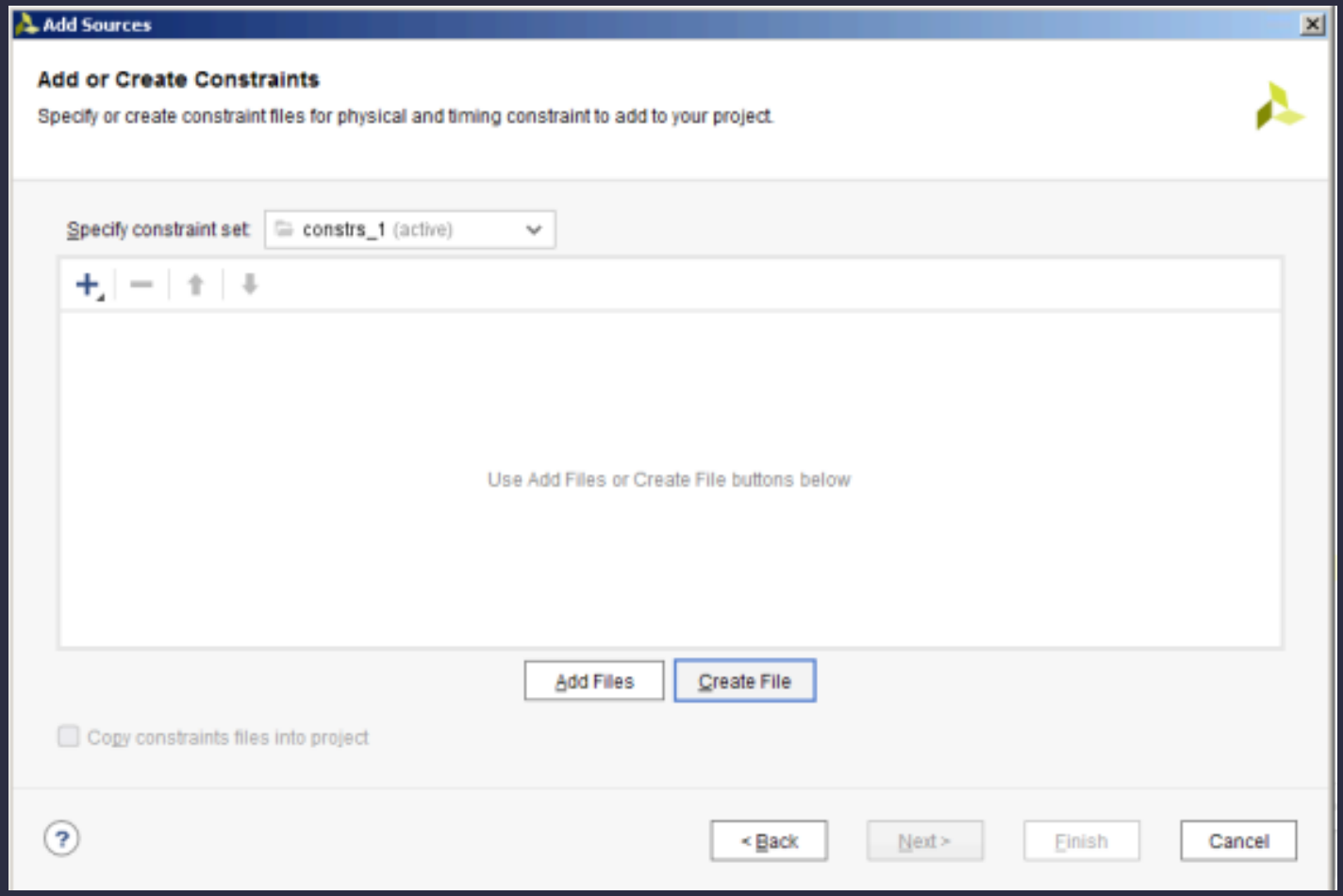
Kısıt (constraint) eklemek



Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

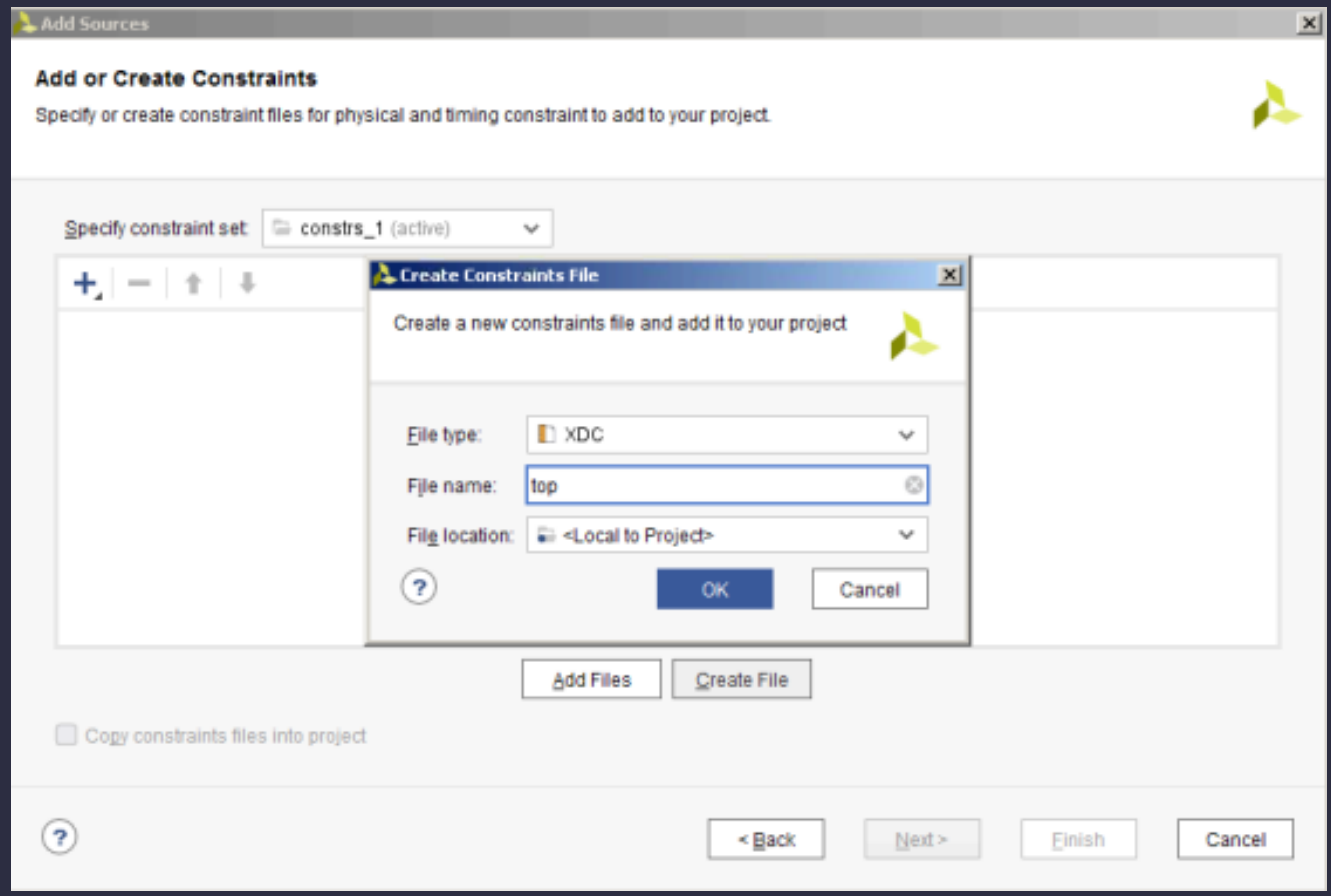
Kısıt (constraint) eklemek



Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

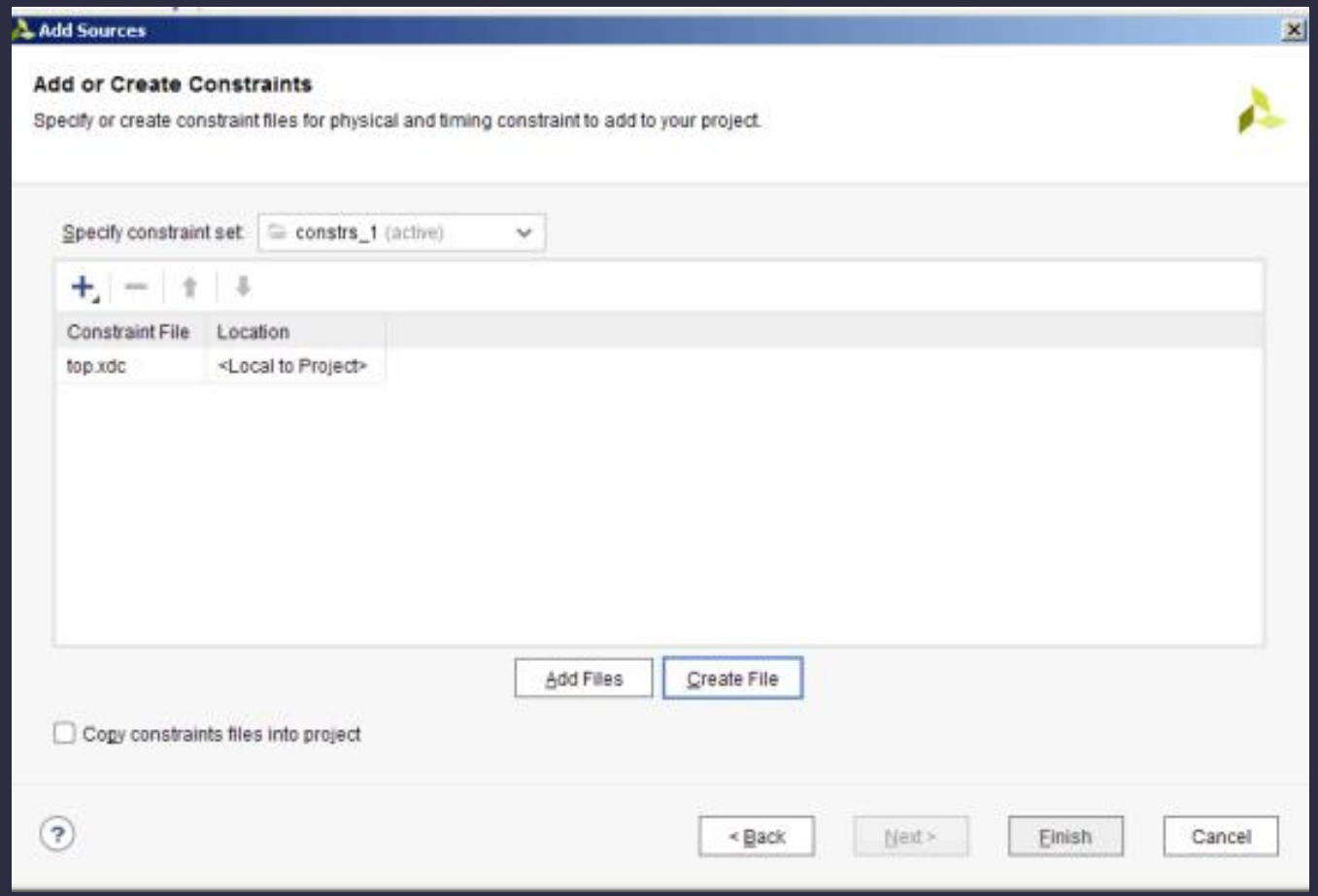
Kısıt (constraint) eklemek



Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

Kısıt (constraint) eklemek

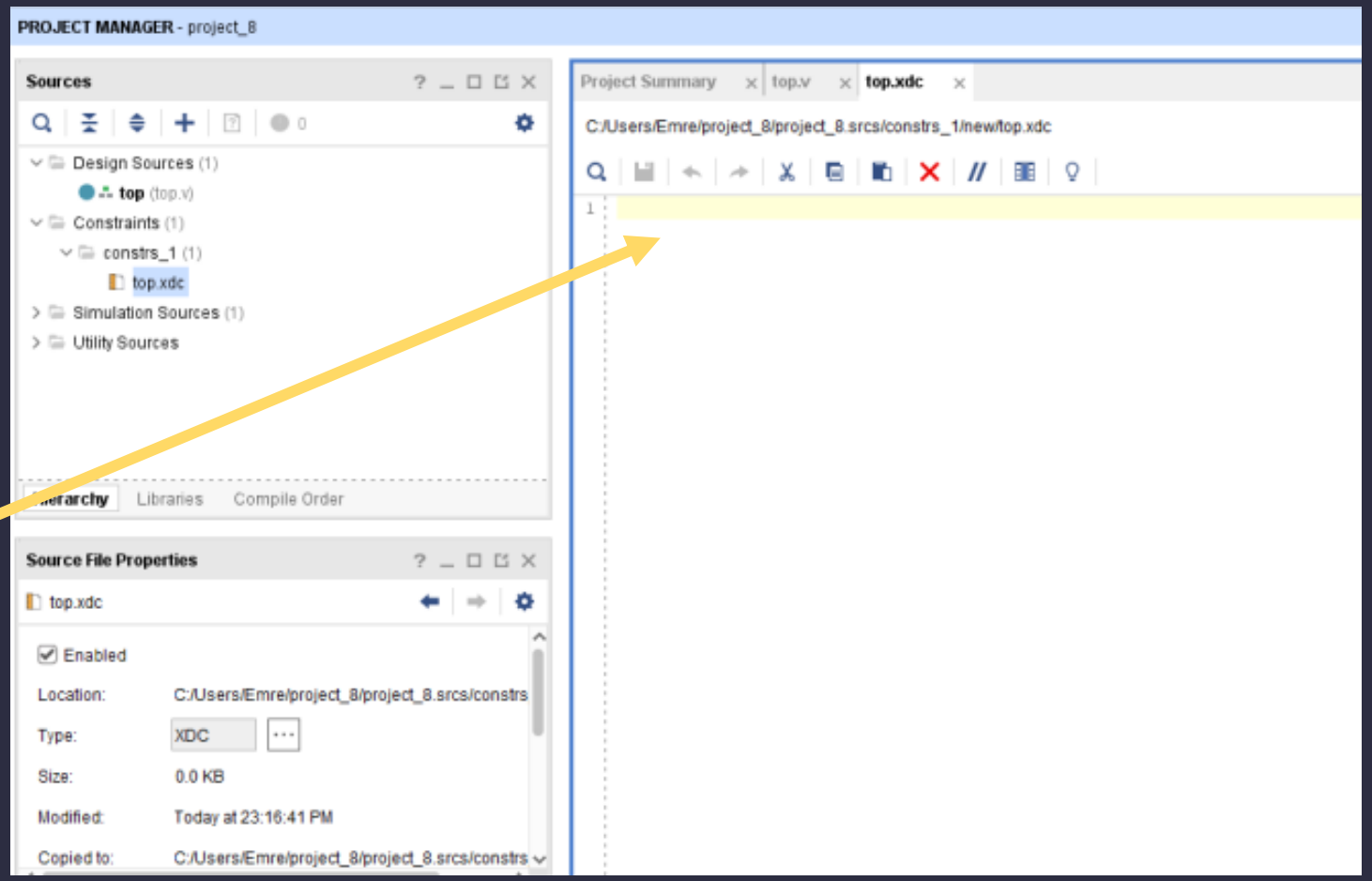


Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

Kısıt (constraint) eklemek

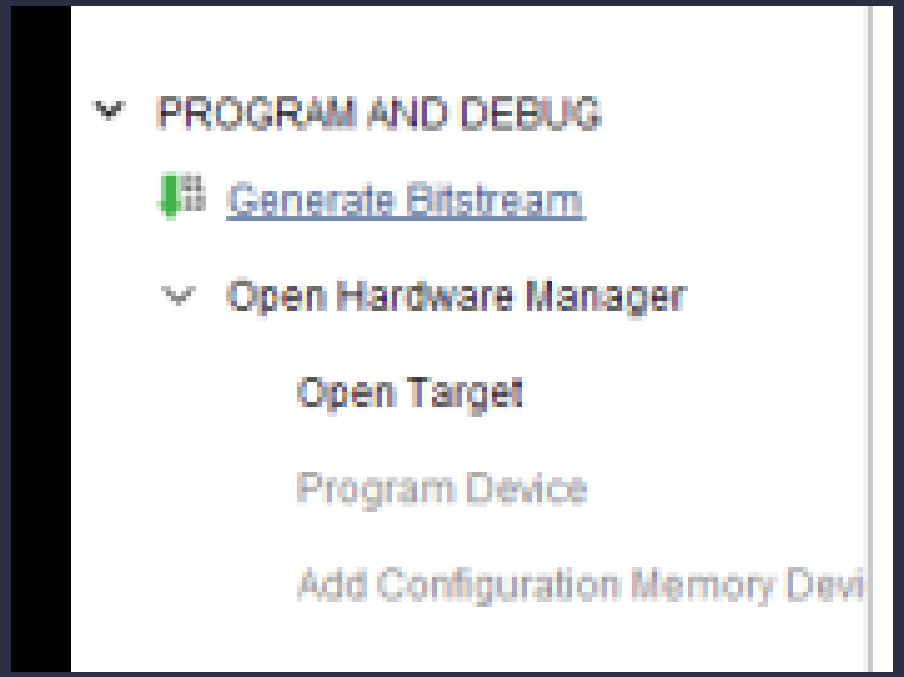
Kısıtların yazılacağı dosya



Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

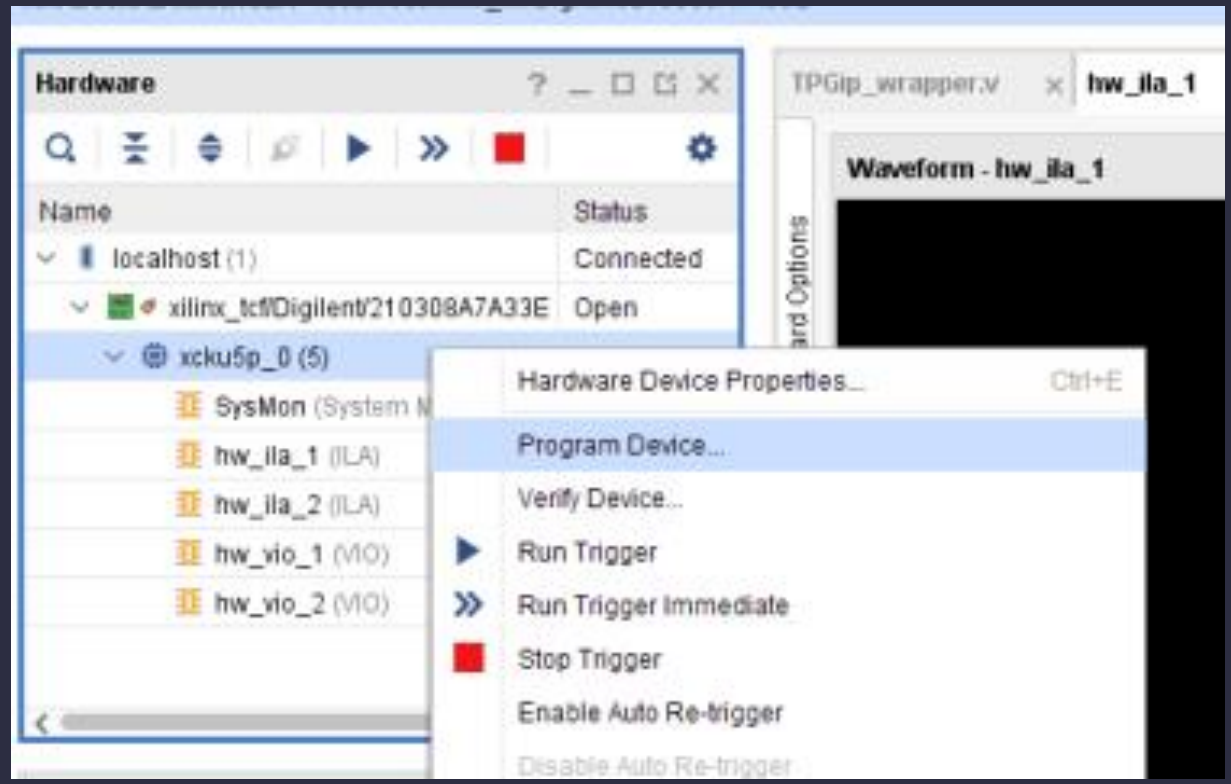
Bitstream üretimi



Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

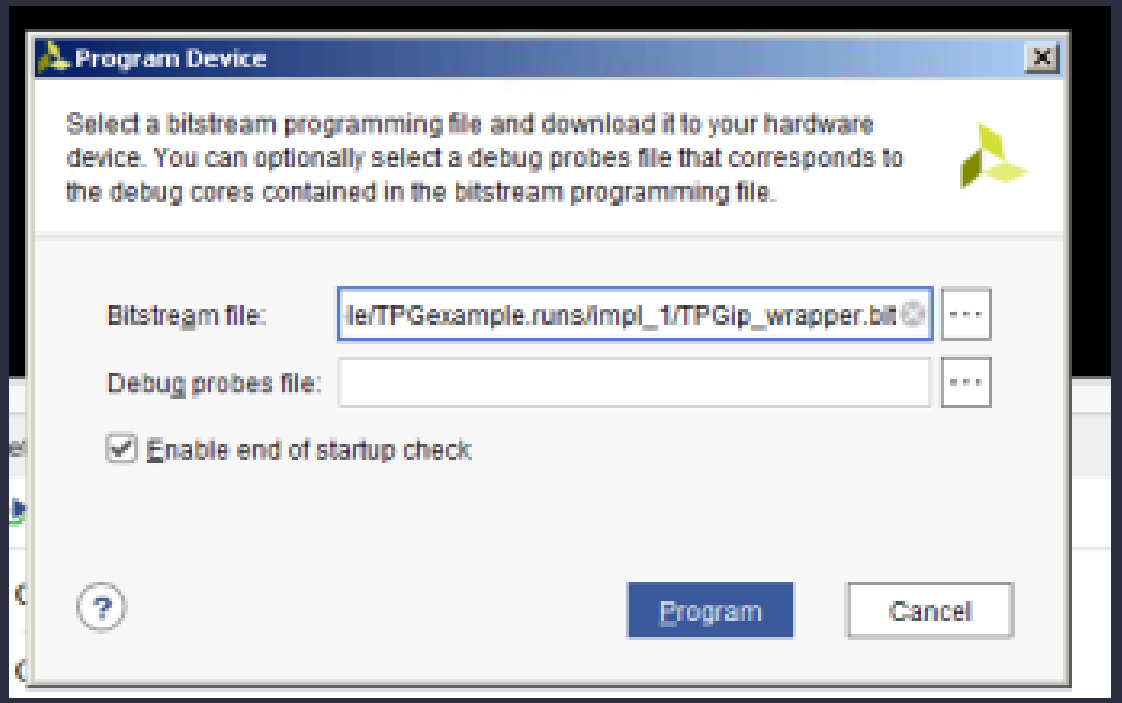
Bitstream üretimi



Verilog – Kombinasyonel Devreler

Vivado Tasarım Aracı

Bitstream üretimi

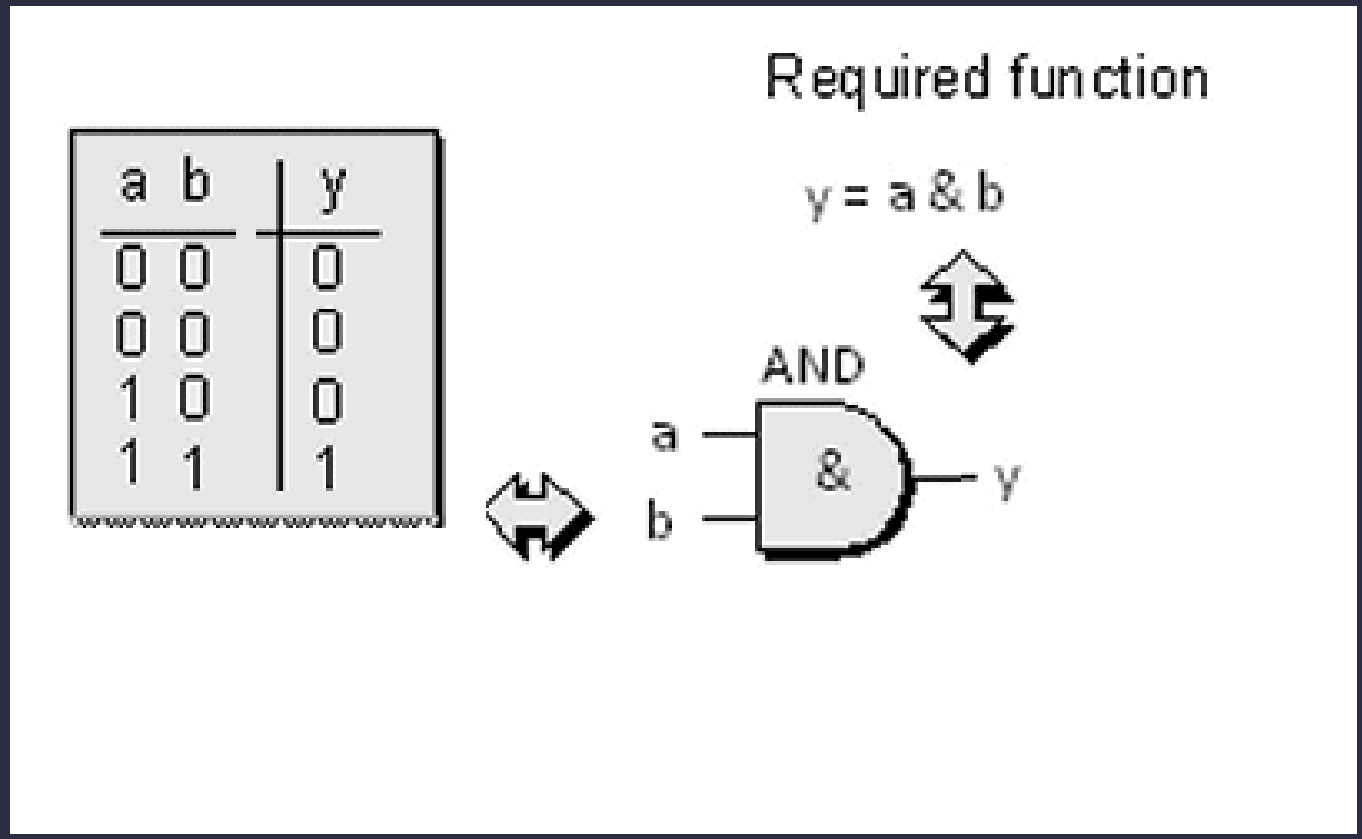


Verilog – Kombinasyonel Devreler

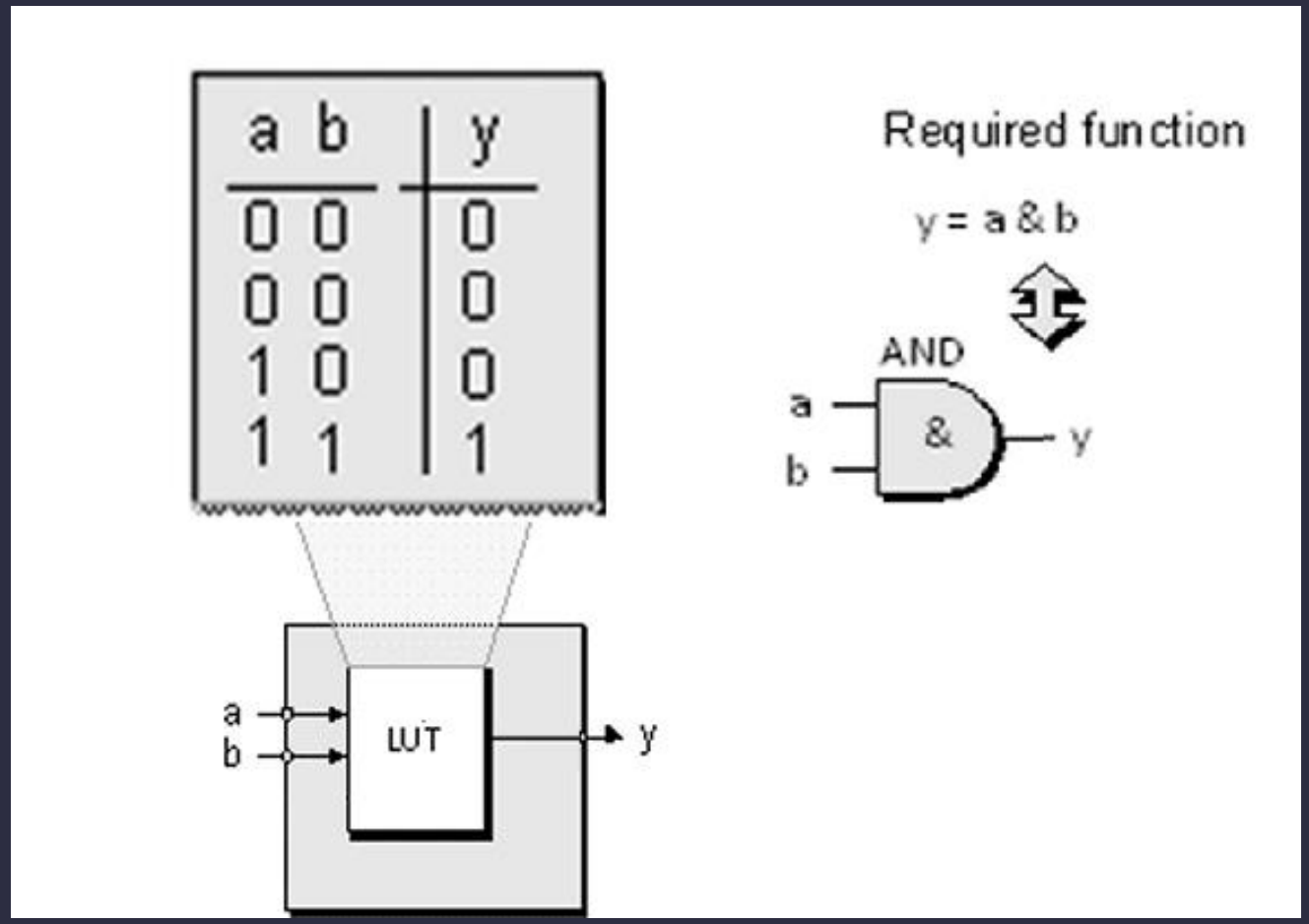
Öne çıkan HDL (Hardware Description Language) Dilleri

- *Verilog*
- *System Verilog*
- *VHDL*

Verilog – Kombinasyonel Devreler



Verilog – Kombinasyonel Devreler



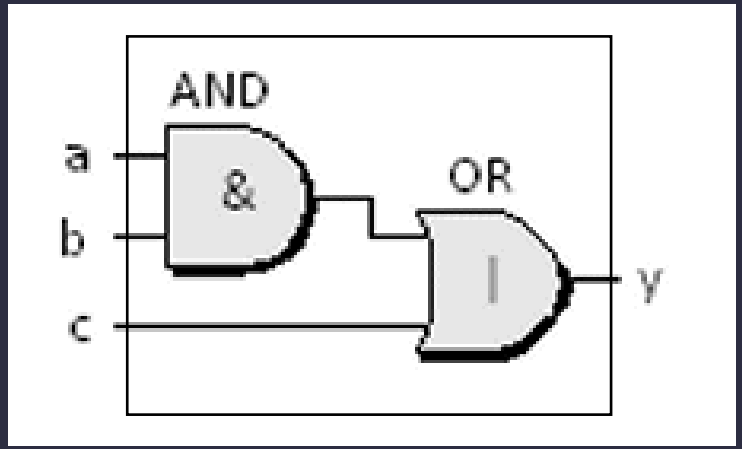
Verilog – Kombinasyonel Devreler

Vivado,

- Verilog
- System Verilog
- VHDL

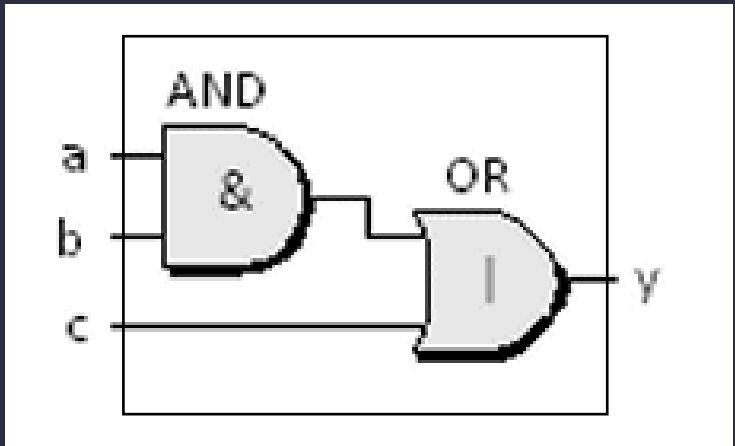
Dillerini desteklemektedir. Ders kapsamında Verilog dili ile tasarımlar yapılacaktır.

Verilog – Kombinasyonel Devreler



myModule

Verilog – Kombinasyonel Devreler

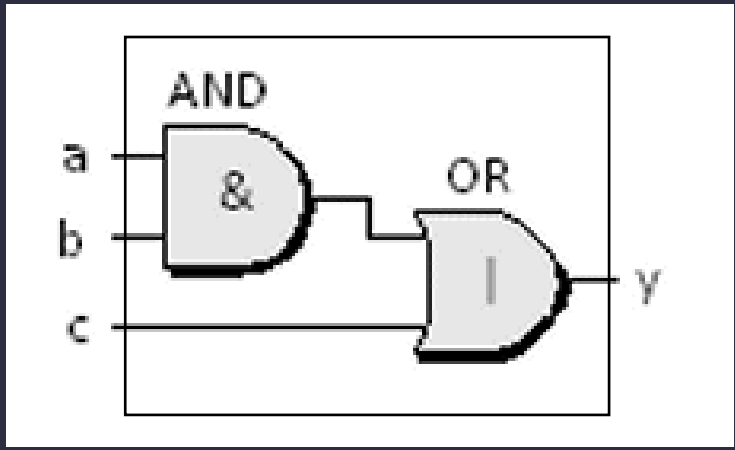


myModule

Verilog Tasarım

```
module myModule(input  a, input b, output y);



endmodule
```

Verilog – Kombinasyonel Devreler

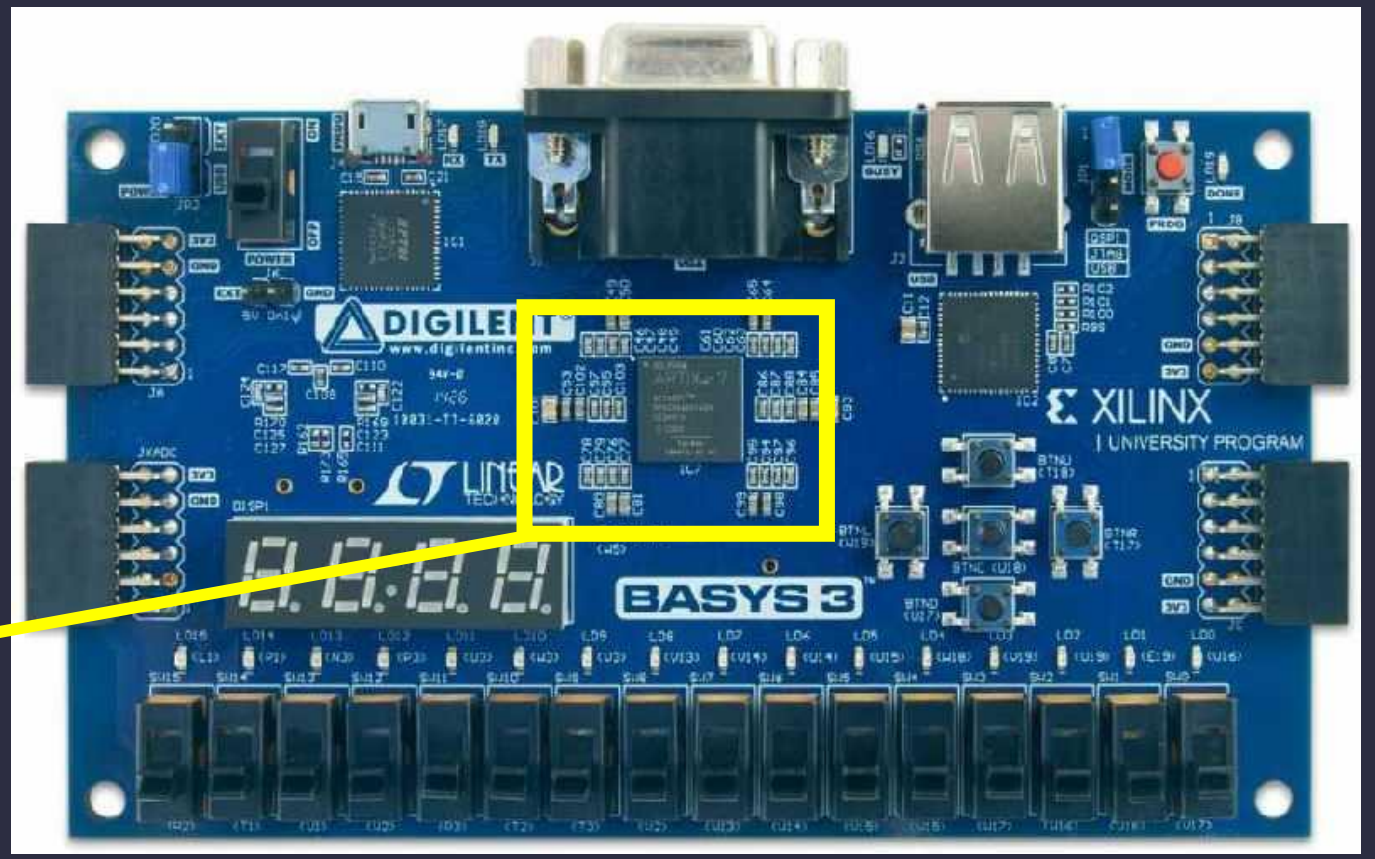


myModule

Verilog Tasarım

```
module myModule(input  a, input b, output y);

        reg tmp;
        always@(*) begin
                tmp = a & b;
                y = tmp | c;
        end

endmodule
```

Verilog – Kombinasyonel Devreler



Verilog – Kombinasyonel Devreler

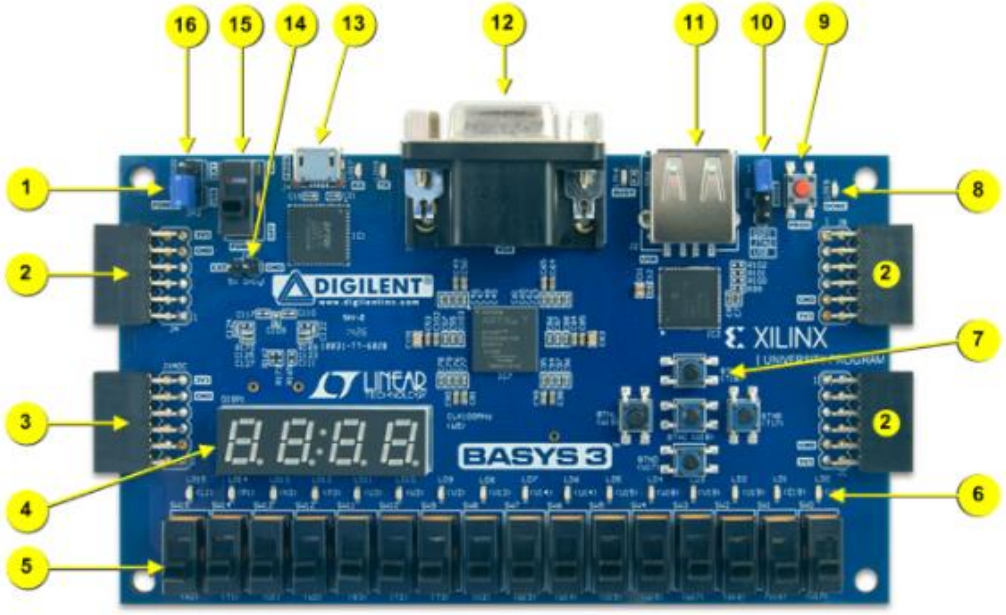
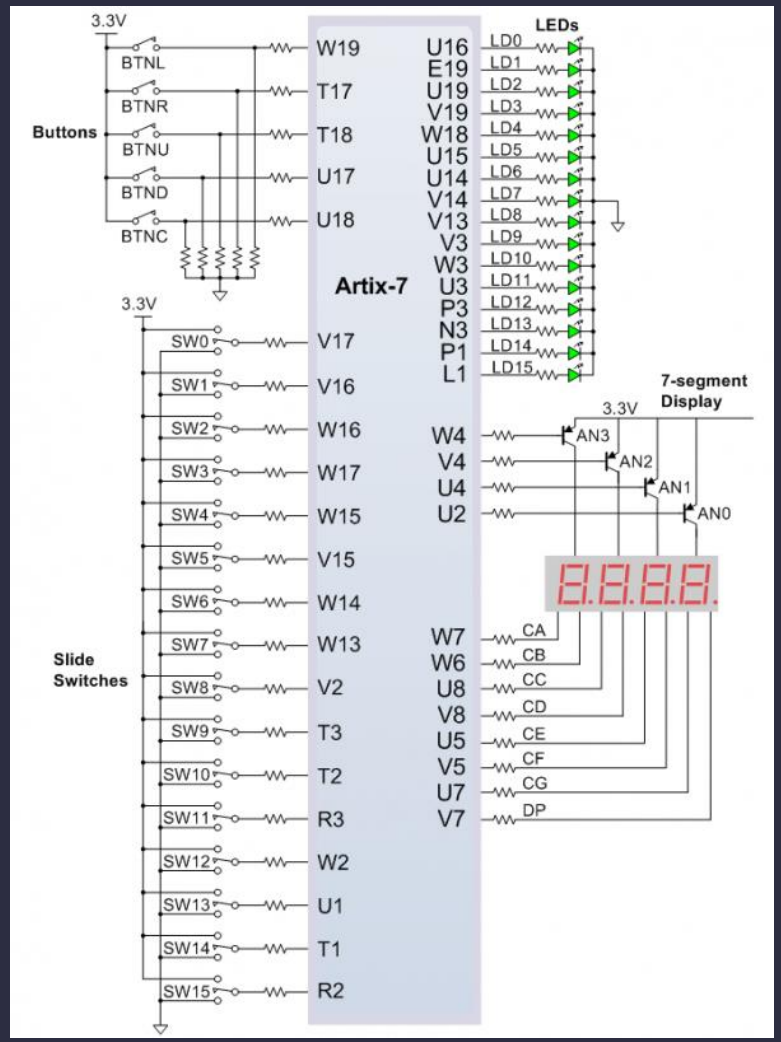


Figure 1. Basys3 board features

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod connector(s)	10	Programming mode jumper
3	Analog signal Pmod connector (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Verilog – Kombinasyonel Devreler



Verilog – Kombinasyonel Devreler

Kısıt (XDC) Dosyası

http://levent.tc/files/courses/digital_design/labs/basys3.xdc

Ardışık Devreler

- Ardışık Devreler
  - *Ardışık devrelerde çıkış, sadece şu andaki girişlere değil, önceki girişlere de bağlıdır.*
  - Örnek: Bir toplama devresi ile yukarı doğru sayan sayaç

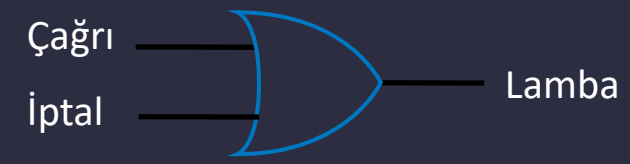
  - Flip-flop denen bir bit'lik hafıza birimi
  - Flip-flopların bir araya gelerek çoklu bit saklama alanı olan saklayıcılar (register)

içermektedir.

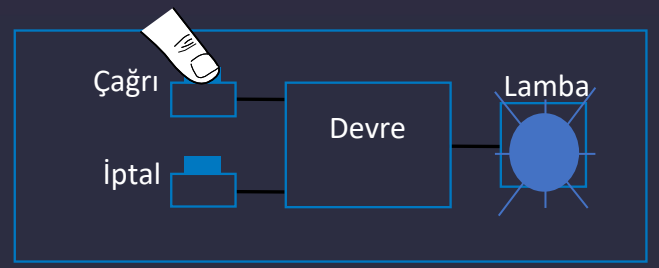
Ardışık Devreler

- Uçuş görevlisi çağrı butonu
  - Basıldığında, lamba tekrar basılana kadar aktif kalır.
  - Tekrar basıldığında lamba söner

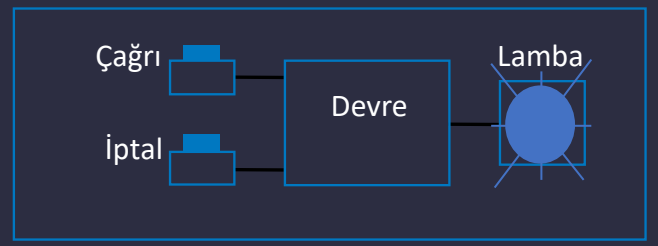
Kombinasyonel devreler ile yapılabilir mi?



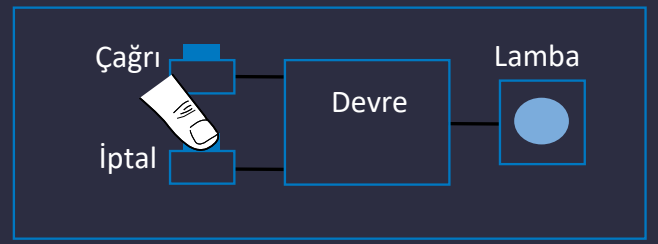
Çalışmayacaktır, çünkü lamba sadece çağrı basılı iken 1 olacaktır. Çağrı butonuna basılmadığında lamba 0'a dönecektir. İstenen davranış için devrenin önceki durumunu tutan bir saklayıcıya ihtiyaç vardır



1. Butona basıldı, ışık yandı



2. Buton bırakıldı, ışık yanmaya devam ediyor



3. İptal edildi, ışık söndü

Ardışık Devreler

Tutucu (Latch) Türleri

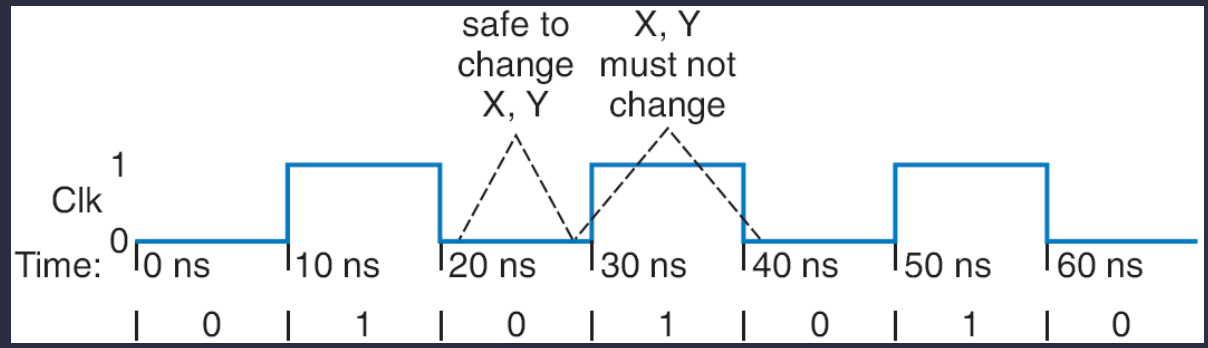
- ***SR(Set-Reset) Latch***
- ***D Latch***

Latch'ler level sensitive olarak çalışır.

- ***D Flip Flop***

Flip flop'lar ise edge sensetive olarak çalışır.

Ardışık Devreler

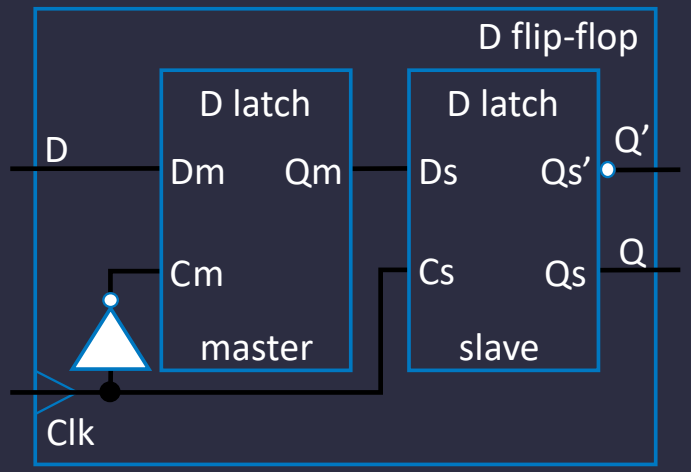


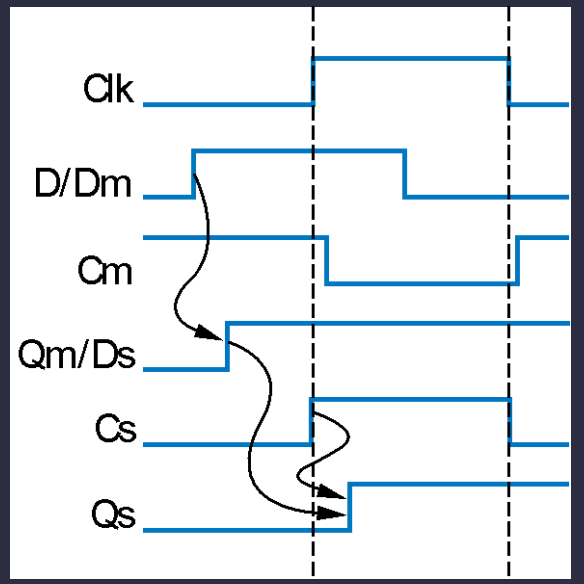
- ***Clock Periyodu***: İki yükselen kenar arası geçen süre
  - Yukarıdaki sinyalin periyodu: 20 ns
- ***Clock cycle***:
  - Bir zaman aralığında geçen yükselen kenar clock sayısı
- ***Clock Frekansı***: 1/periyot
  - Yukarıdaki sinyalin periyodu = 1 / 20 ns = 50 MHz

Freq	Periyot
100 GHz	0.01 ns
10 GHz	0.1 ns
1 GHz	1 ns
100 MHz	10 ns
10 MHz	100 ns

Ardışık Devreler

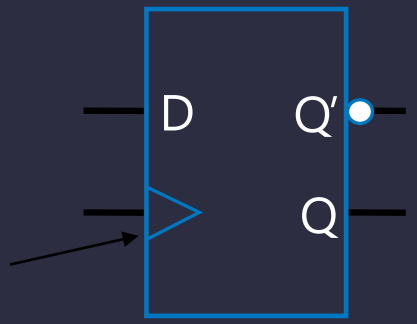
- ***Flip-flop***: Dışarıdan gelen sinyali, kendisine gelen clock'un yükselen kenarında örnekler
- Tasarım örneği– master-slave
  - 2 D latch kullanılır.



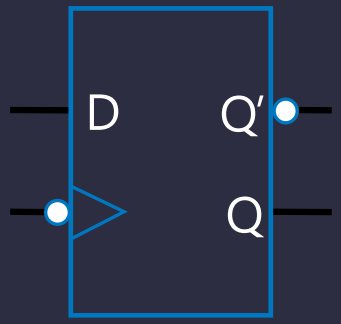


Ardışık Devreler

Üçken, clock girişini ifade etmektedir



Yükselen Kenarlı aktif olan flip flop gösterimi



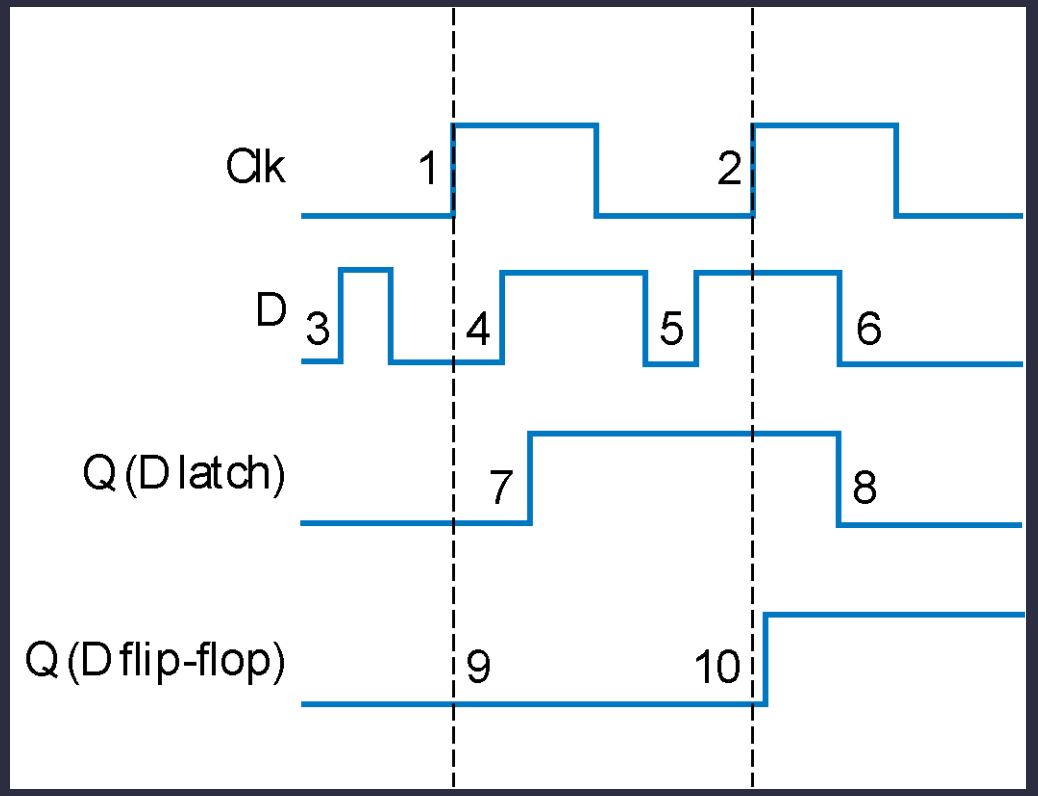
Düşen Kenarlı aktif olan flip flop gösterimi

Yükselen Kenarlar (Positive Edge)

Düşen Kenarlar (Negative Edge)

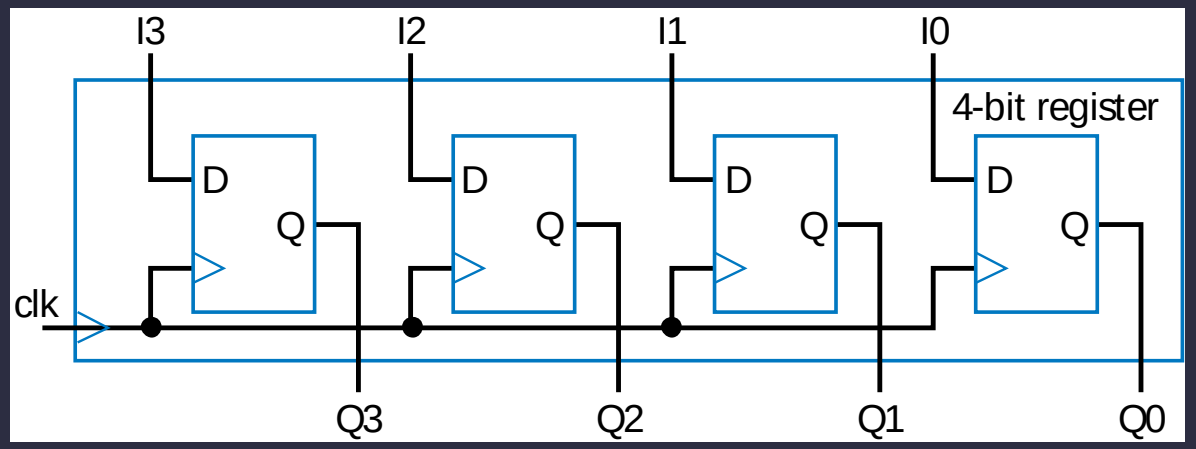
Ardışık Devreler

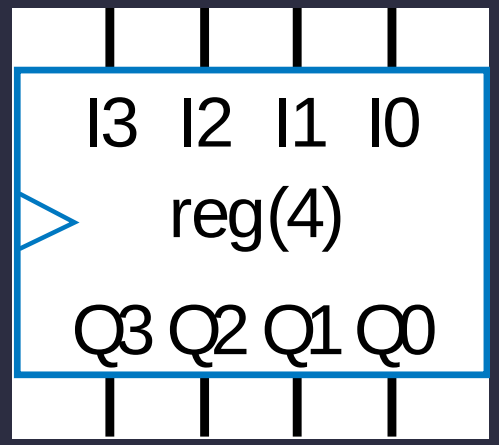
- D latch
- D flip flop



Ardışık Devreler

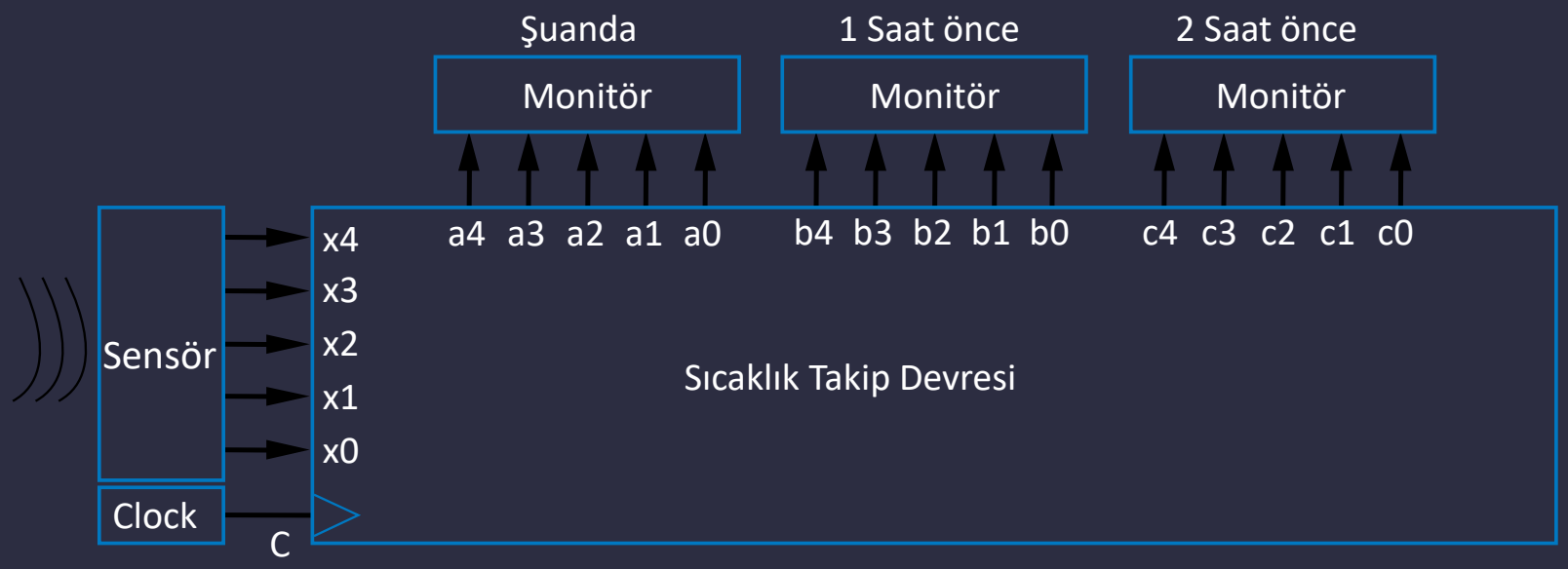
- Birden çok bit tutulma ihtiyacı olduğunda, D flip flop'lar bir arada kullanılır.
  - Örneğin 4 bitlik bir sayının tutulma ihtiyacı olduğunda
- Çoklu flip flopların bir arada tutulduğu yapıya ***Register*** *denmektedir.*





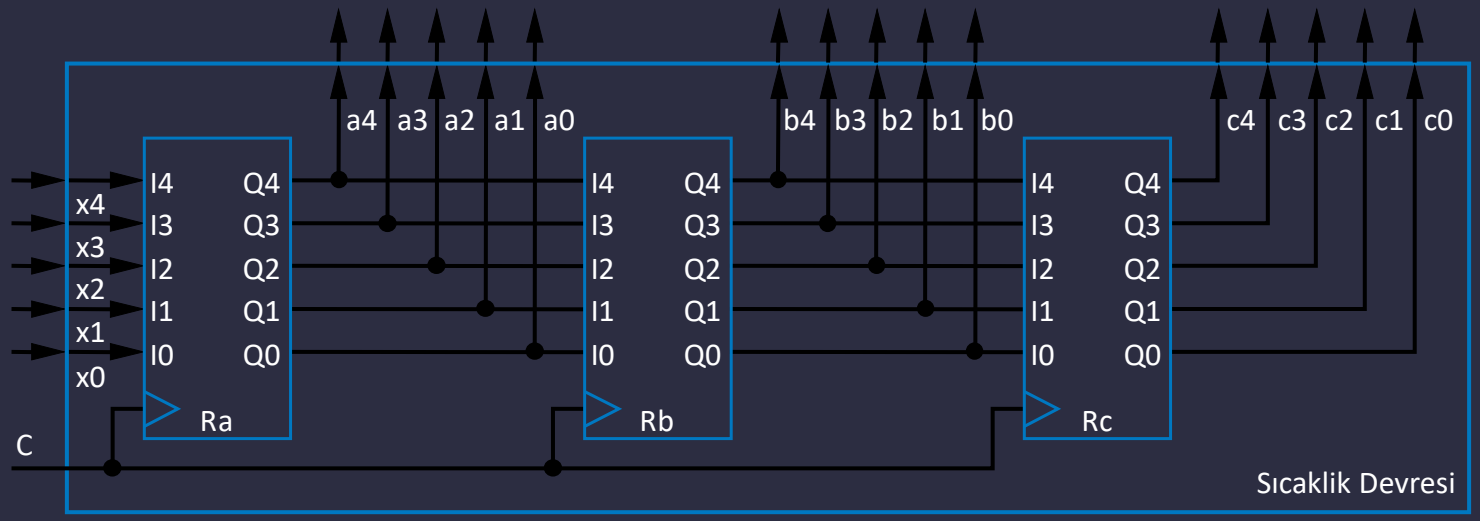
Ardışık Devreler

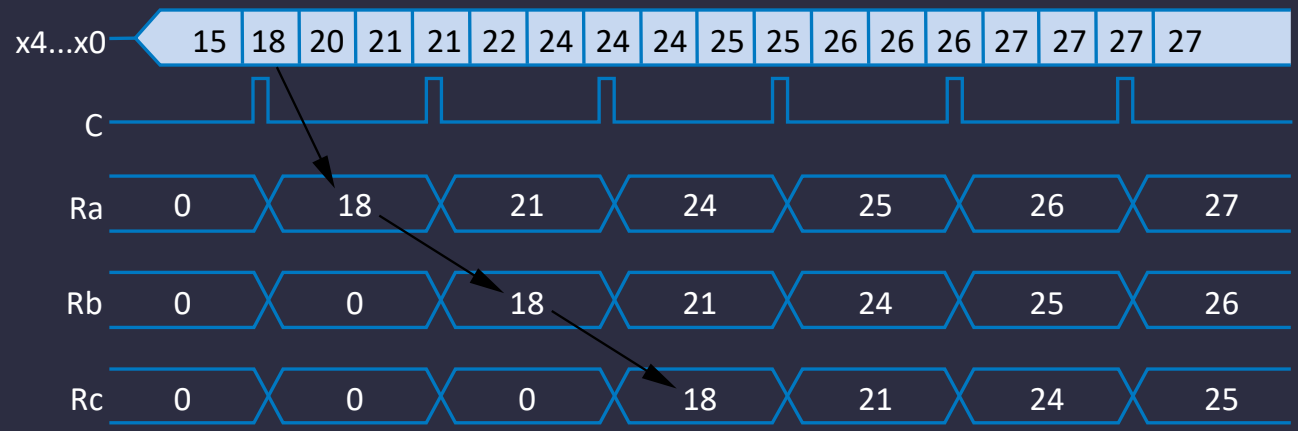
- Sıcaklık Devresi
  - Sıcaklık sensörü 5 bitlik çıktı vermektedir.
  - Clock'un periyodu 1 saattir. (Oldukça yavaş bir clock)
  - Her clock yükselen kenarda kayıt alıp monitöre gösteren bir devre yapılmaktadır.



Ardışık Devreler

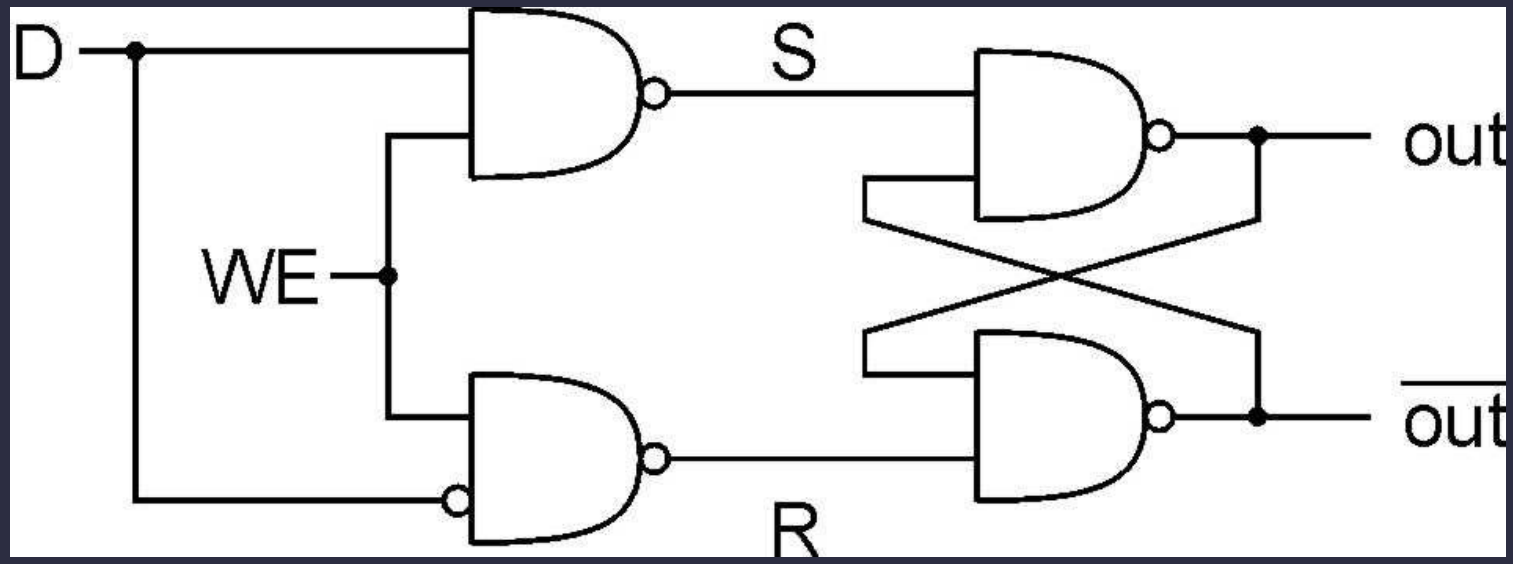
- 5bitlik saklayıcılar





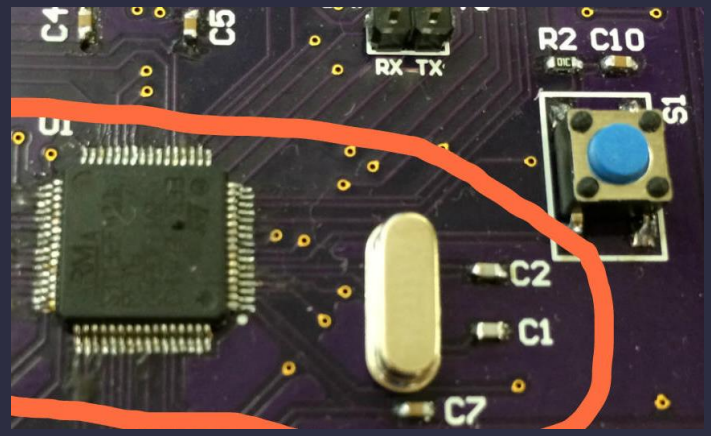
D Tipi Tutucular (D Latch)

- İki girişi vardır. Bunlar; D (data) ve WE (Write Enable)
  - WE = 1, D girişindeki değeri içerisine alır.
    - S = NOT(D), R = D
  - WE = 0, önceki değerini tutar.

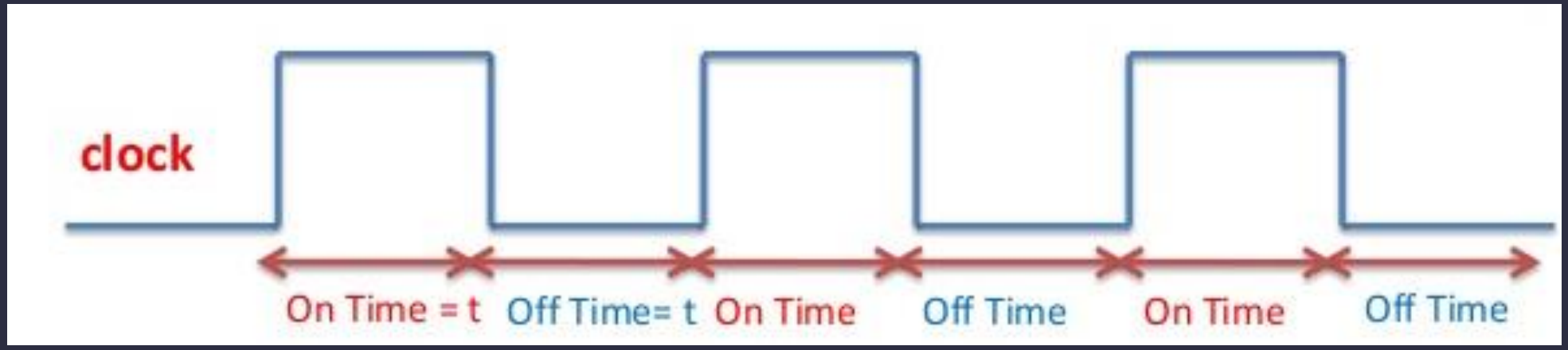


Saat Kristali (Clock Crystal)

- Periyodik olarak kare tipinde bir sinyal üretir.
- Birisi belli aralıklar ile bir anahtarı açıp kapatıyor gibi…



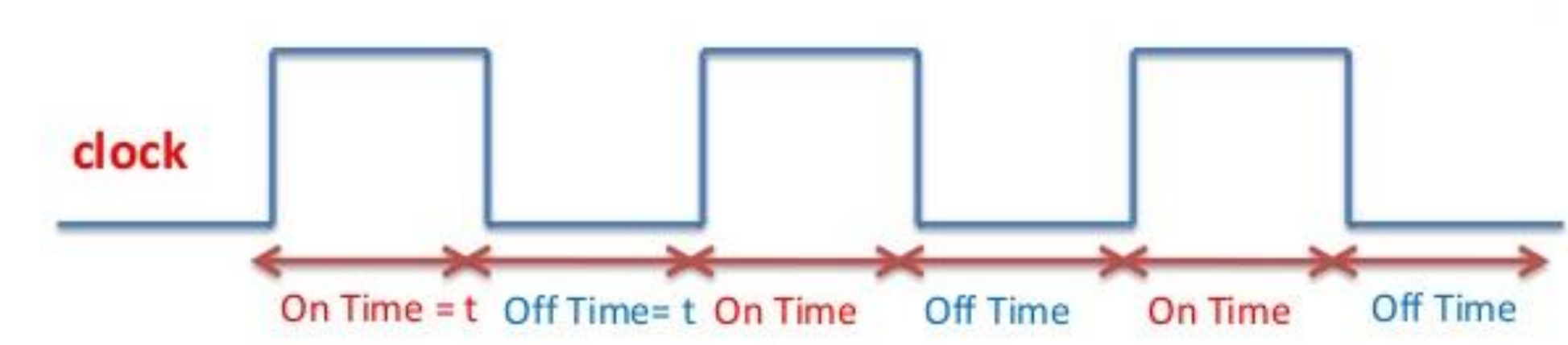
Saat (Clock)



- Şekilde verilen clock sinyali, 2t zamanında/periyodunda kendini tekrarlamaktadır. Clock'un her bir periyoduna Saat Çevrimi (Clock Cycle) denmektedir.

- Frekans = 1/Periyot demektir.

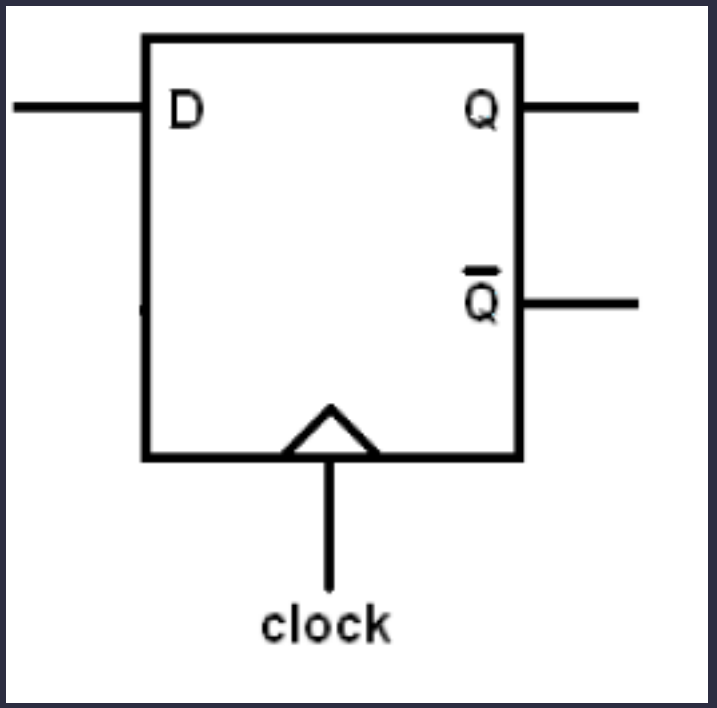
Saat (Clock)



- Yani bir clock sinyalinin periyodu 100 ms ise (Yani her 100ms'de bir kendini tekrarlıyor – İki yükselen kenar arası 100 ms ise),
- Bu sinyalin frekansı = 1/100 ms = 1 / 0.1 sn = 10 Hz'dir.
- İşlemler yapılmadan önce birim saniyeye dönüştürülmelidir.

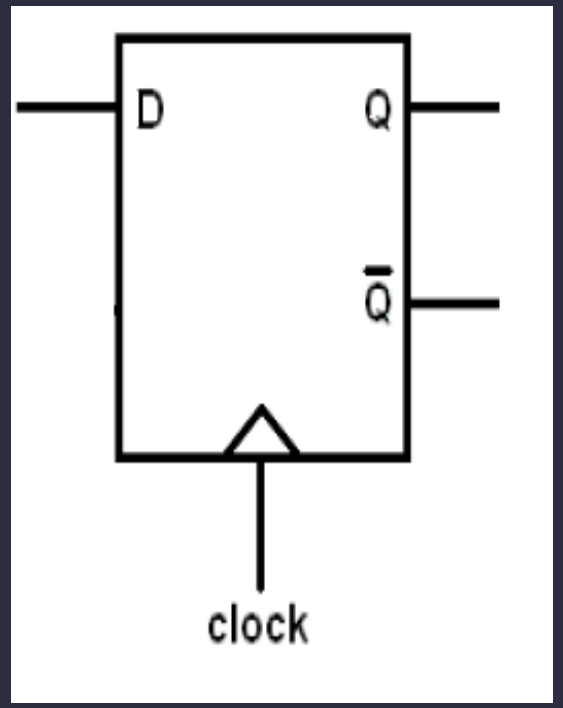
D Tipi Saklayıcı (D Register, Flip-Flop)

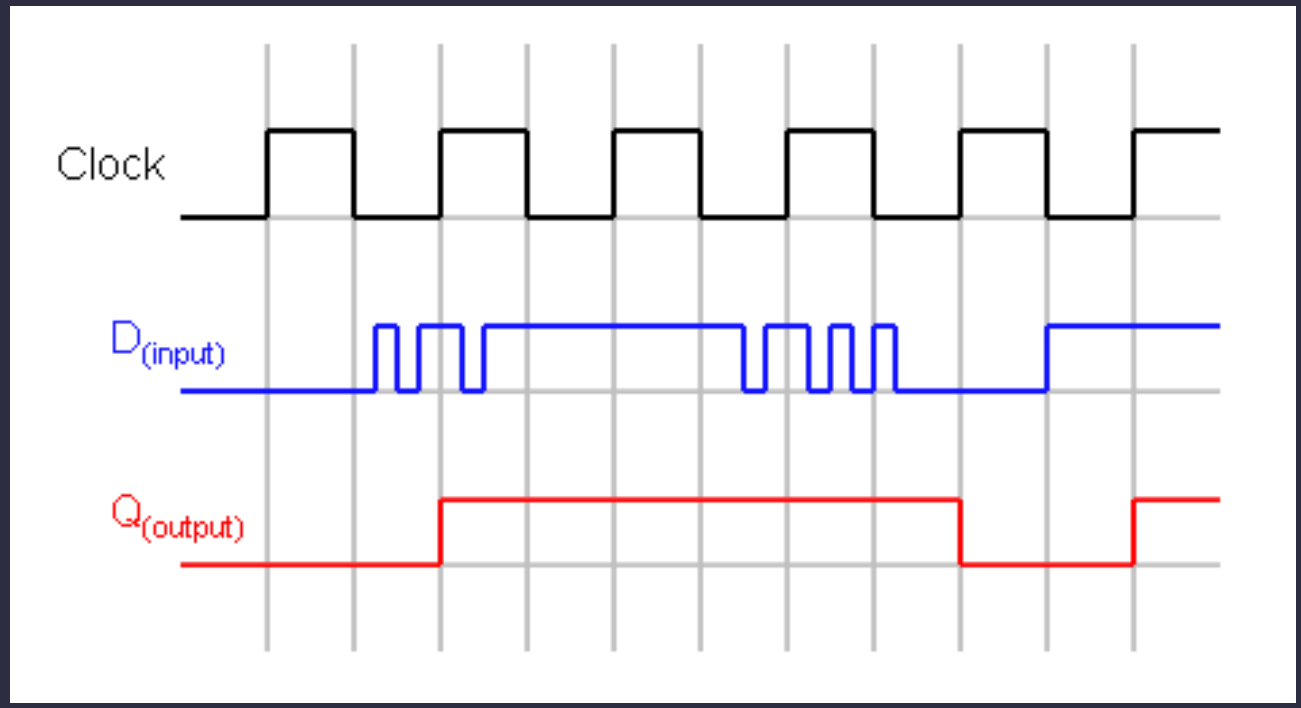
- Clock sinyalinin yükselen ya da alçalan kenarı kendisine geldiğinde D girişindeki, değer ne ise Q çıkışına aktarır.
- Diğer durumlarda Q çıkışındaki değer, D girişi değişse bile, değişmez.



D Tipi Saklayıcı (D Register, Flip-Flop)

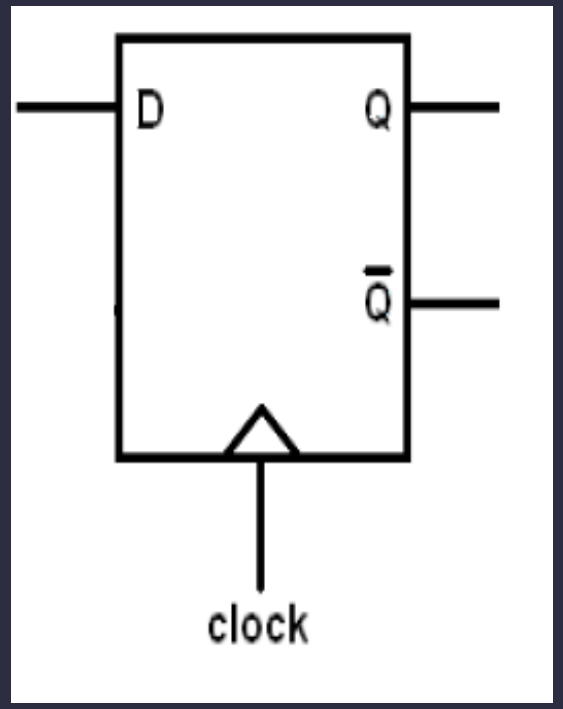
- Yükselen kenar D Tipi Saklayıcı Giriş ve Çıkışları

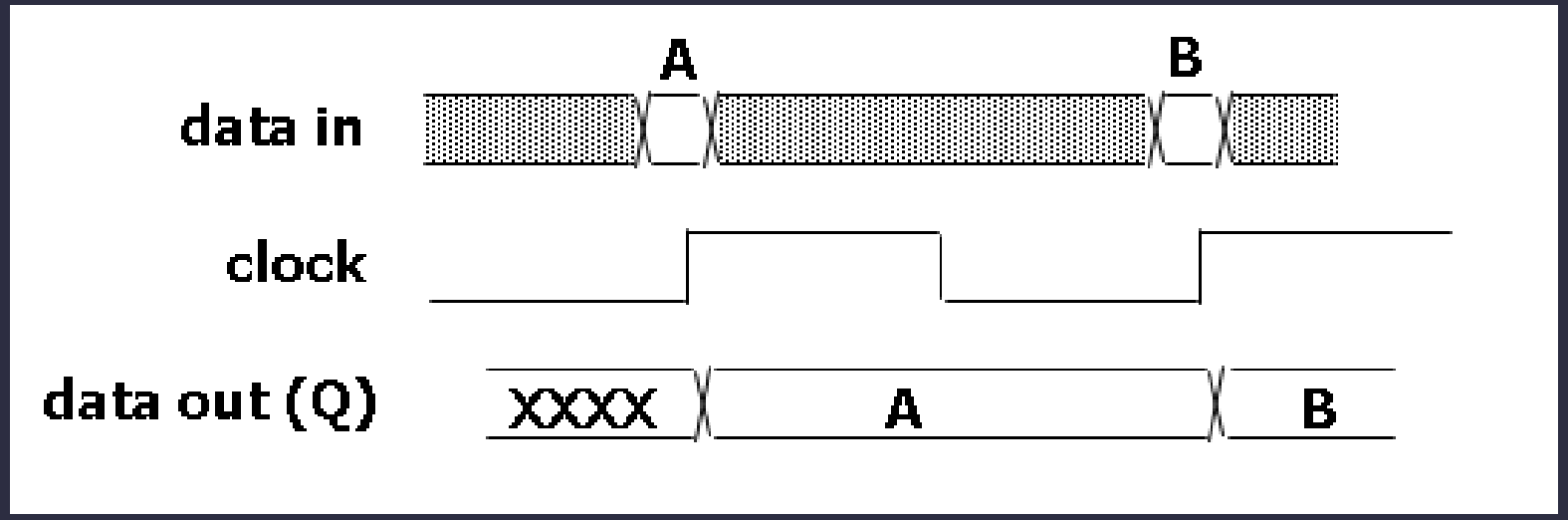




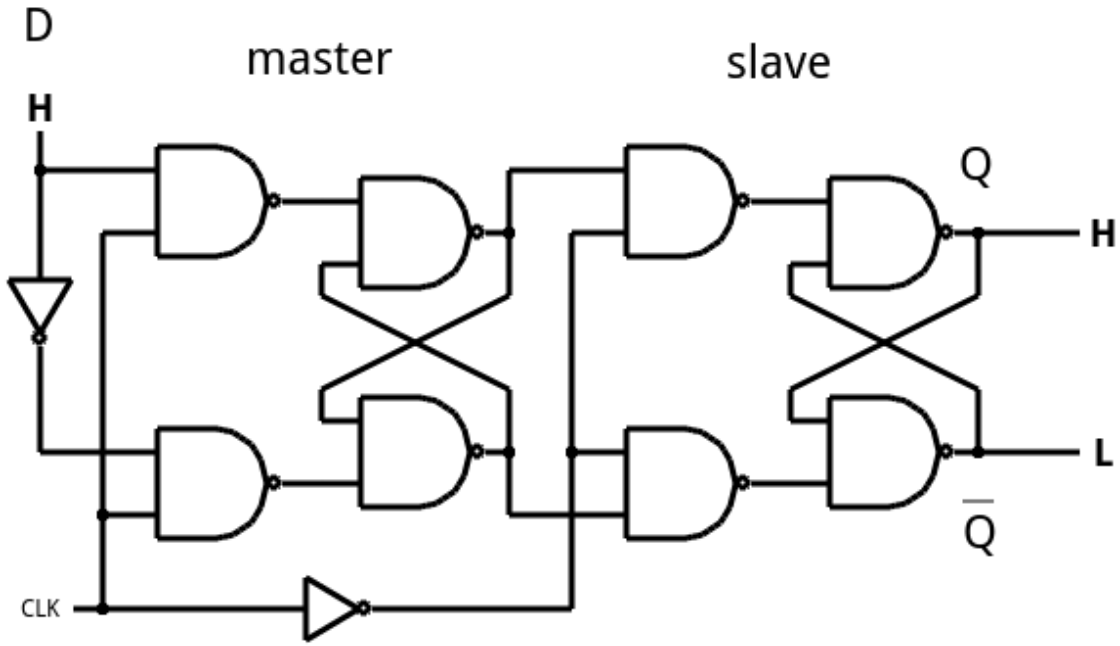
D Tipi Saklayıcı (D Register, Flip-Flop)

- Yükselen kenar D Tipi Saklayıcı Giriş ve Çıkışları





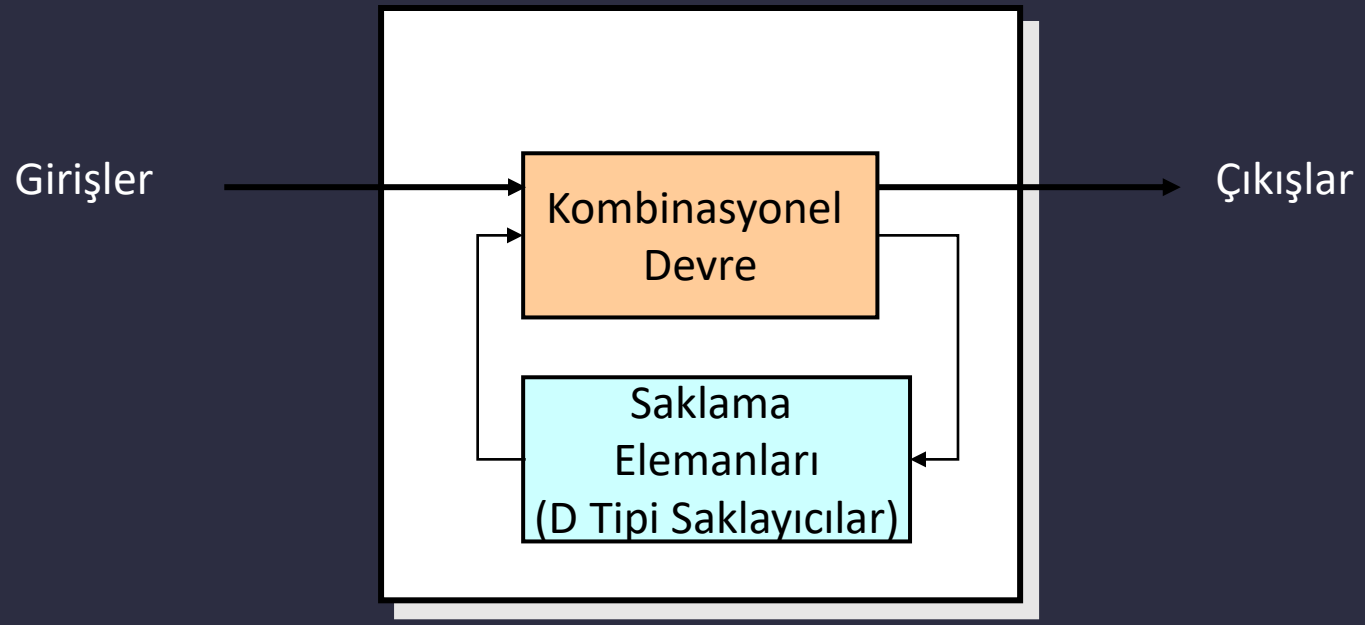
D Tipi Saklayıcı (D Register, Flip-Flop)



Düşen Kenar (Falling Edge) D tipi Saklayıcı

Ardışık Devreler

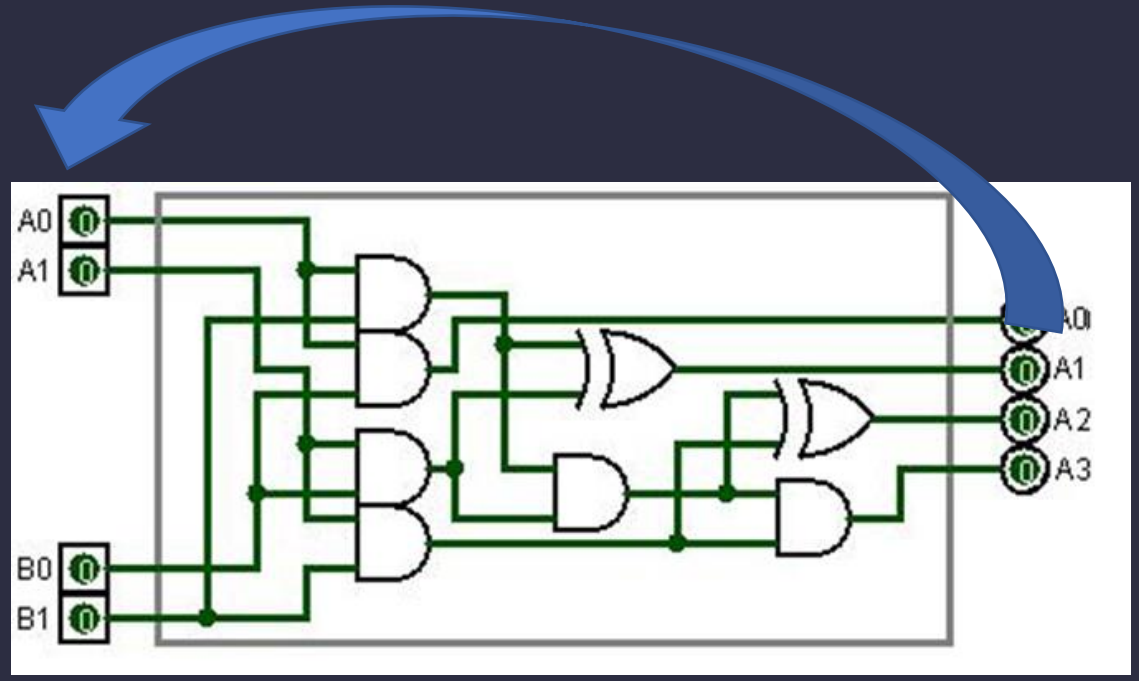
- Kombinasyonel devreler ve saklama elemanlarının bir araya getirilmesi ile oluşur.
- Saklama elemanlarının kullanılması ile devrenin ürettiği önceki değerler de kullanılabilmektedir.



Ardışık Devreler

- Ne zaman gereklidir?

- Bir devrenin ürettiği sonuç, yine devreye giriş olarak beslenecek ise;

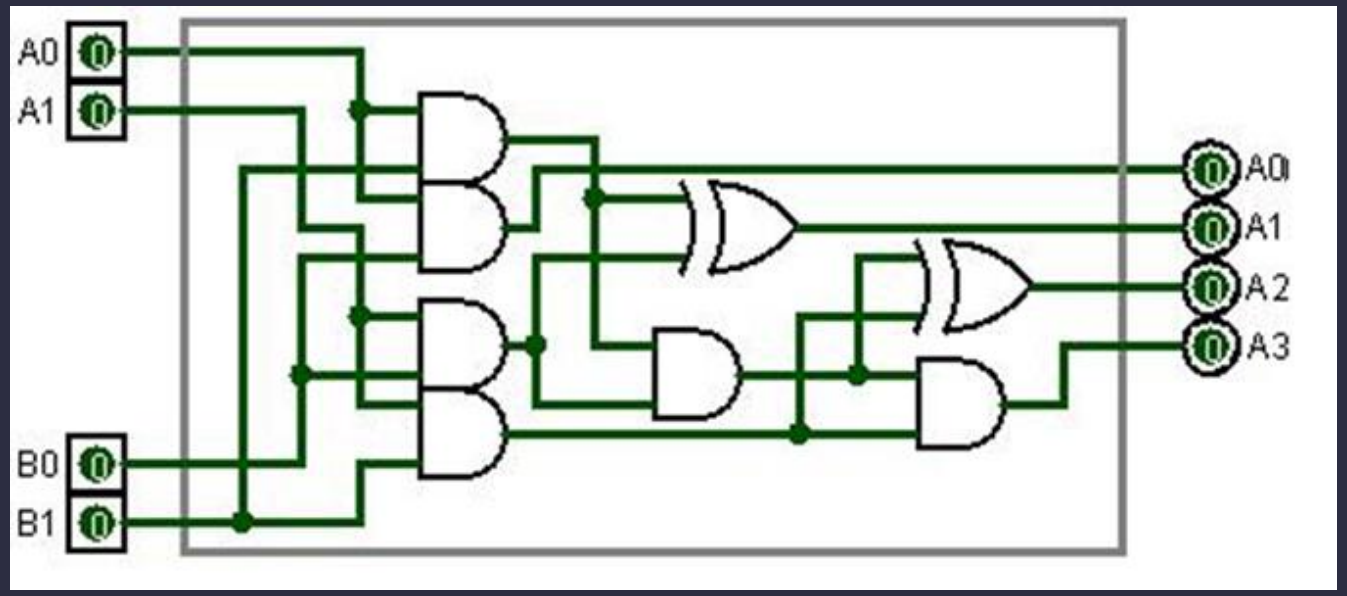


A ve B isimli 2 Bitlik iki sayının toplamını yapan devre

Ardışık Devreler

- Ne zaman gereklidir?

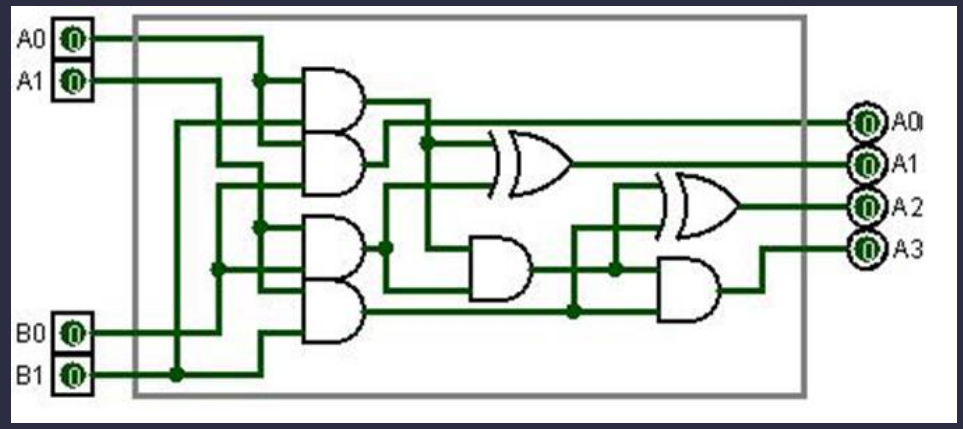
- Devrenin doğru sonuç üretebilmesi için, tüm girişlerin sabit kalması gerekmektedir.



Ardışık Devreler

- Ne zaman gereklidir?

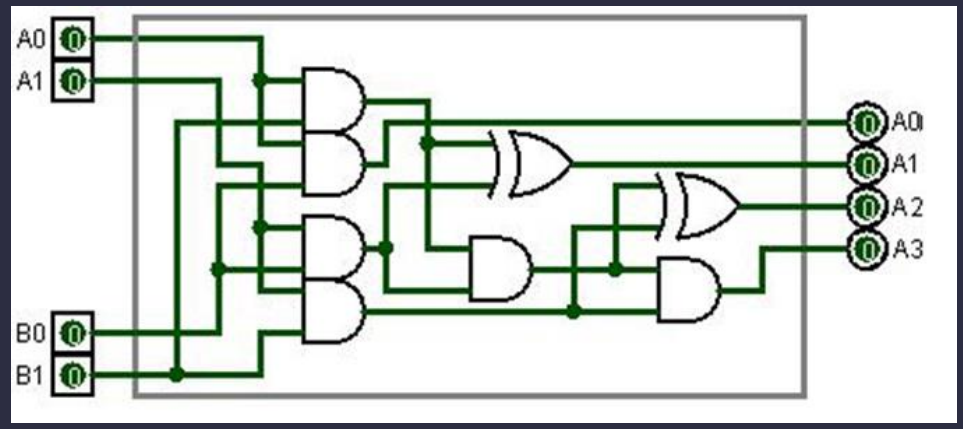
- Bu örnekte üretilen her bir bit farklı mantık kapıları yollarından gelmektedir. Gecikmeleri farklıdır. Dolayısıyla en hızlı hesaplaması yapılıp bitecek olan A0 çıktısı ile en uzun sürecek A2 çıktısının arasında gecikme farkı olacaktır.



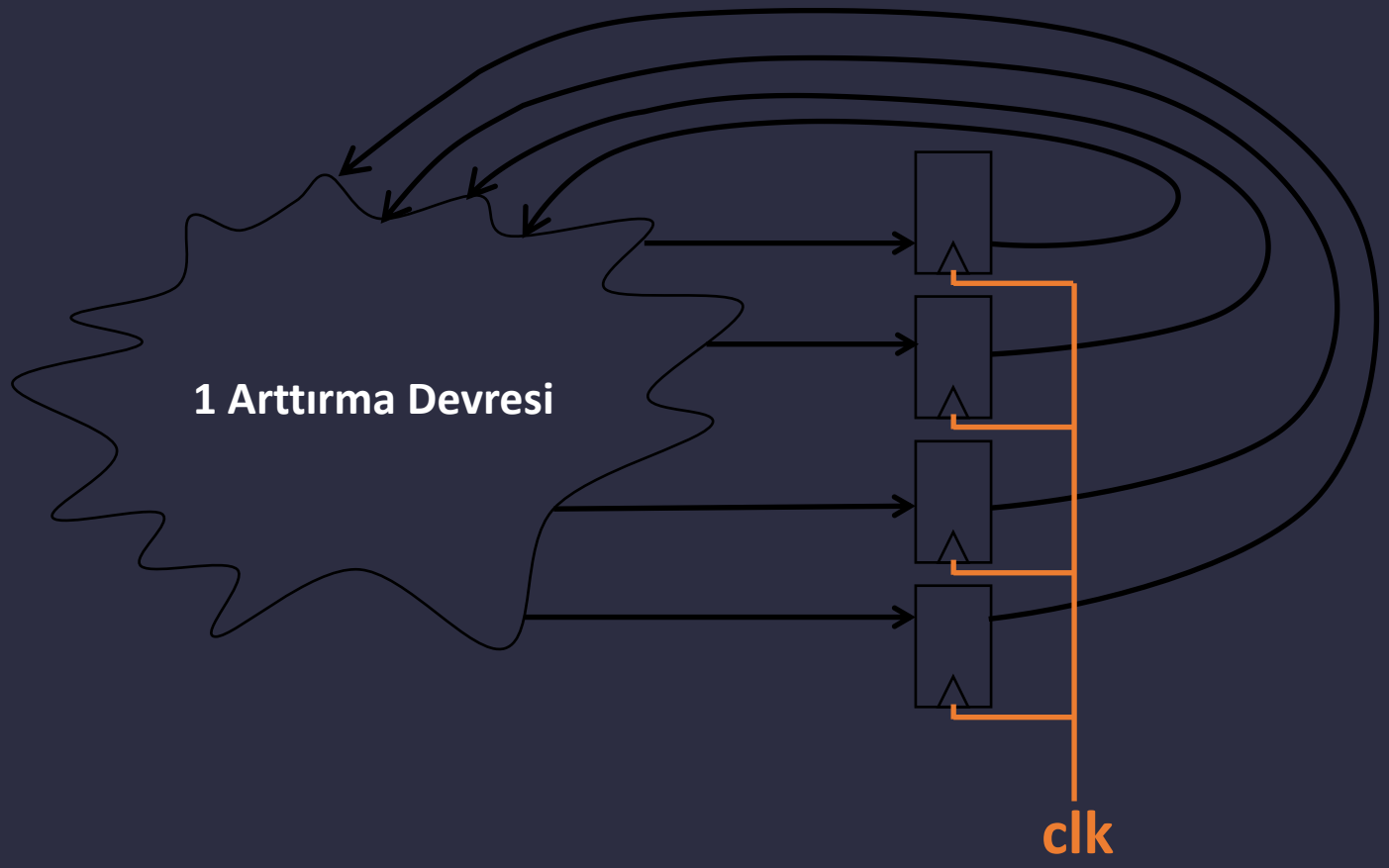
Ardışık Devreler

- Ne zaman gereklidir?

- Dolayısıyla çıktı sinyalleri, doğrudan girişe bağladığımızda, devre daha doğru sonuç üretemeden, girişi değiştirilmiş olur.
- Hiçbir zaman doğru sonuç elde edilemez.



Ardışık Devreler

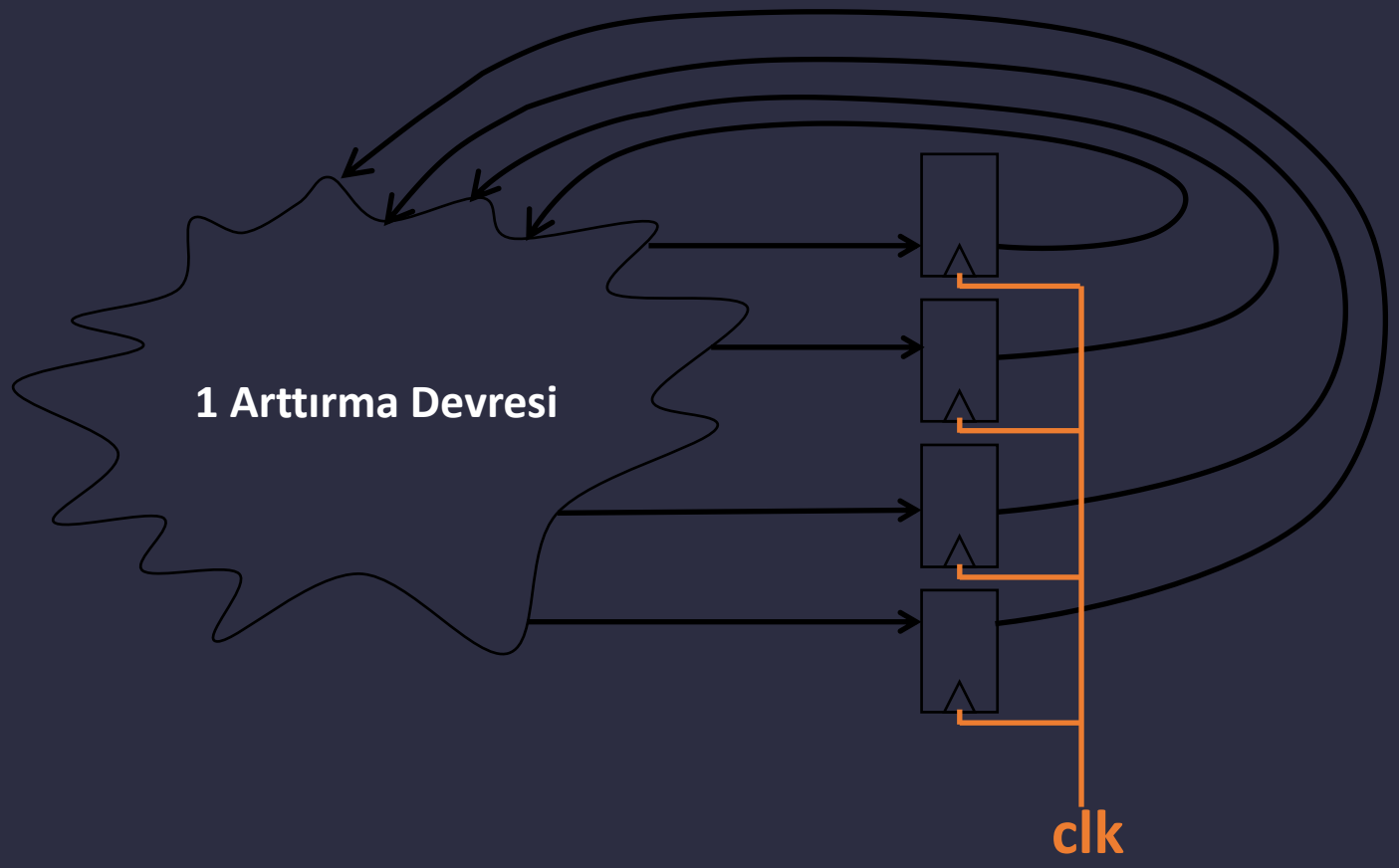


4 Bitlik 1 arttırma devresi.

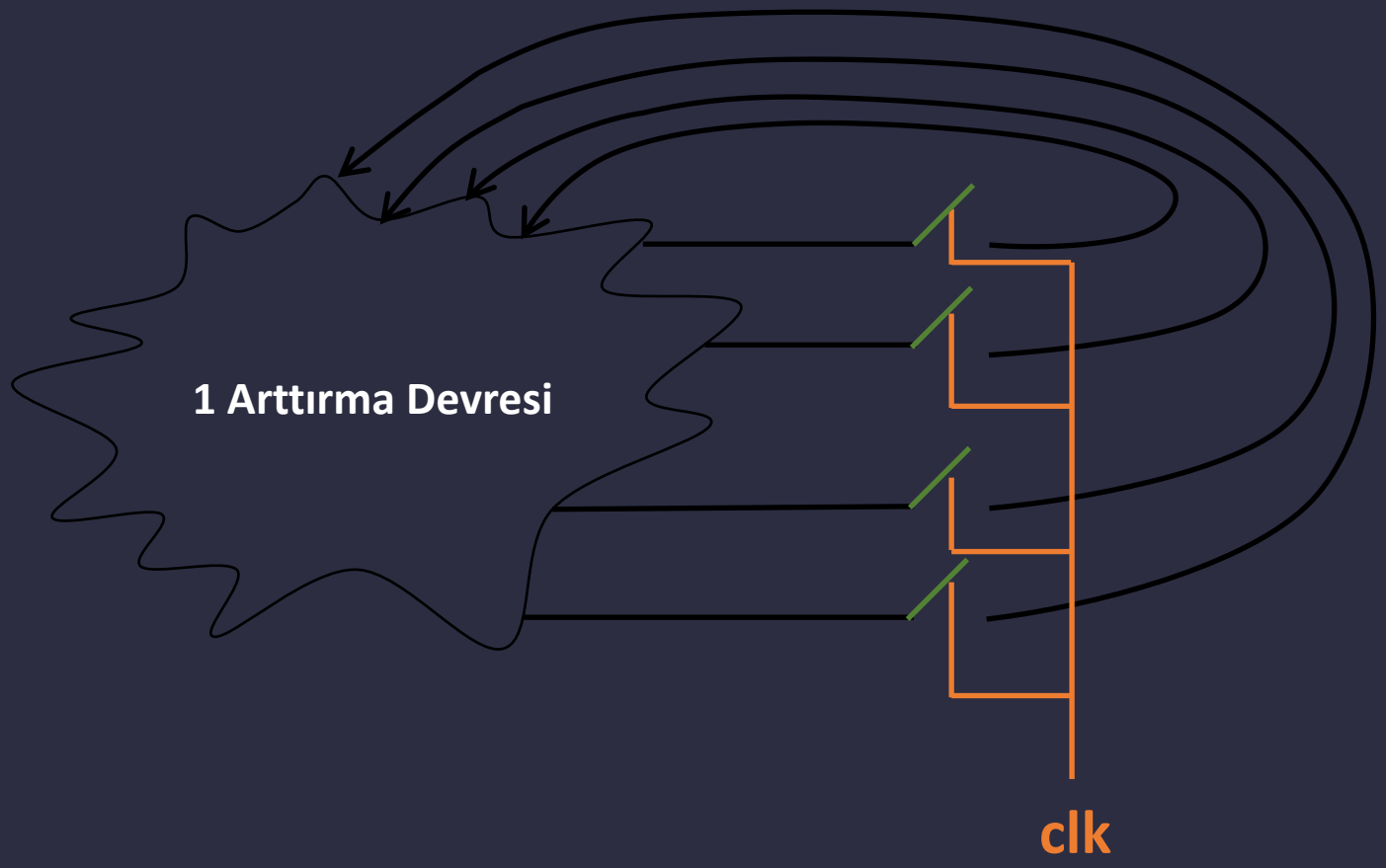
Ardışık Devreler

Clock girişinin periyodu, devrenin ürettiği en yavaş çıkış kadar olursa,

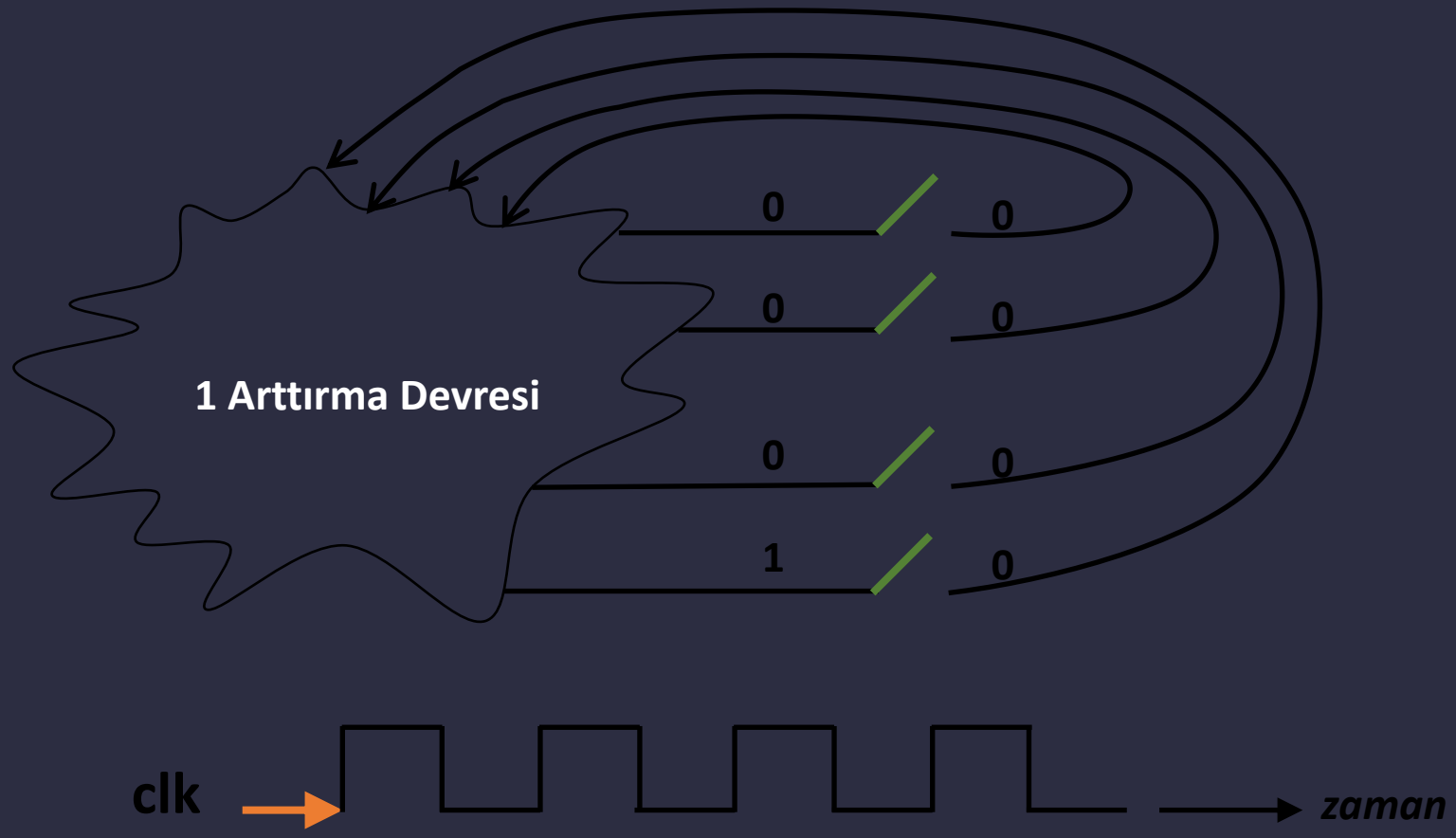
devrenin bütün çıkışları, en yavaş üretilen çıkış kadar bekletilerek, devreye tekrar beslenecektir.



Ardışık Devreler



Ardışık Devreler



Ardışık Devreler

Ardışık Devreler

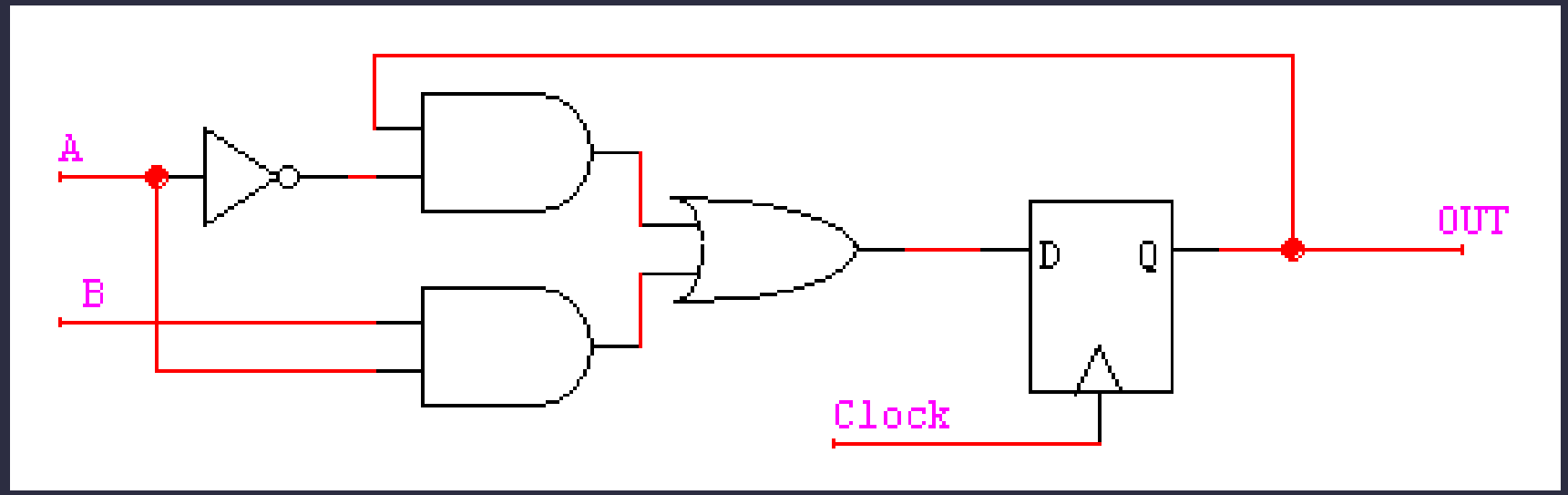
Ardışık Devreler

Ardışık Devreler

Ardışık Devreler

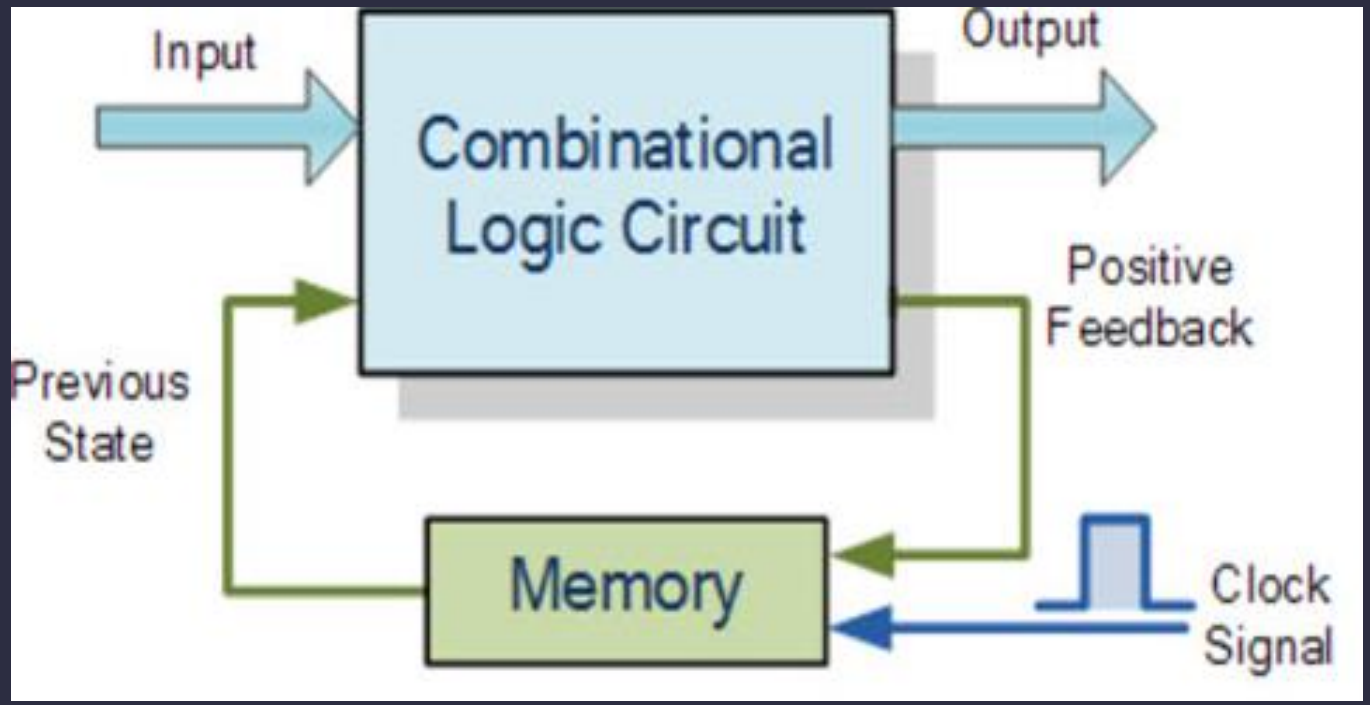
Ardışık Devreler

Ardışık Devreler



Kombinasyonel Devre ve Saklayıcı ile Ardışık Devre Örneği

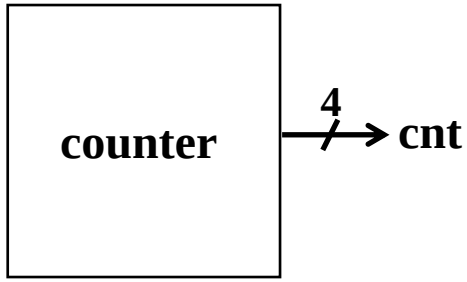
Ardışık Devreler



Ardışık Devreler

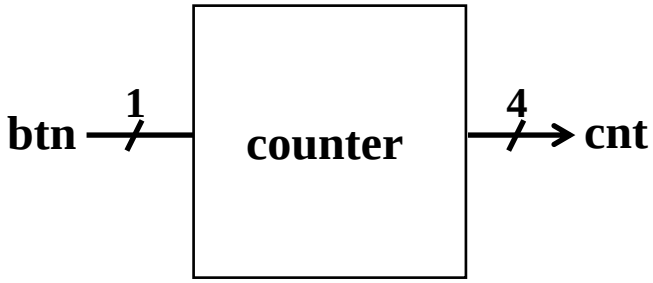

```
module counter();

endmodule
```

Ardışık Devreler



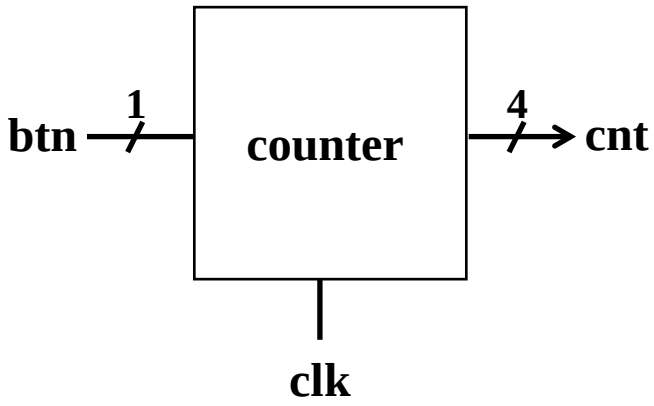

```
module counter(
   cnt
);
   output [3:0] cnt;

endmodule
```

Ardışık Devreler



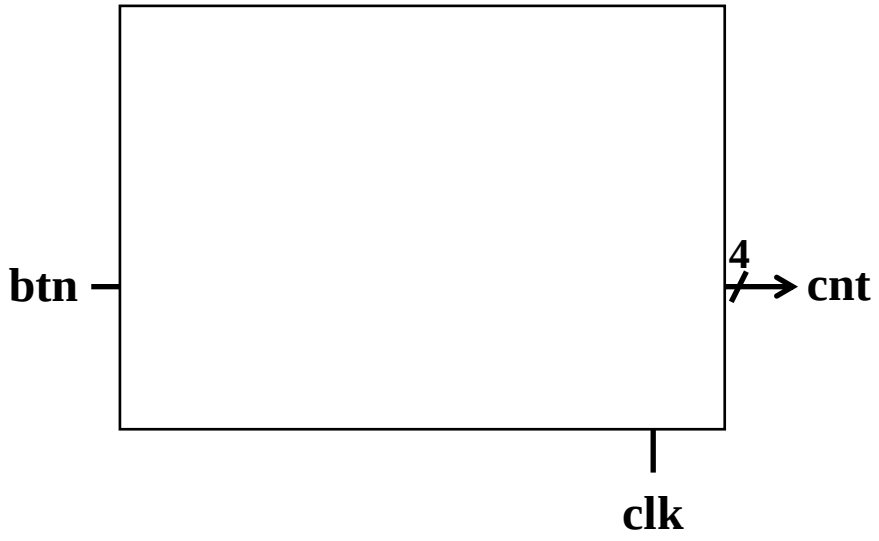

```
module counter(
   cnt,
   btn
);
   output [3:0] cnt;
   input btn;

endmodule
```

Ardışık Devreler



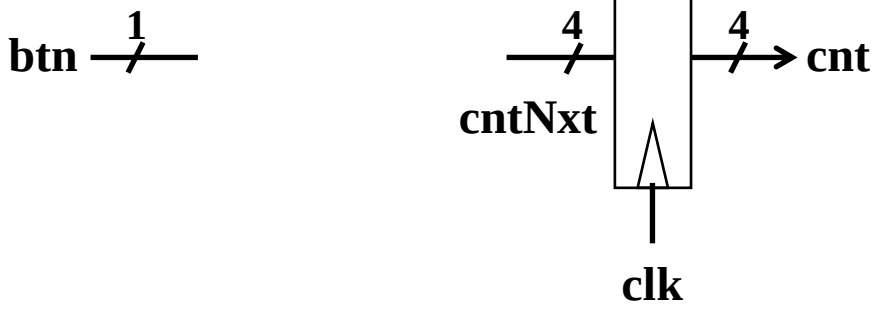

```
module counter(cnt, btn, clk);
   output [3:0] cnt;
   input btn, clk;

endmodule
```

Ardışık Devreler



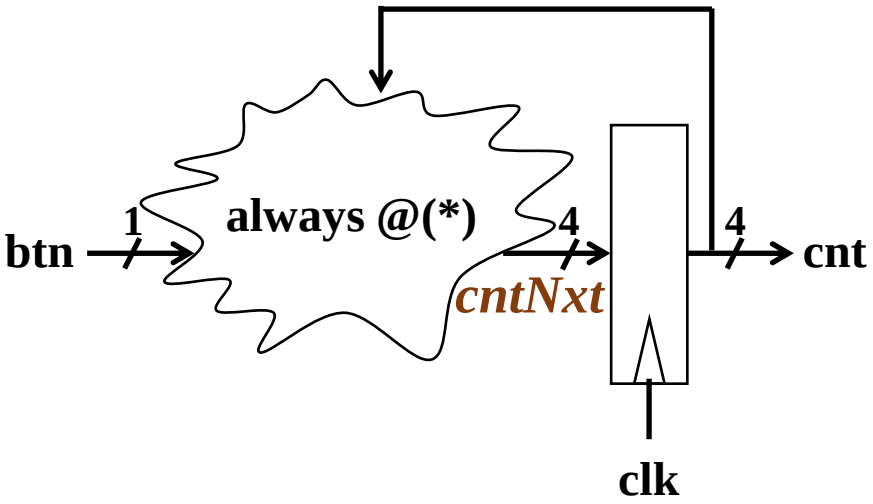

```
module counter(cnt, btn, clk);
   output [3:0] cnt;
   input btn, clk;

endmodule
```

Ardışık Devreler



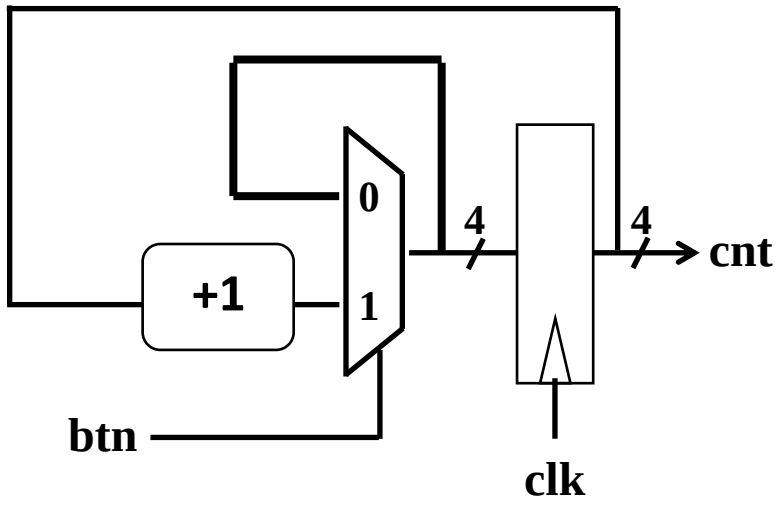

```
module counter(cnt, btn, clk);
   output [3:0] cnt;
   input btn, clk;

   reg [3:0] cnt, cntNxt;

    always @(posedge clk) begin
       cnt <= #1 cntNxt;
    end

endmodule
```

Ardışık Devreler



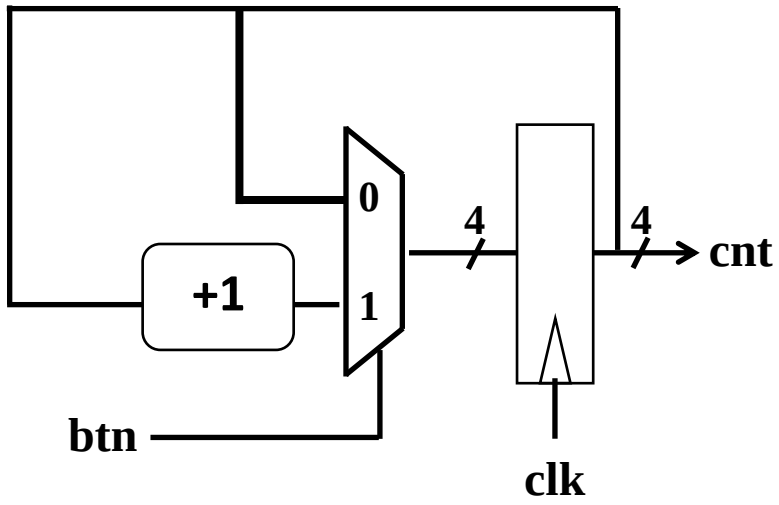

```
module counter(cnt, btn, clk);
   output [3:0] cnt;
   input btn, clk;

   reg [3:0] cnt, cntNxt;

    always @(posedge clk) begin
       cnt <= #1 cntNxt;
    end

    always @(*) begin
       if(btn)
          cntNxt = cnt +1;
    end

endmodule
```

Ardışık Devreler




```
module counter(cnt, btn, clk);
   output [3:0] cnt;
   input btn, clk;

   reg [3:0] cnt, cntNxt;

    always @(posedge clk) begin
       cnt <= #1 cntNxt;
    end

    always @(*) begin
       if(btn)
          cntNxt = cnt +1;
    end

endmodule
```

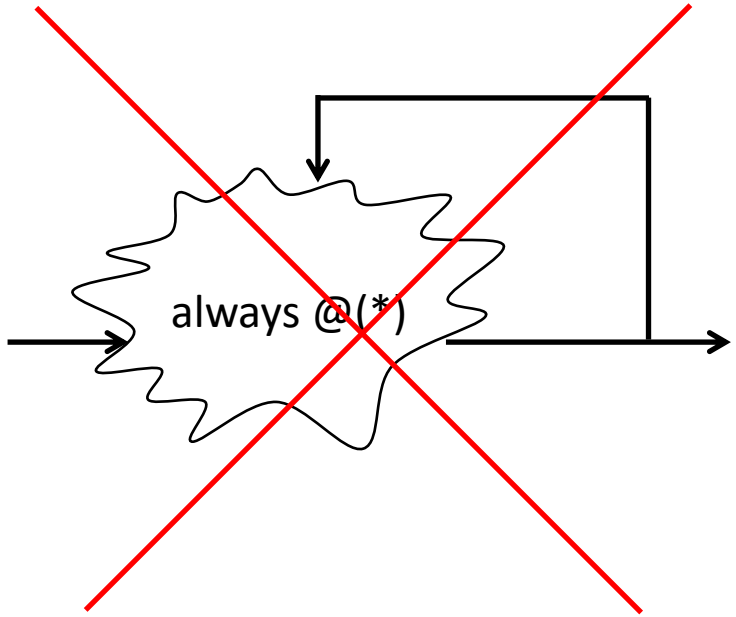
Ardışık Devreler




```
module counter(cnt, btn, clk);
   output [3:0] cnt;
   input btn, clk;

   reg [3:0] cnt, cntNxt;

    always @(posedge clk) begin
       cnt <= #1 cntNxt;
    end

    always @(*) begin
       cntNxt = cnt;
       if(btn)
          cntNxt = cnt +1;
    end

endmodule
```

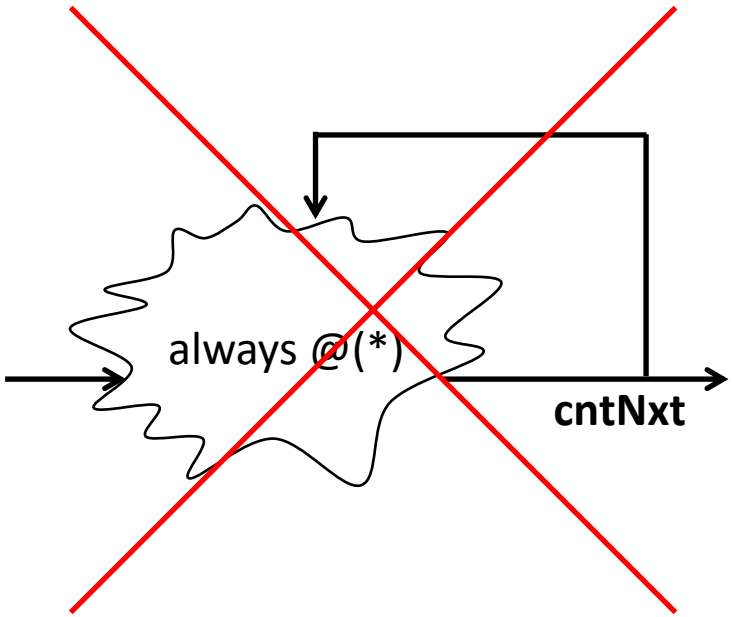
Ardışık Devreler

Kombinasyonel Döngü Yapmayın



Ardışık Devreler

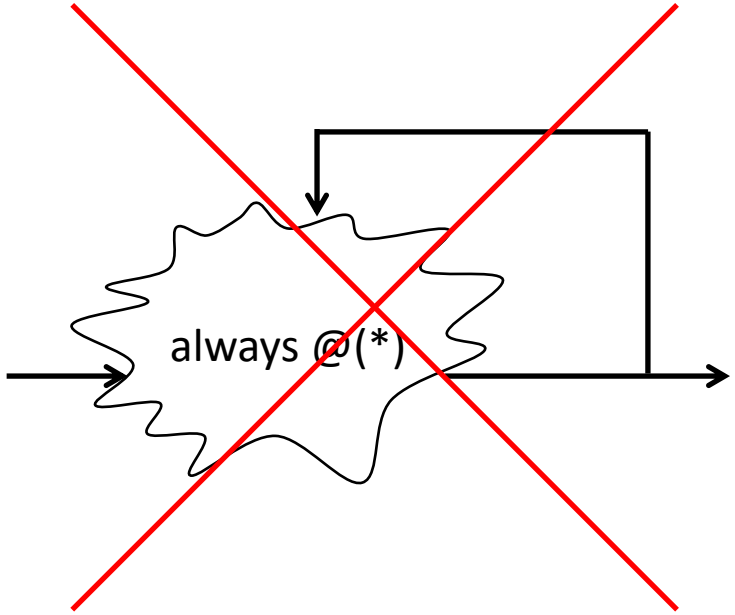
Kombinasyonel Döngü Yapmayın




```
always @(*) begin
    if(cntNxt)
        cntNxt = cnt –1;
end
```

Ardışık Devreler

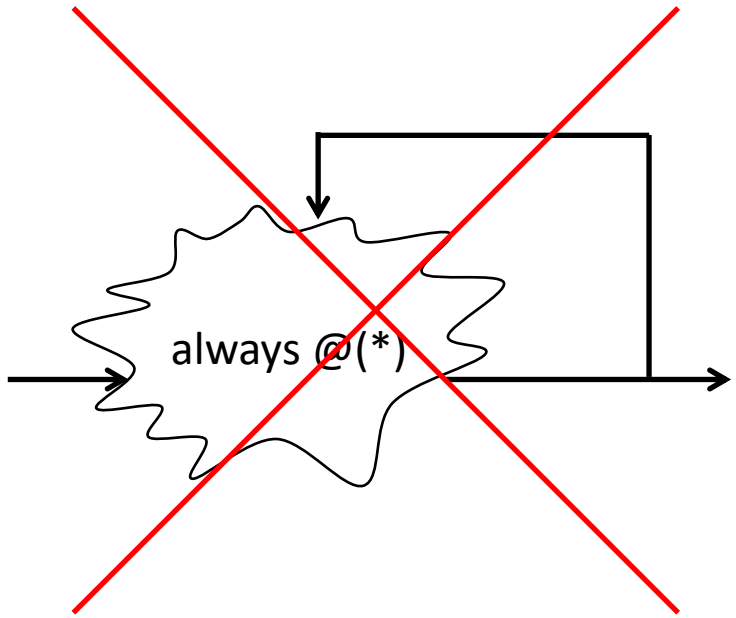
Kombinasyonel Döngü Yapmayın

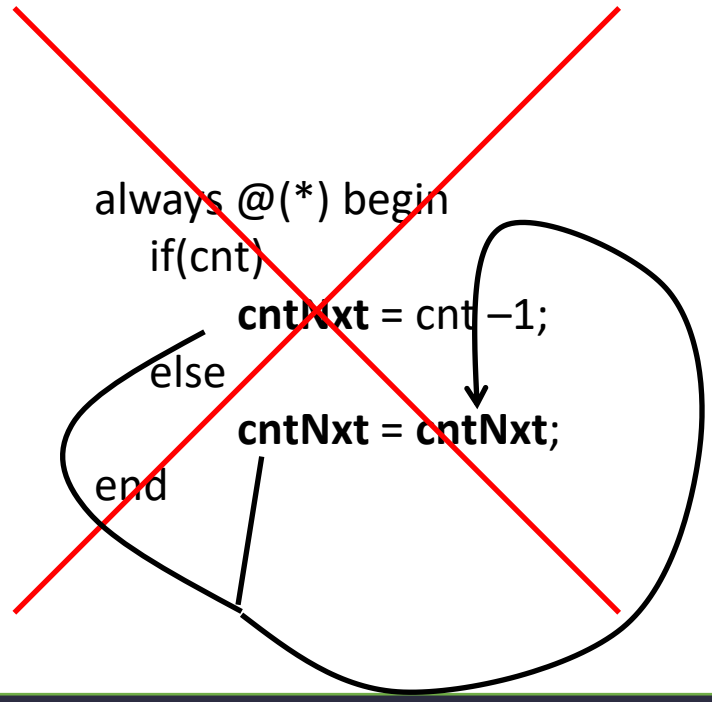



```
always @(*) begin
    if(cnt)
        cntNxt = cnt –1;
end
```

Ardışık Devreler

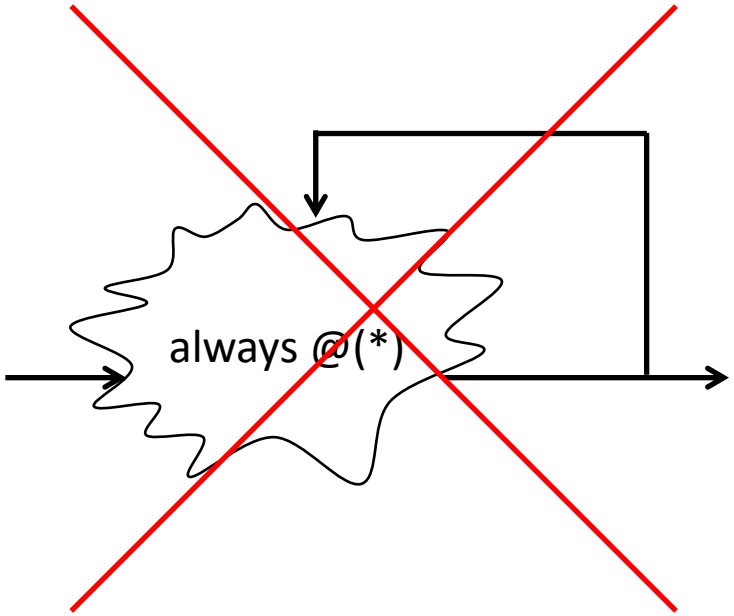
Kombinasyonel Döngü Yapmayın





Ardışık Devreler

Kombinasyonel Döngü Yapmayın



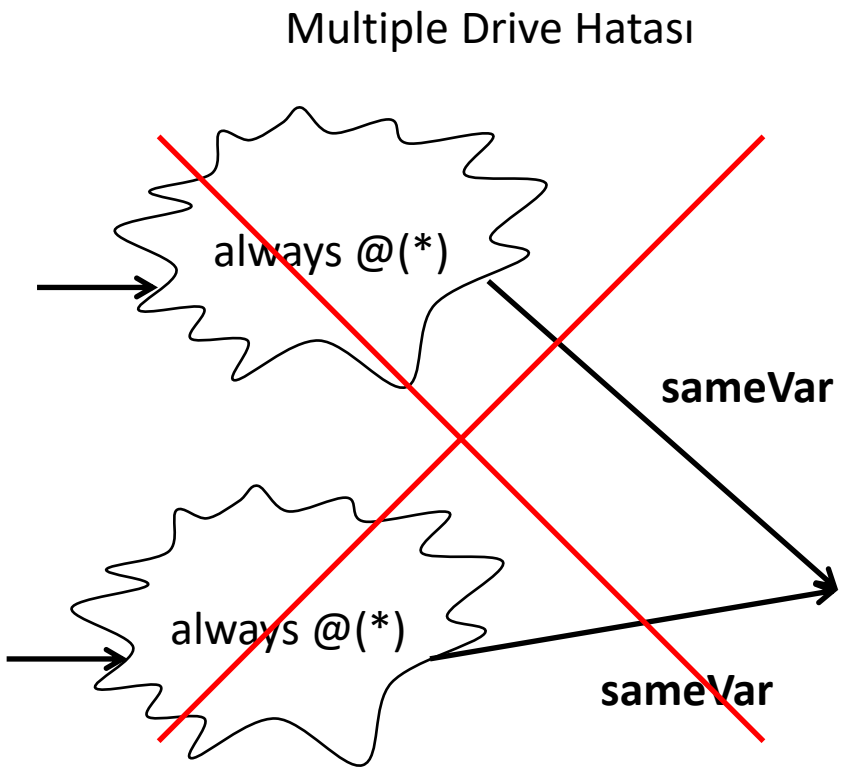

```
always @(*) begin
    if(cnt)
        cntNxt = cnt –1;
    else
        cntNxt = cnt;
end
```

Ardışık Devreler

Kombinasyonel Döngü Yapmayın

```
always @(*) begin
    cntNxt = cnt;
    if(cnt)
        cntNxt = cnt –1;
end
```

Ardışık Devreler



Ardışık Devreler

Multiple Drive Hatası

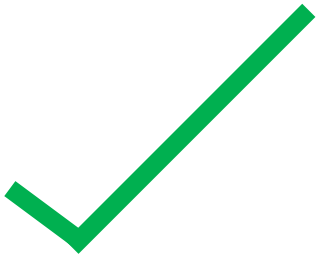
```
always @(*) begin
    cntNxt = cnt;
    if(btn1)
        cntNxt = cnt +1;
end

always @(*) begin
    cntNxt = cnt;
    if(btn2)
        cntNxt = cnt –1;
end
```

Ardışık Devreler

Multiple Drive Hatası

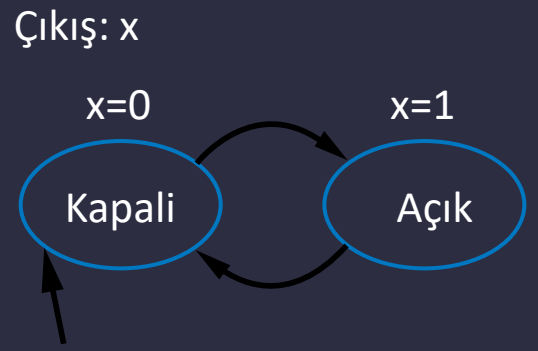
// Tek bir always bloğunda birleştirildi

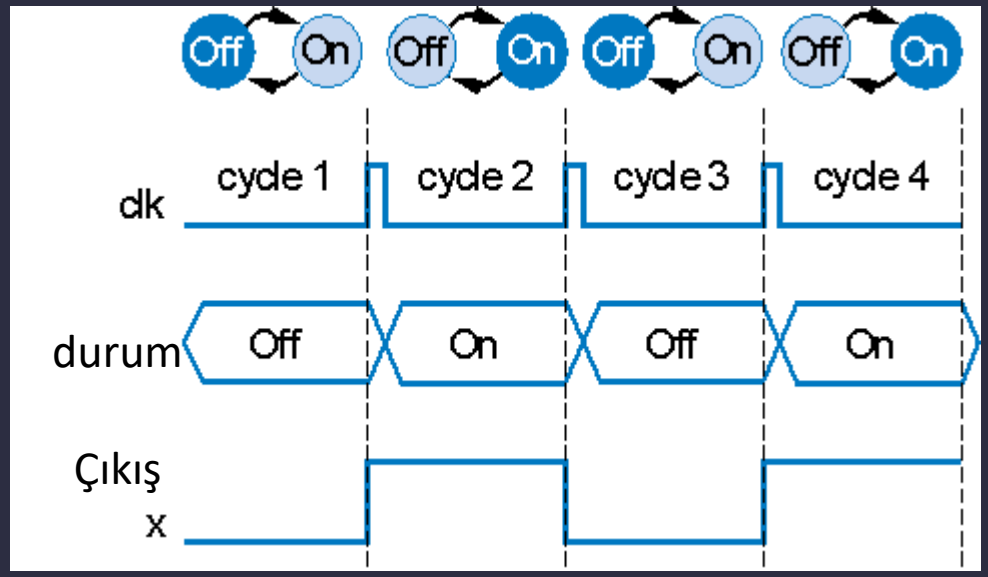
```
always @(*) begin
    cntNxt = cnt;
    if(btn1)
        cntNxt = cnt +1;
    if(btn2)
        cntNxt = cnt –1;
end
```



Ardışık Devrenin Davranışını Belirleme FSM

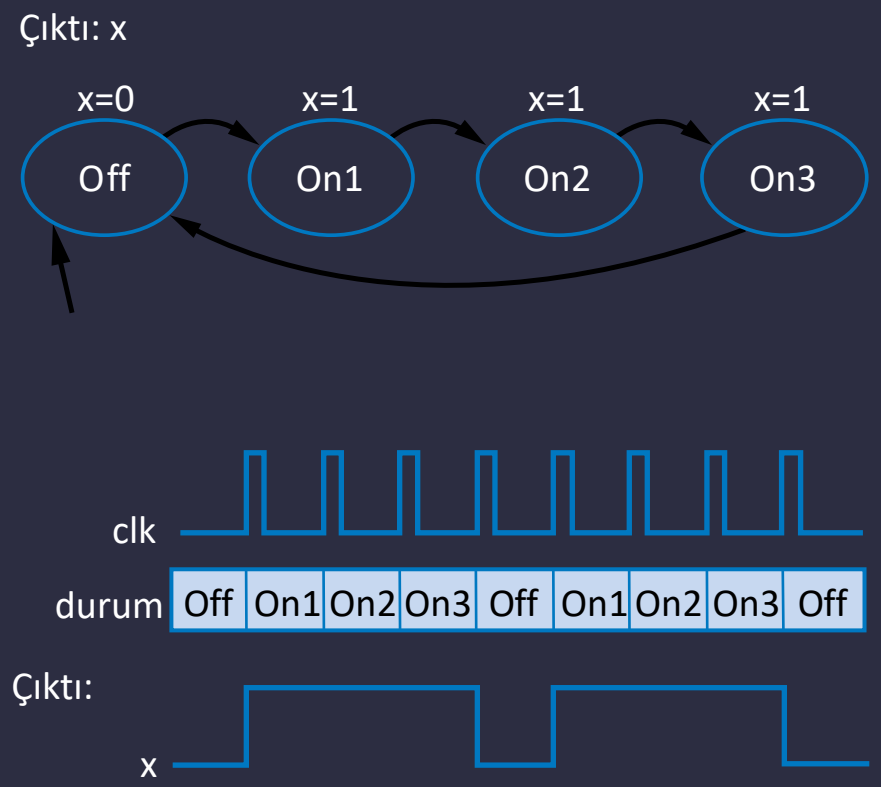
- Finite-State Machine (FSM)
  - Ardışık devrenin durumlara göre davranışının gösterim biçimidir
  - Durumları ve durumlar arası geçişler listelenir
    - Örnek: X isminde bir çıkış sinyali olsun ve her clock cycle'da değeri değişsin
    - İki durum: "Off" (x=0), ve "On" (x=1)
    - Off durumundan On durumuna ve On durumundan Off durumuna geçişler bulunmaktadır.
    - Başlangıç durumu bir ok ile belirtilir.





FSM Örneği: 0,1,1,1, tekrarı

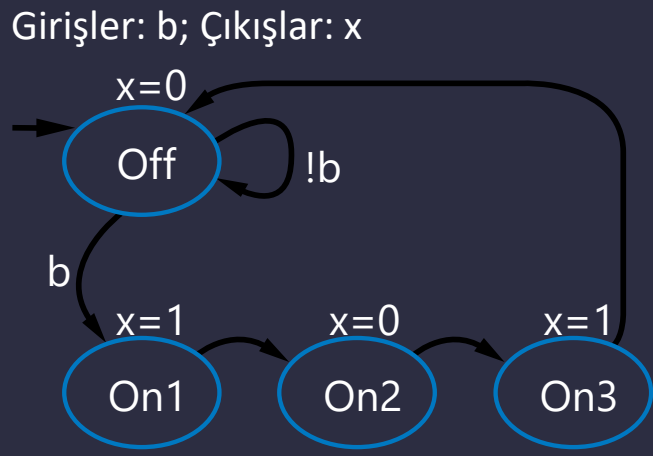
- Sırayla 0, 1, 1, 1, 0, 1, 1, 1, ... cıktılarını veren bir devre tasarlanacaktır.
  - Her bir değer bir clock cycle'da çıkmaktadır.
- FSM ile tasarlanabilir
  - 4 durum bulunur

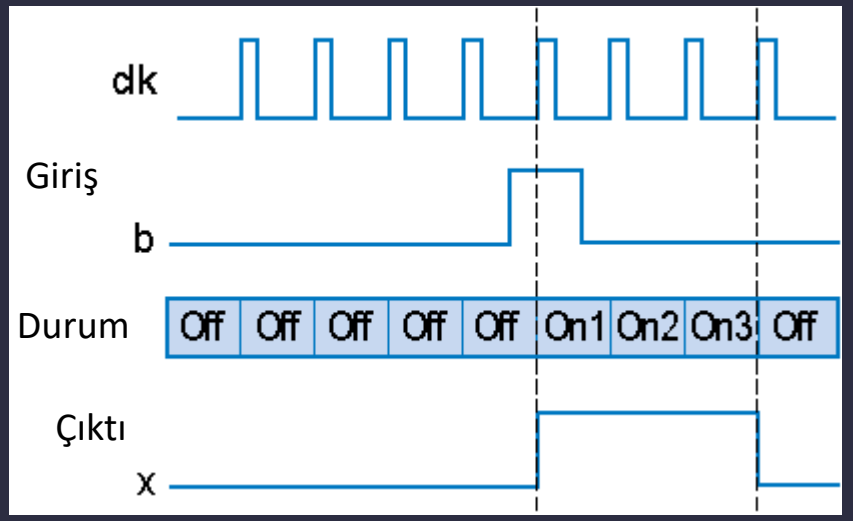


FSM Örneği

b isminde bir bitlik giriş alıp,

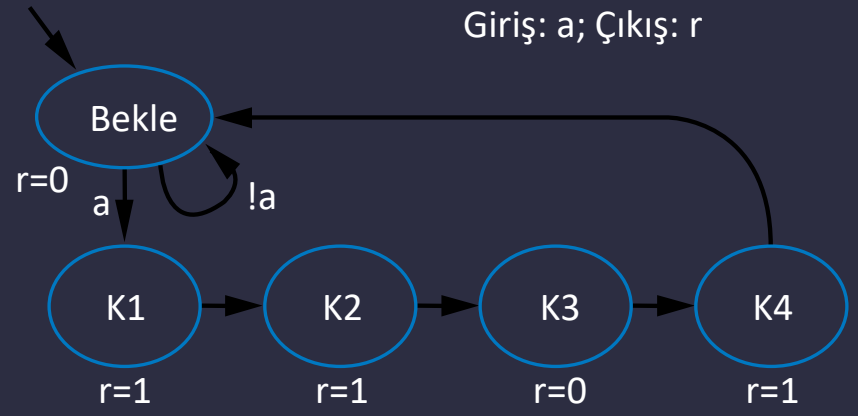
b girişi aktif olduğunda 101 pattern'ini çıktı verip, tekrar b girişini bekleyen bir durum makinası tasarımı yapınız.





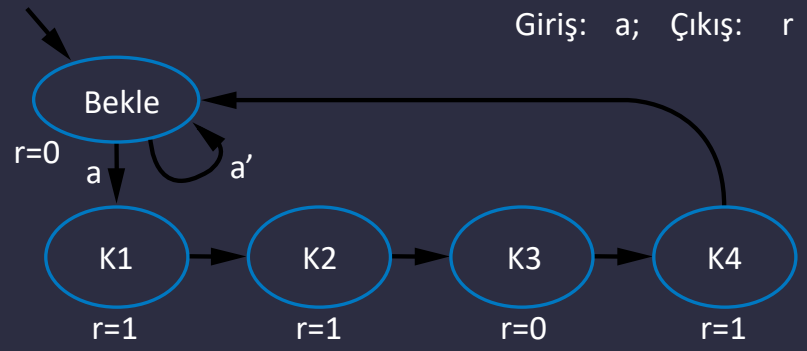
FSM Örneği: Araba Anahtarı

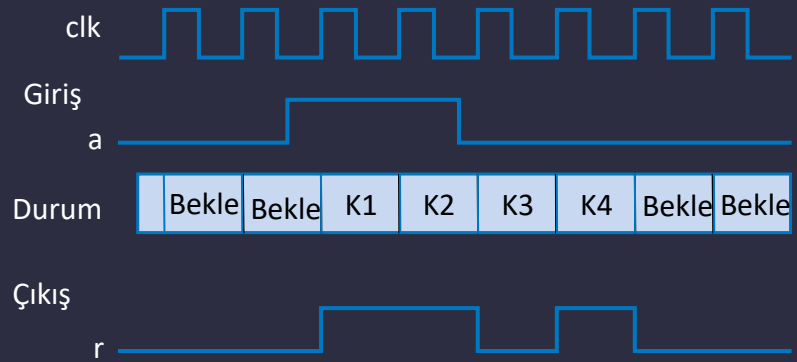
- Araba anahtarlarının içerisinde anahtarın kimliğini tutan çip bulunur.
  - Arabaya anahtar takıldığında, anahtardan kimlik bilgisini ister.
  - Anahtar kimlik bilgisini gönderir, doğru değilse araç çalışmaz
- FSM
  - İstek yapılmasını bekle (a=1)
  - Kimlik'i gönder (Örneğin, 1101)

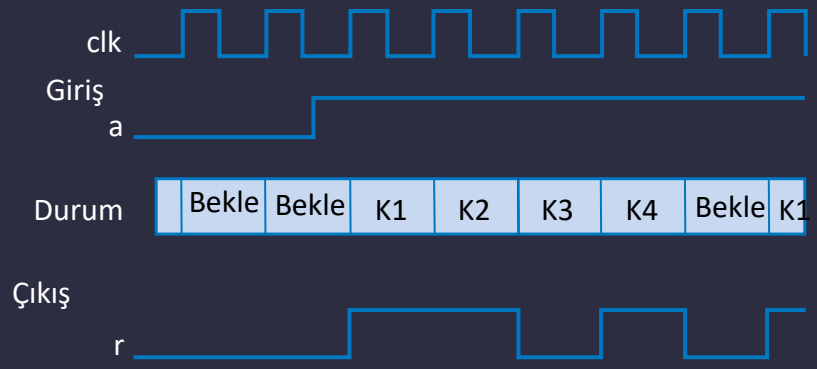


FSM Örneği: Araba Anahtarı

- FSM Zamanlama'ları

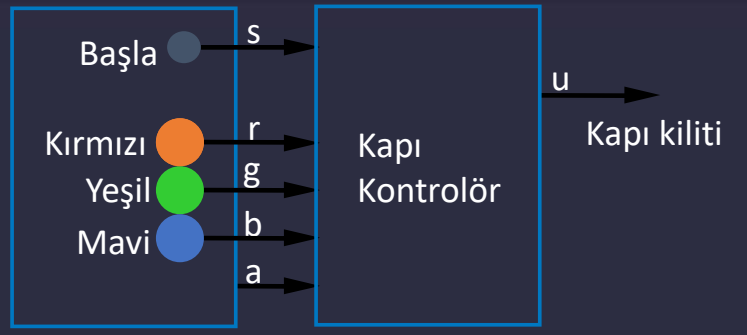


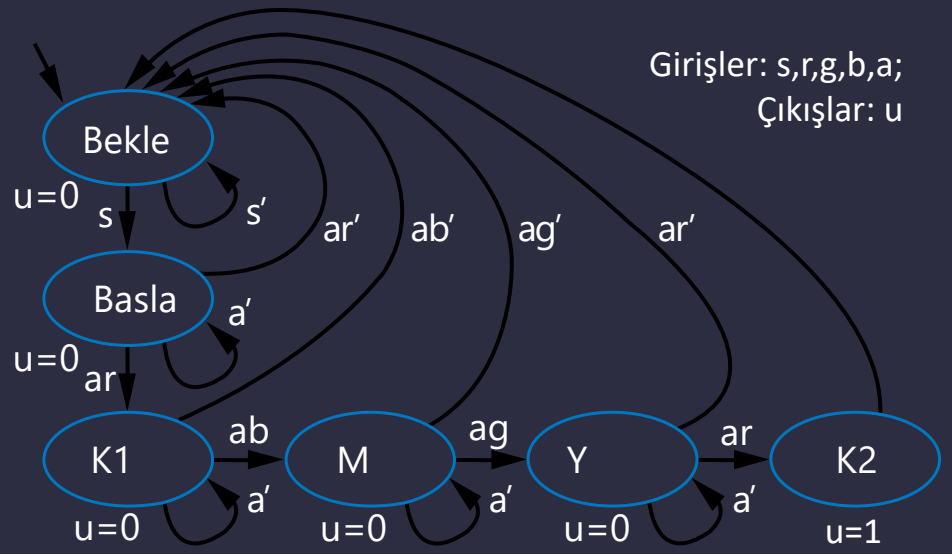




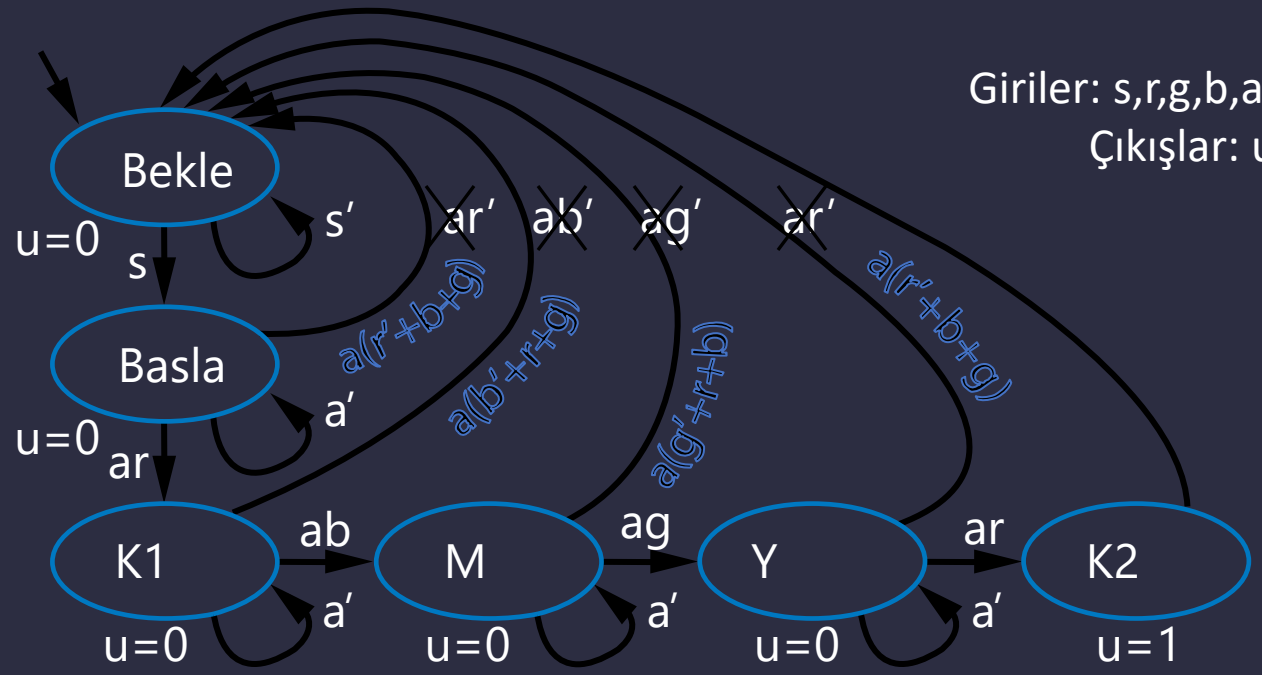
FSM Örneği: Kod Tespit Edici

- Bir kapı sadece aşağıdaki buton sıralamasına basıldığı zaman açılmaktadır (u=1)
  - başla, kırmızı, mavi, yeşil, kırmızı
- *s, r, g, b*
  - a girişi ise, diğer renkteki butonlar
- FSM
  - Bekle durumunda iken, başlat geldiğinde başla
  - Başlata basıldığında
    - Eğer kırmızı ise, K1'e
    - Sonra, mavi ise, M
    - Sonra, yeşil ise, Y
    - Sonra, kırmızı ise, K2
      - Bu durumda kapı açılacak sinyal üretilir (u=1)
    - Herhangi yanlış bir girişte bekle durumuna geri dönülür





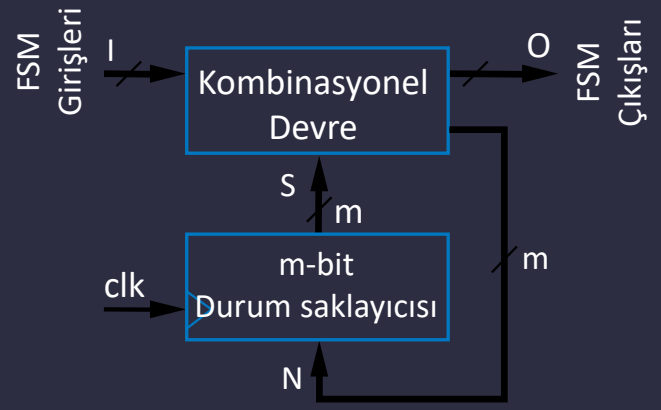
FSM Örneği: Kod Tespit Edici, İyileştirmeler



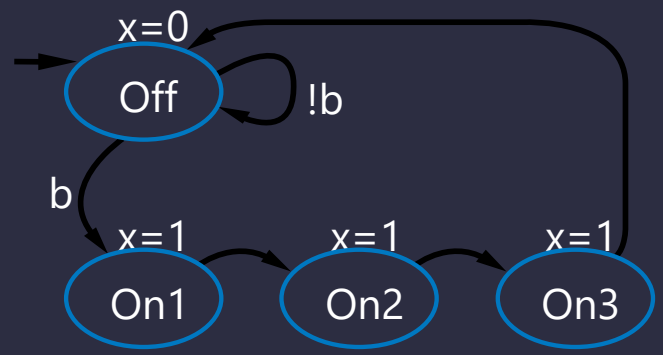
Yeni durum makinasında, yanlış butona basıldığında, "Bekle" durumuna geri dönülmektedir.

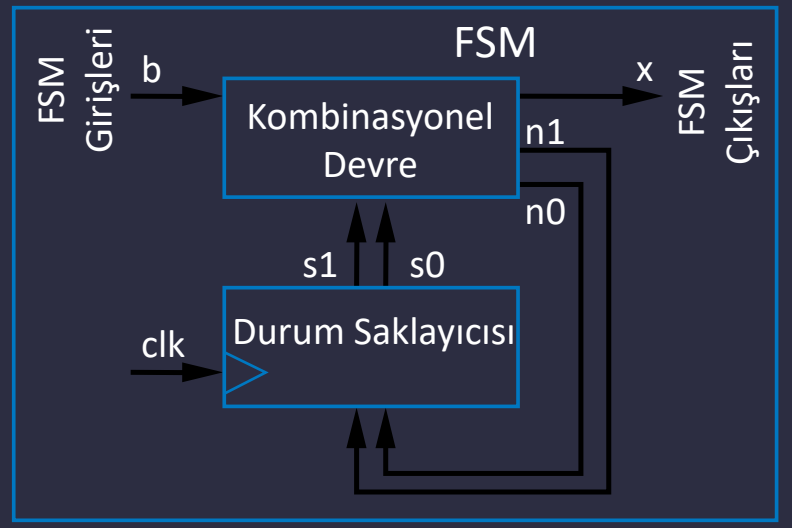
FSM Mimarisi

- FSM ardışık devre olarak nasıl gerçeklenir?
  - Durum saklayıcısı - şu anki durumu tutmak için
  - Kombinasyonel devre – çıkışı ve sonraki durumu hesaplamak için



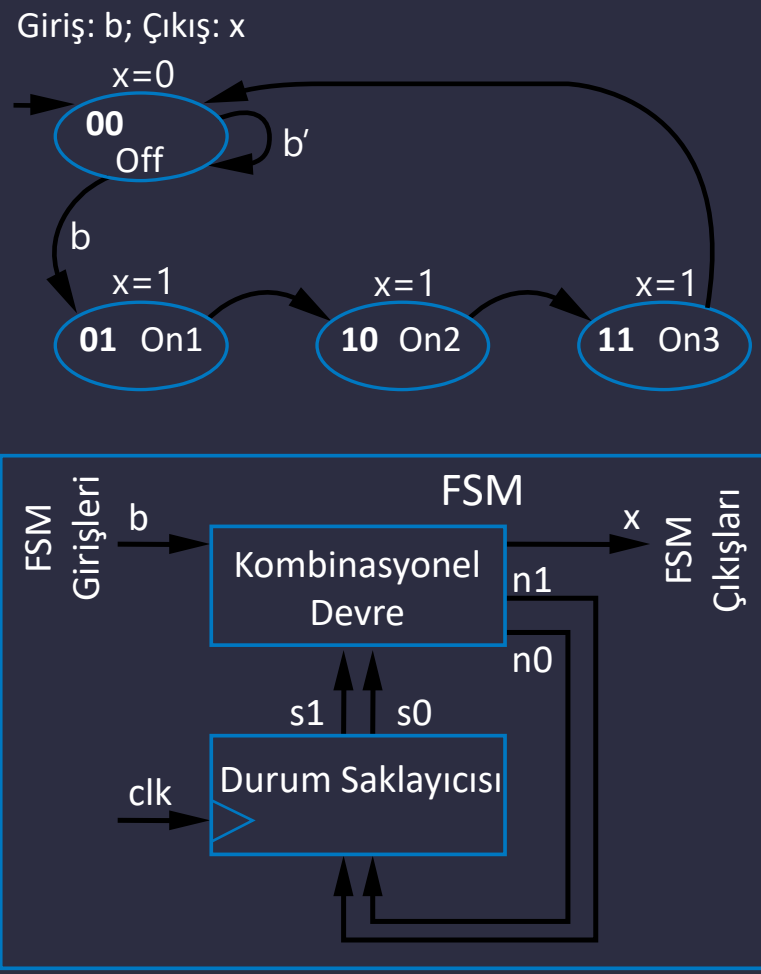
Giriş: b; Çıkış: x





FSM Tasarım Örneği

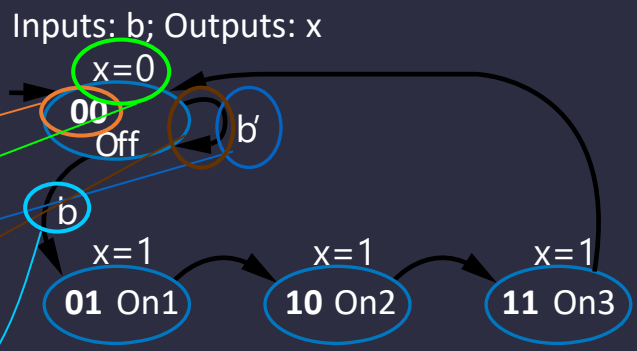
- Adım 1: Gereksinimleri analiz
- Adım 2: Mimari çıkart
  - 2-bit durum saklayıcısı (4 durum için)
  - Giriş b, çıkış x
  - Sonraki durum sinyalleri n1, n0
- Adım 3: State'leri kodla

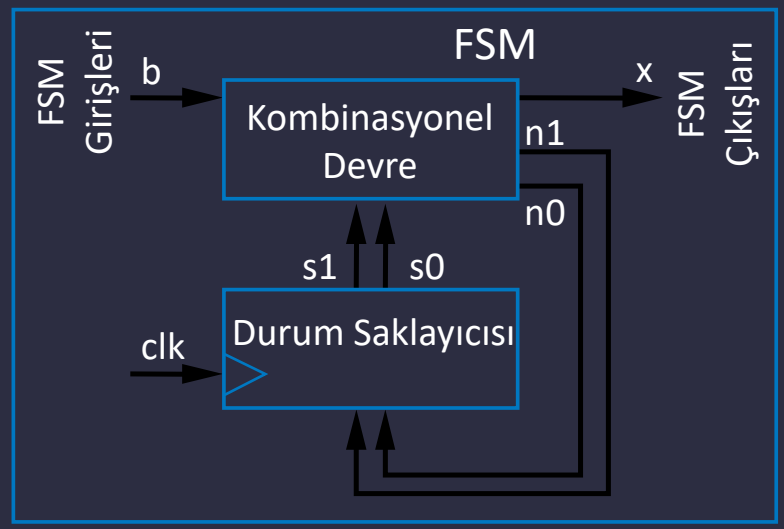


FSM Tasarım Örneği

- Adım 4: Doğruluk Tablosu oluştur

	Girişler			Çıkışlar		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	0
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0

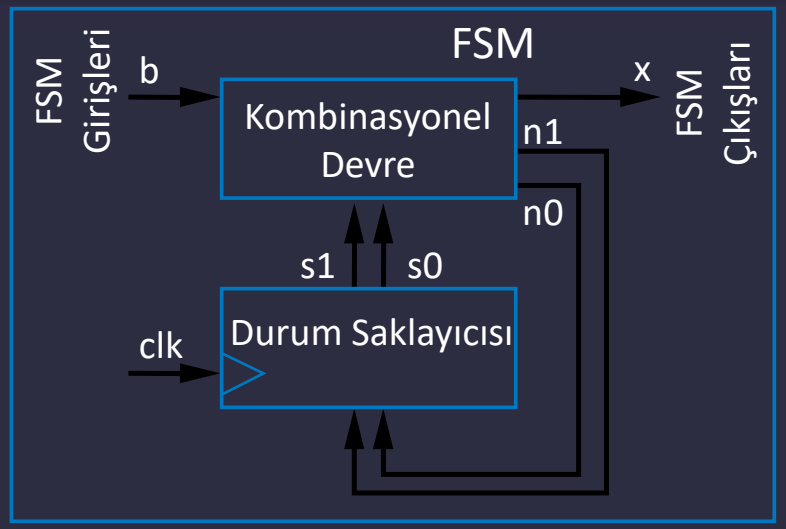




FSM Tasarım Örneği

- Adım 5: Kombinasyonel Devre Gerçeklemesi

	Girişler			Çıkışlar		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	0
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0



x = s1 | s0

n1 = s1's0b' | s1's0b | s1s0'b' | s1s0'b

n1 = s1's0 | s1s0'

n0 = s1's0'b | s1s0'b' | s1s0'b

n0 = s1's0'b | s1s0'

FSM Tasarım Örneği

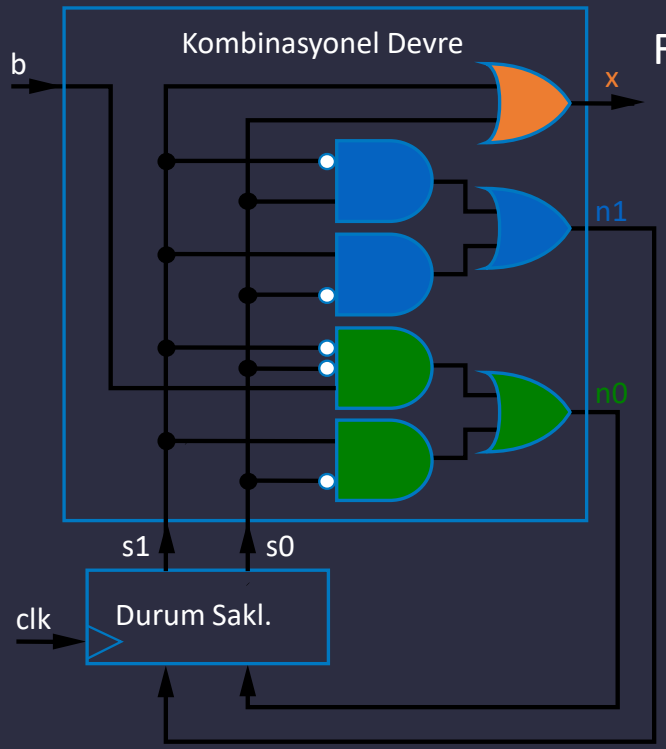
- Adım 5: Kombinasyonel devreyi oluştur

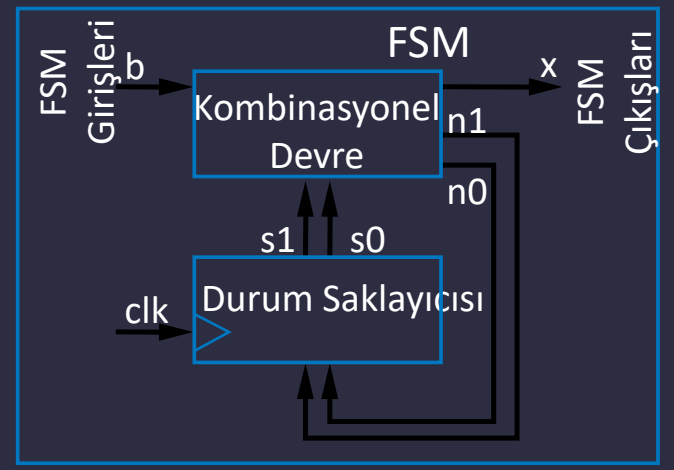
	Girişler			Çıkışlar		
	s1	s0	b	x	n1	n0
Off	0	0	0	0	0	0
	0	0	1	0	0	1
On1	0	1	0	1	1	0
	0	1	1	1	1	0
On2	1	0	0	1	1	1
	1	0	1	1	1	1
On3	1	1	0	1	0	0
	1	1	1	1	0	0

x = s1 | s0

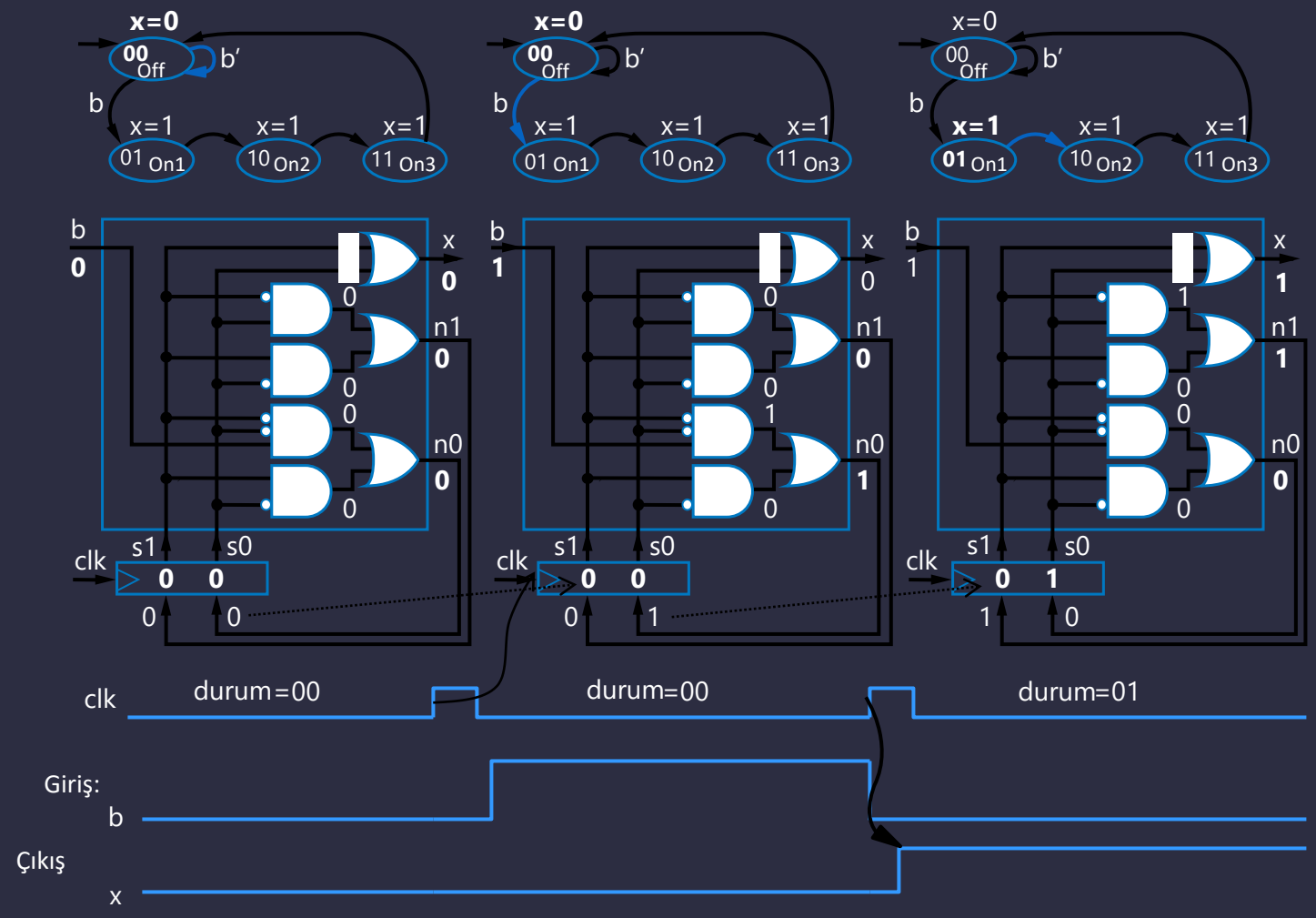
n1 = s1's0 | s1s0'

n0 = s1's0'b | s1s0'

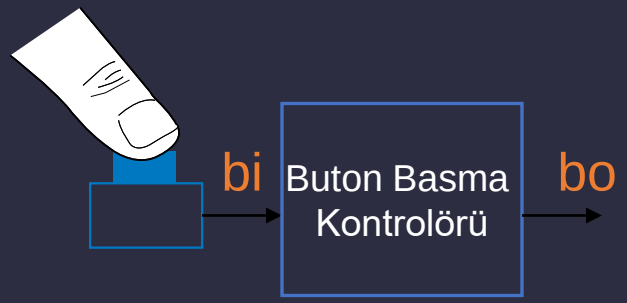


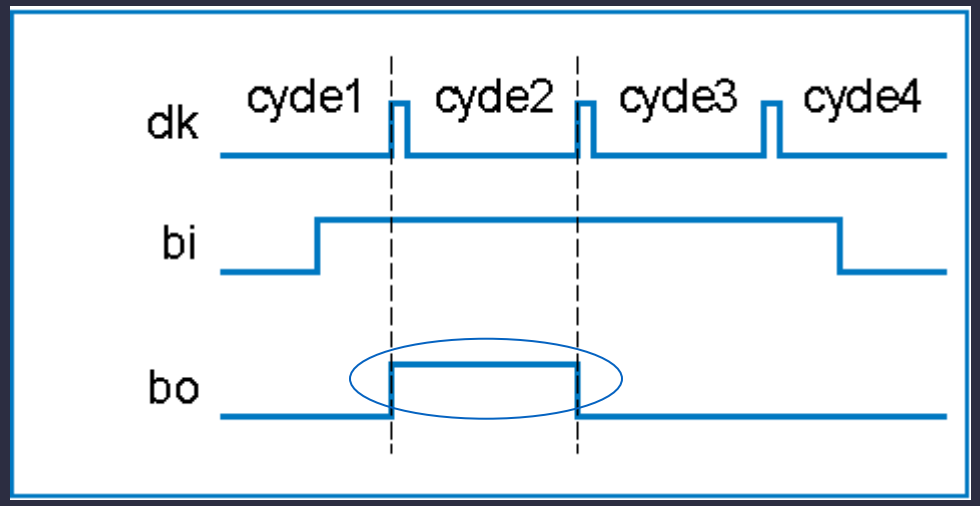


FSM’nin İç Yapısı



Buton Filtre Modülü Örneği

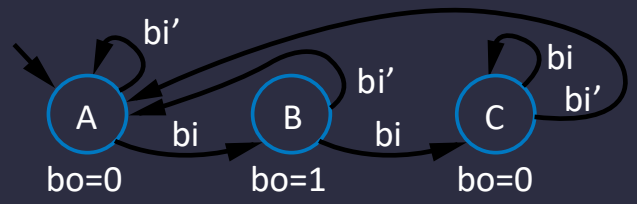




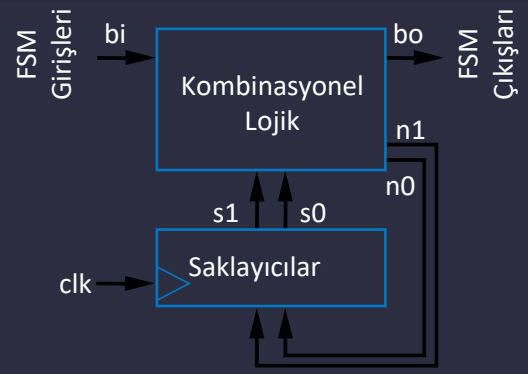
- Butona basıldığında sadece 1 cycle pulse üreten bir tasarım yapılmak isteniyor

Buton Filtre Modülü Örneği

FSM Girişler: bi; FSM çıkışlar: bo

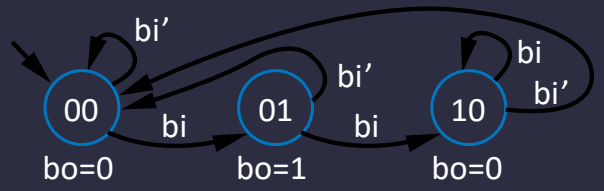


Adım 1: FSM



Adım 2: Mimari oluşturun

FSM giriş: bi; FSM çıkış: bo

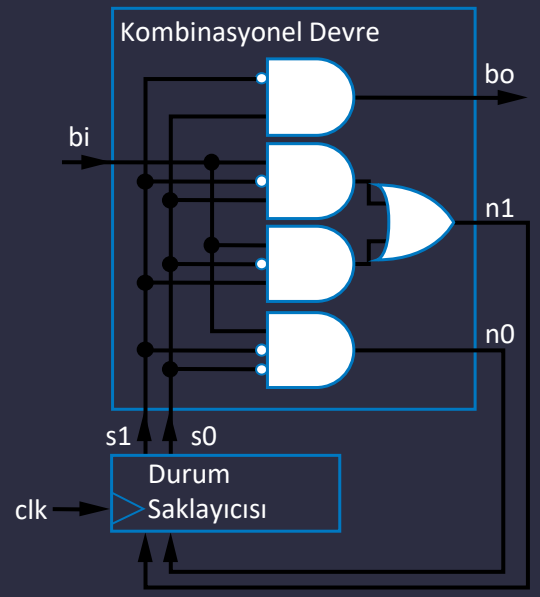


Adım 3: Durum kodlaması

Girişler			Çıkışlar		
s1	s0	bi	n1	n0	bo
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	1	0	0	0

Adım 4: Durum Tablosu

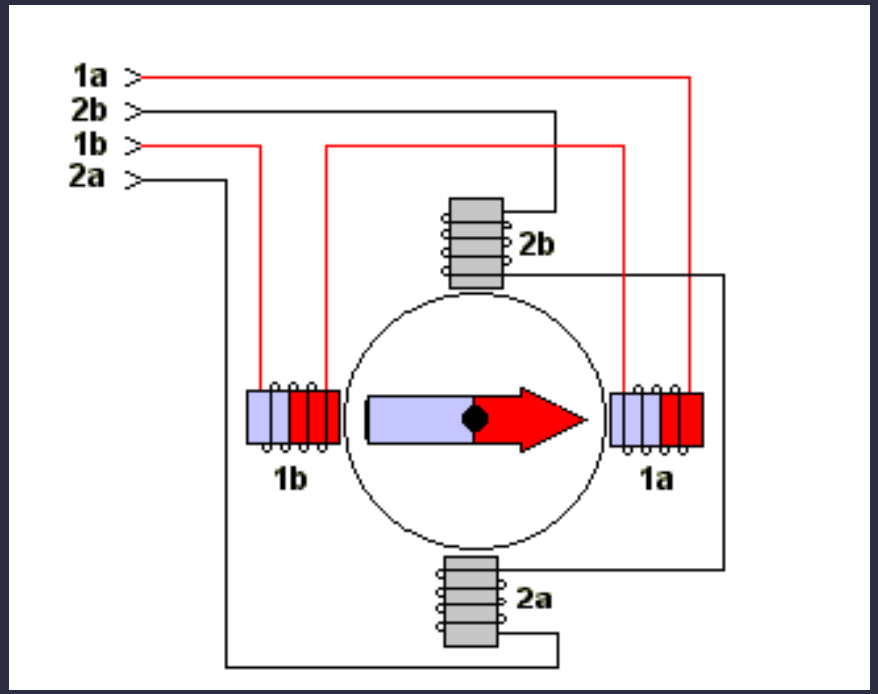
n1 = s1's0bi | s1s0bi
n0 = s1's0'bi
bo = s1's0bi' | s1's0bi = s1s0



Adım 5: Devre Oluşturun

FSM Örneği

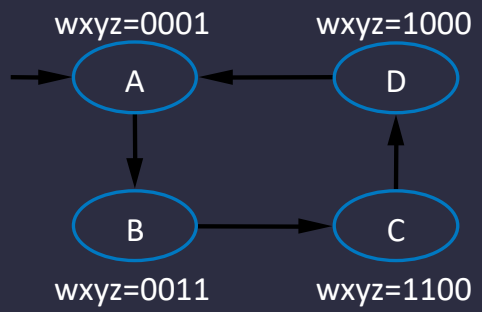
- 0001, 0011, 1100, 1000 ardışık olarak çıktı veren bir FSM tasarlayınız.
  - Örneğin step motor kontrolü için



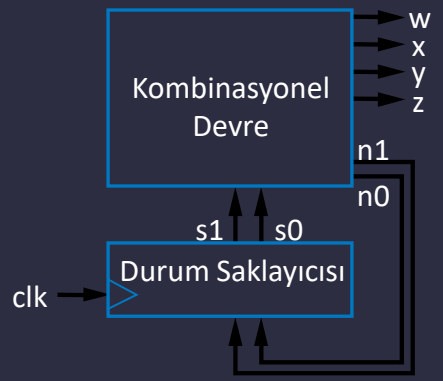
FSM Örneği

- 0001, 0011, 1100, 1000 ardışık olarak çıktı veren bir FSM tasarlayınız.
  - Örneğin step motor kontrolü için

Girişler: yok; Çıkışlar: w,x,y,z

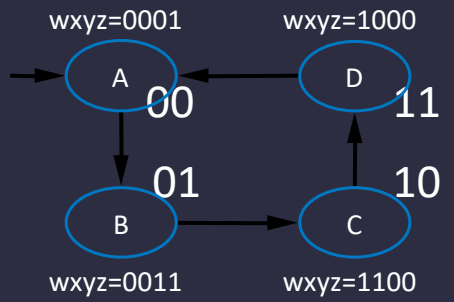


Adım 1: FSM Oluşturma



Adım 2: Mimari Oluşturma

Giriş: yok; çıkış: w,x,y,z

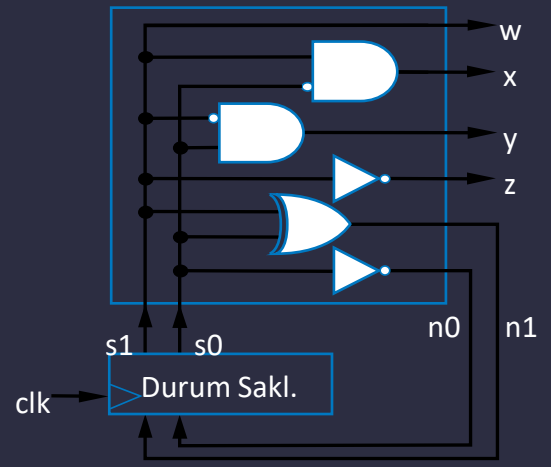


Adım 3: Durum Kodlaması

Girişler		Çıkışlar					
s1	s0	w	x	y	z	n1	n0
A 0	0	0	0	0	1	0	1
B 0	1	0	0	1	1	1	0
C 1	0	1	1	0	0	1	1
D 1	1	1	0	0	0	0	0

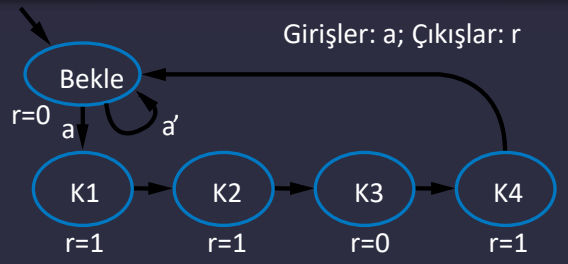
Adım 4: Doğruluk Tablosu Oluşturma

w = s1
x = s1s0'
y = s1's0
z = s1'
n1 = s1 xor s0
n0 = s0'

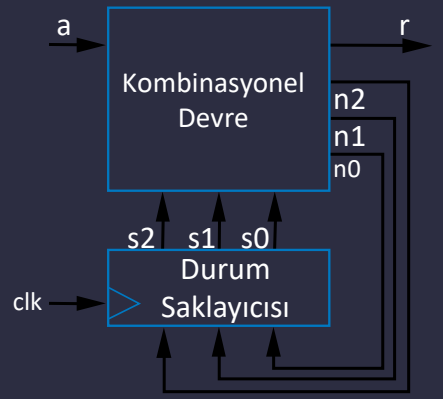


FSM Örneği, Araba Anahtarı

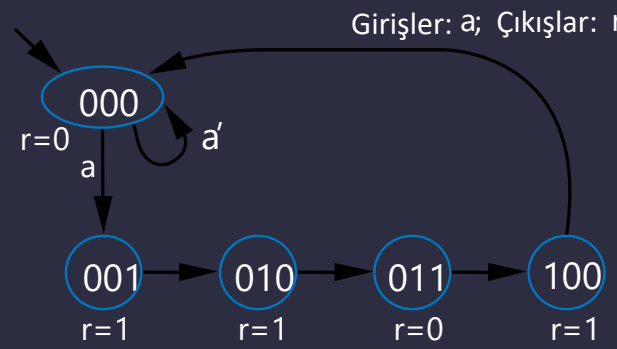
Adım 1



Adım 2



Adım 3

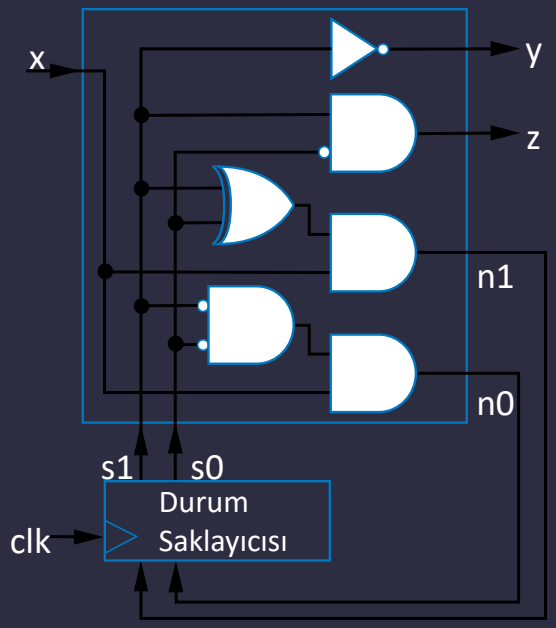


	Girişler				Çıkışlar			
	s2	s1	s0	a	r	n2	n1	n0
Bekle	0	0	0	0	0	0	0	0
	0	0	0	1	0	0	0	1
K1	0	0	1	0	1	0	1	0
	0	0	1	1	1	0	1	0
K2	0	1	0	0	1	0	1	1
	0	1	0	1	1	0	1	1
K3	0	1	1	0	0	1	0	0
	0	1	1	1	0	1	0	0
K4	1	0	0	0	1	0	0	0
	1	0	0	1	1	0	0	0
-	1	0	1	0	0	0	0	0
	1	0	1	1	0	0	0	0
	1	1	0	0	0	0	0	0
	1	1	0	1	0	0	0	0
	1	1	1	0	0	0	0	0
	1	1	1	1	0	0	0	0

Adım 4

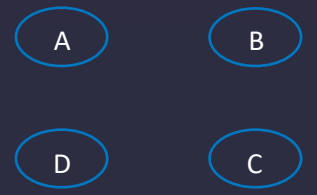
Devreden FSM'e Dönmek

Bu devre ne yapıyor?

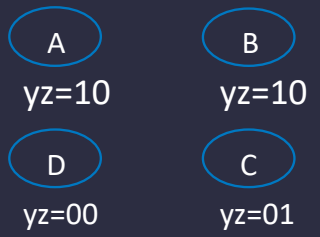


y=s1'
z = s1s0'
n1=(s1 xor s0)x
n0=(s1'*s0')x

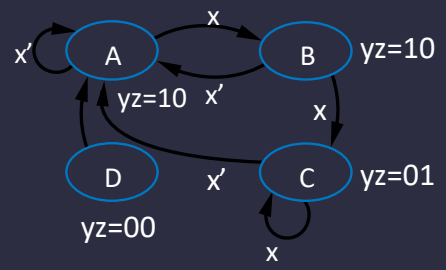
	Girişler			Çıkışlar			
	s1	s0	x	n1	n0	y	z
A	0	0	0	0	0	1	0
	0	0	1	0	1	1	0
B	0	1	0	0	0	1	0
	0	1	1	1	0	1	0
C	1	0	0	0	0	0	1
	1	0	1	1	0	0	1
D	1	1	0	0	0	0	0
	1	1	1	0	0	0	0



durumlar



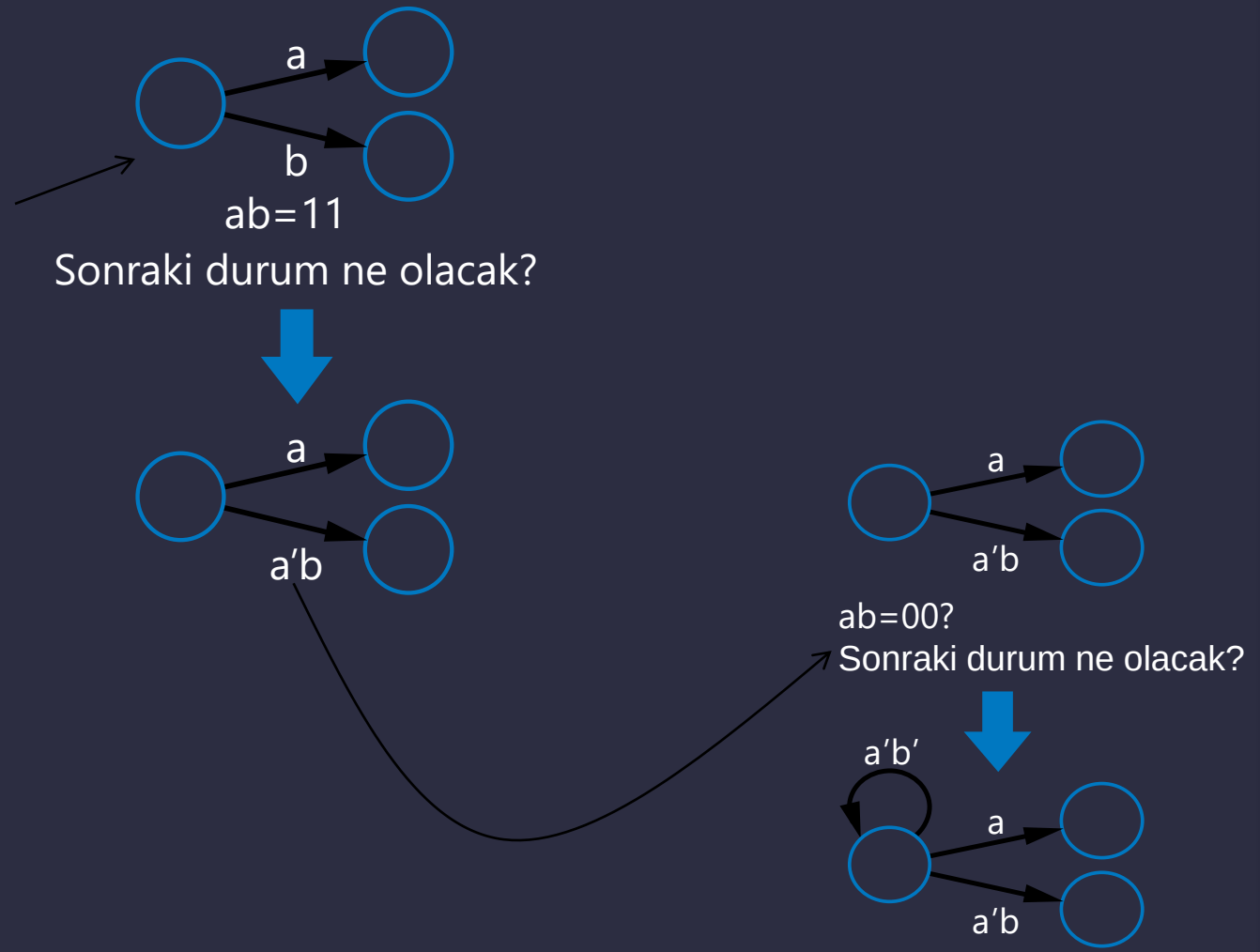
Durumlar ve çıkışlar



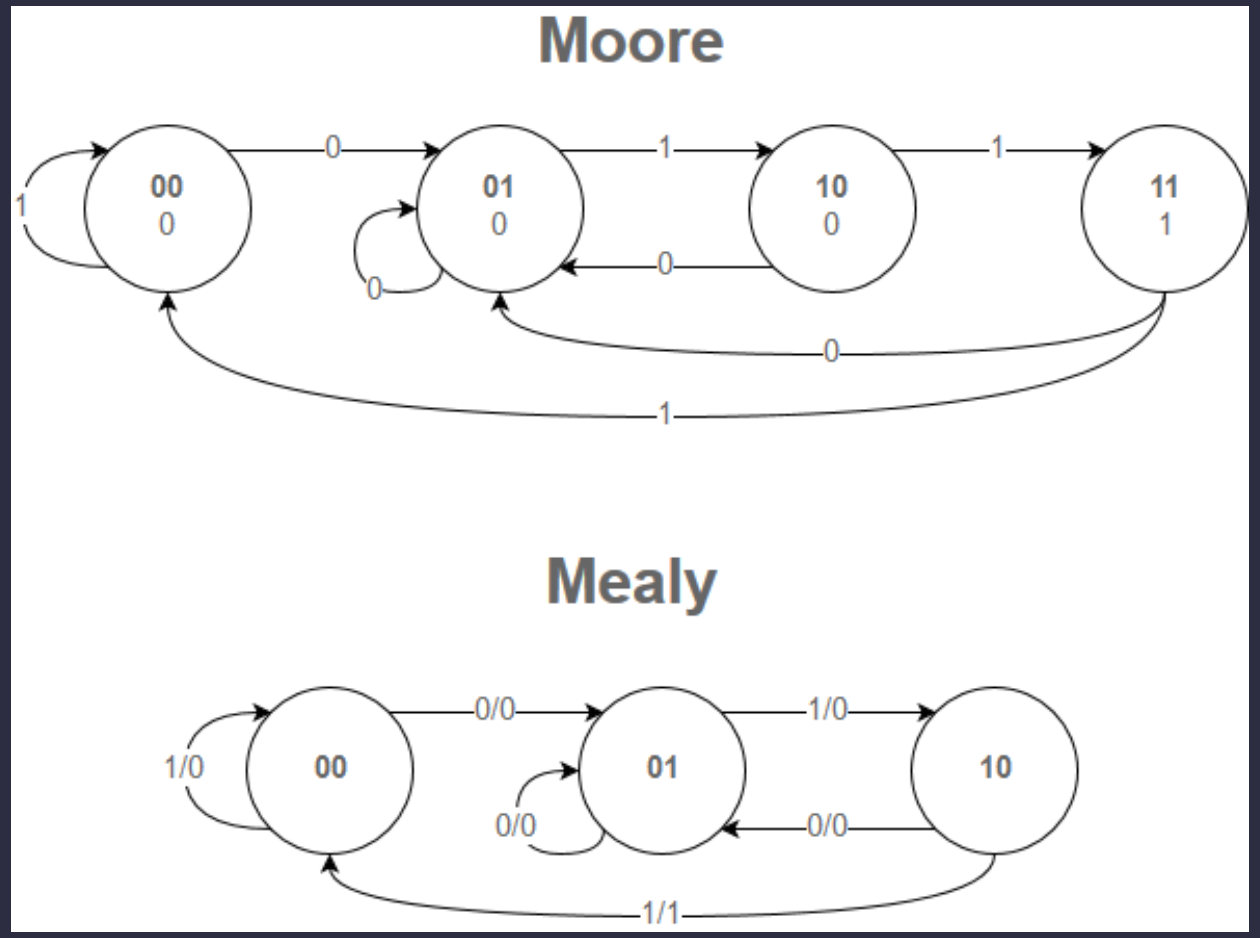
Durumlar, çıkışlar ve geçişler

FSM Tasarım Hataları

- *Durumlar arası geçişlerdeki koşullarda aynı anda tek bir geçiş koşulu doğru olmalıdır*



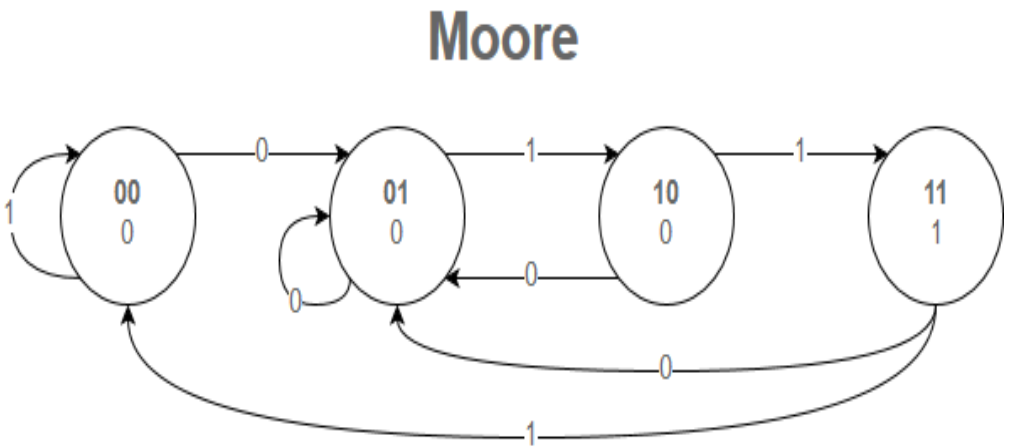
Durum Makinaları



Moore Durum Makinası Örnekleri

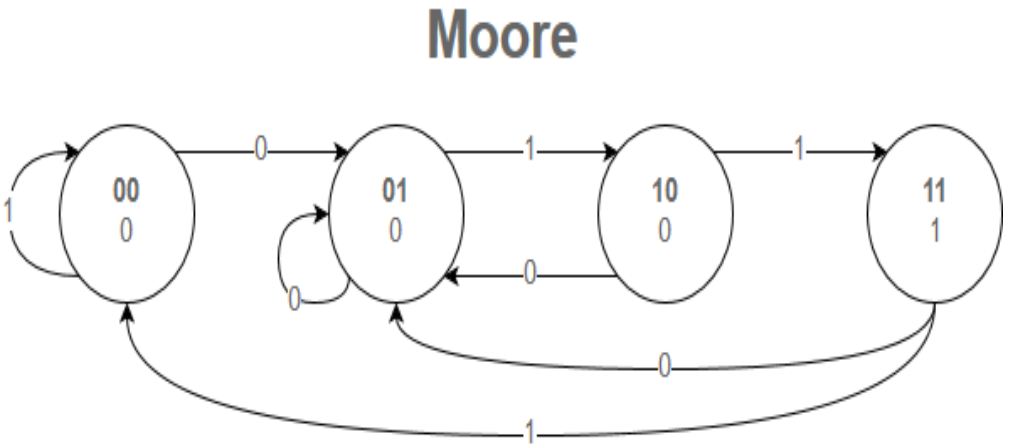
Durum Makinaları

```
module stateMachineTest (input clk, input in, output reg out)

reg [1:0] state, stateNext;
reg outNext;

initial begin
        state = 0;
        stateNext = 0;
        out = 0;
        outNext = 0;
end

always@(posedge clk) begin
        state <= stateNext;
        out <= outNext;
end
```



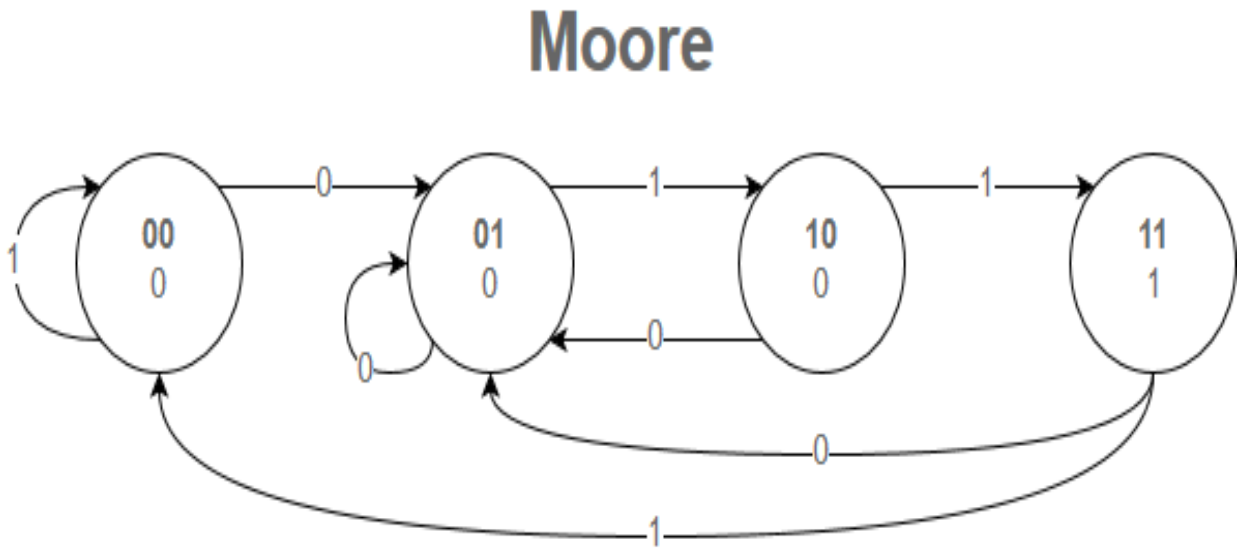
Durum Makinaları

```
always@(*) begin
        stateNext = state;
        outNext = out;
        case(state)
                0: begin
                        if(in == 0)begin
                                stateNext = 1;
                                outNext = 0;
                        end
                end
        endcase
end

endmodule
```



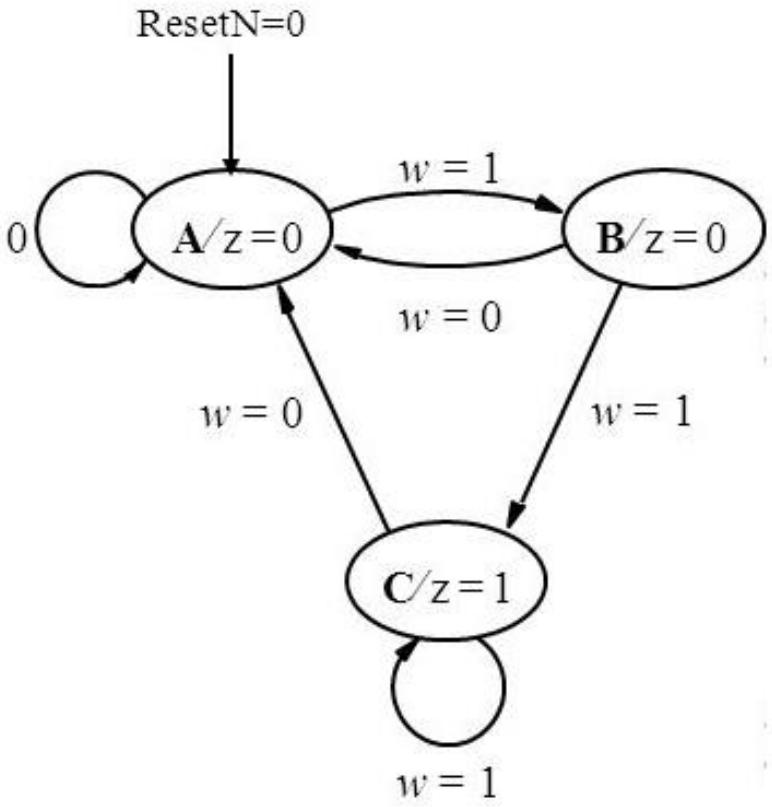
Durum Makinaları

```
1: begin
        if(in == 1)begin
                stateNext = 2;
                outNext = 0;
        end
end
2: begin
        if(in == 1)begin
                stateNext = 3;
                outNext = 1;
        end else begin
                stateNext = 1;
                outNext = 0;
        end
end
3: begin
        if(in == 0)begin
                stateNext = 1;
                outNext = 0;
        end else begin
                stateNext = 0;
                outNext = 0;
        end
end
```



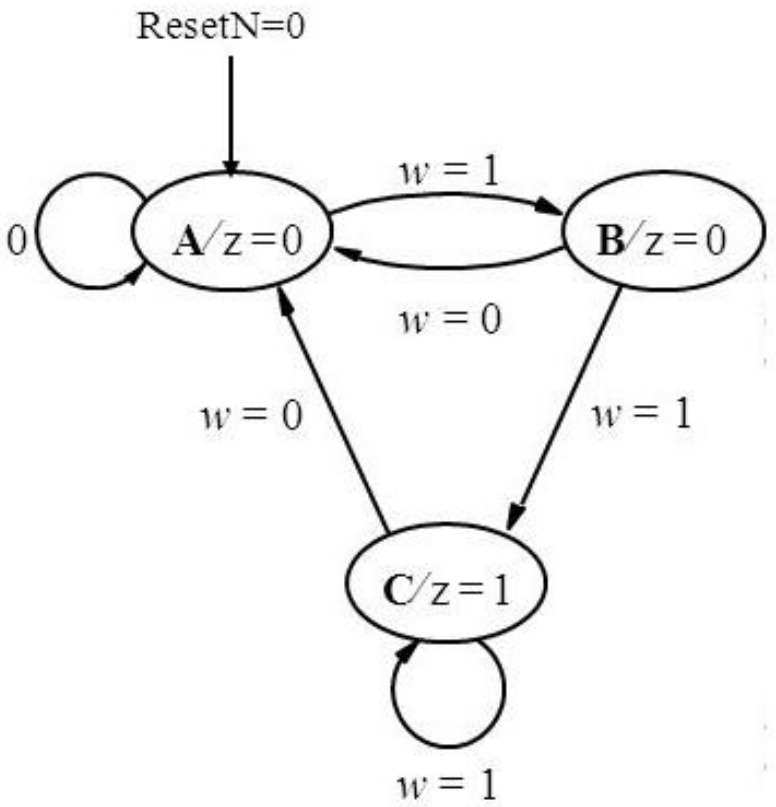
Durum Makinaları

```
module mooreMachine2 (input clk, input ResetN, input w, output reg z)

reg [1:0] state, stateNext;
reg zNext;

initial begin
        state = 0;
        stateNext = 0;
        z = 0;
        zNext = 0;
end

always@(posedge clk) begin
        state <= stateNext;
        z <= zNext;
end
```



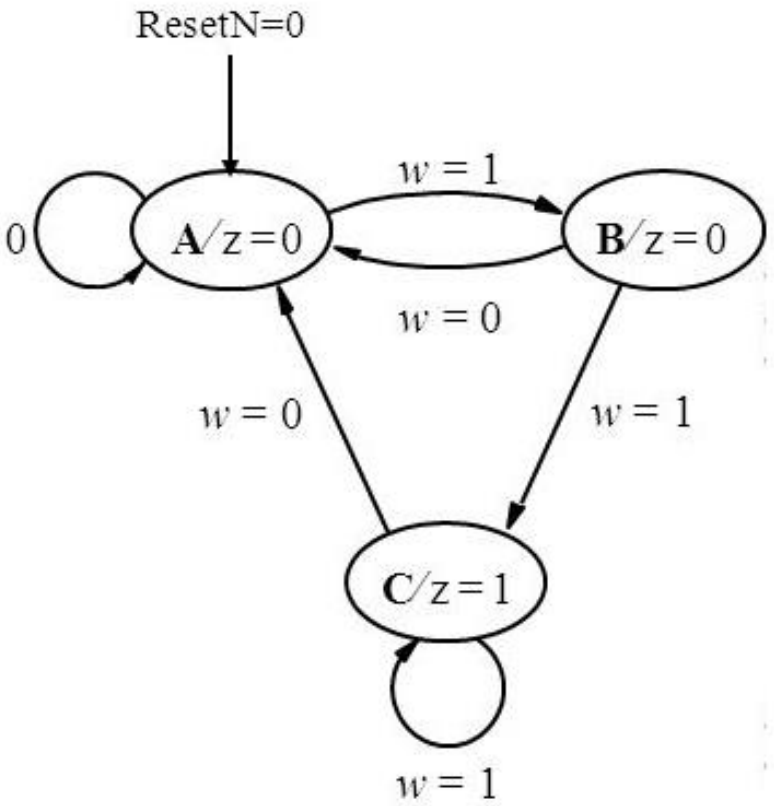
Durum Makinaları

```
always@(*) begin
        stateNext = state;
        zNext = z;
        if(ResetN) begin
                stateNext = 0;
                zNext = 0;
        end else begin
                case(state)
                        0: begin
                                if(w == 1)begin
                                        stateNext = 1;
                                        zNext = 0;
                                end
                        end
                endcase
        end
end
endmodule
```



Durum Makinaları

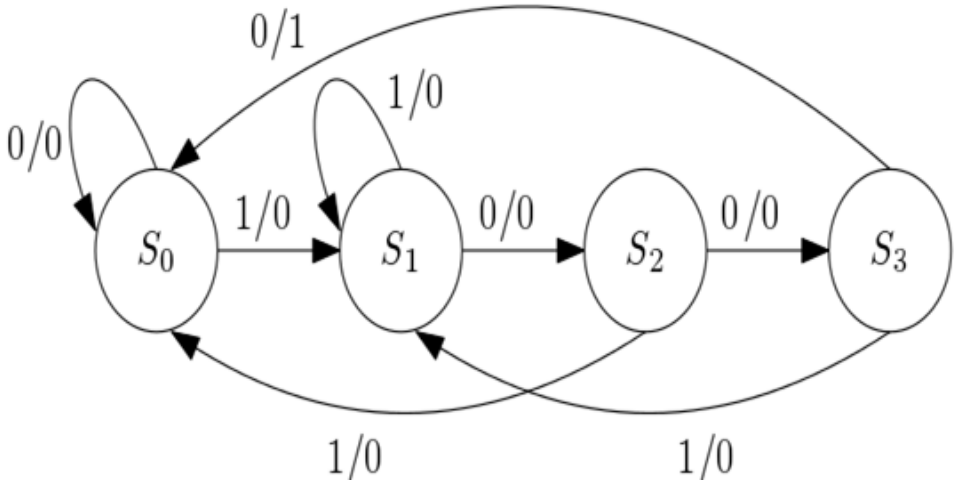
```
1: begin
        if(w == 0)begin
                stateNext = 0;
                zNext = 0;
        end else begin
                stateNext = 2;
                zNext = 1;
        end
end
2: begin
        if(w == 0)begin
                stateNext = 0;
                zNext = 0;
        end
end
```



Mealy Durum Makinası Örnekleri

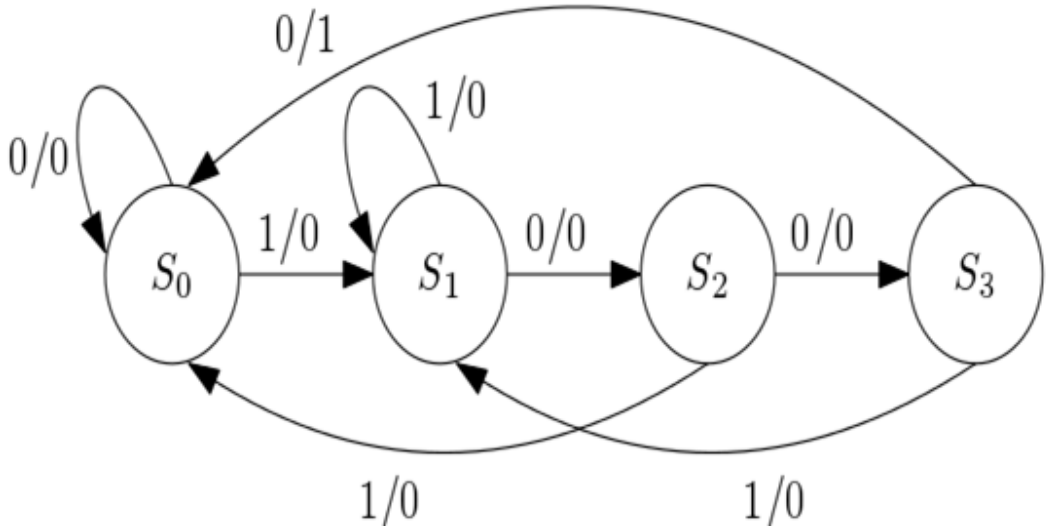
Durum Makinaları

```
module mooreMachine1 (input clk, input in, output reg out)

reg [1:0] state, stateNext;

initial begin
        state = 0;
        stateNext = 0;
        out = 0;
end

always@(posedge clk) begin
        state <= stateNext;
end
```



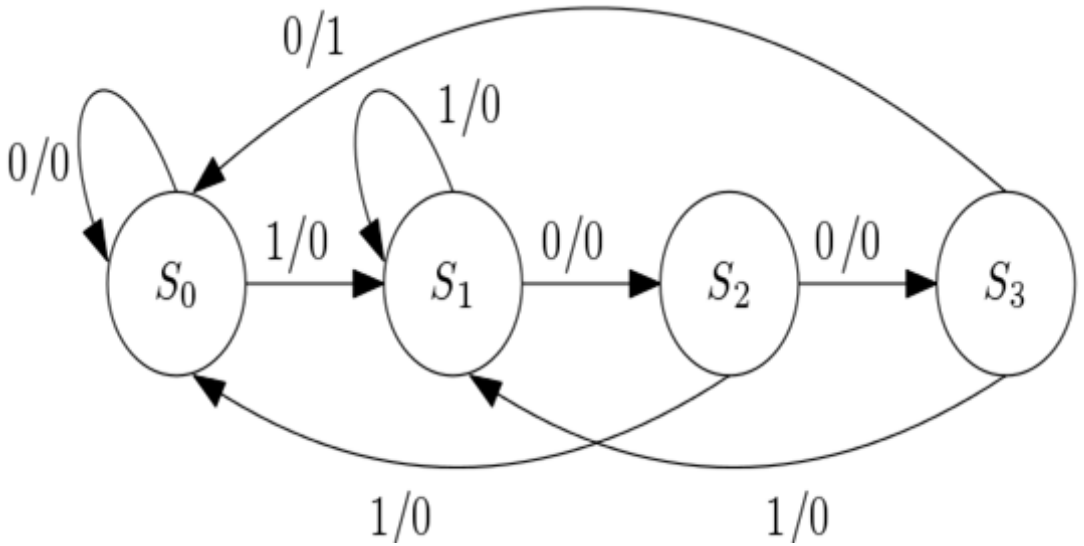
Durum Makinaları

```
always@(*) begin
        stateNext = state;
        out = 0;
        case(state)
                0: begin
                        if(in == 0)begin
                                out = 0;
                        end else begin
                                out = 1;
                                stateNext = 1;
                        end
                end
        endcase
end
```



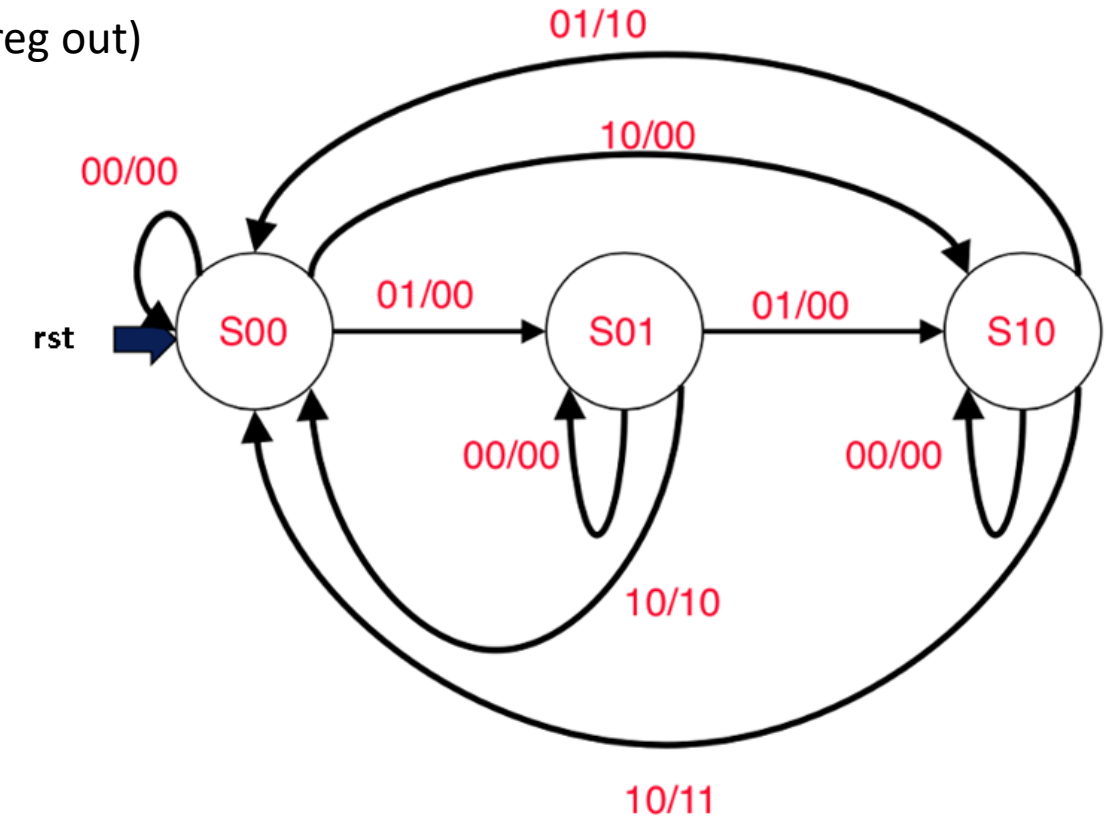
Durum Makinaları

```
1: begin
        if(in == 0)begin
                out = 0;
                stateNext = 2;
        end else begin
                out = 0;
        end
end
2: begin
        if(in == 0)begin
                out = 0;
                stateNext = 3;
        end else begin
                out = 0;
                stateNext = 0;
        end
end
3: begin
        if(in == 0)begin
                out = 1;
                stateNext = 0;
        end else begin
                out = 0;
                stateNext = 1;
        end
end
```



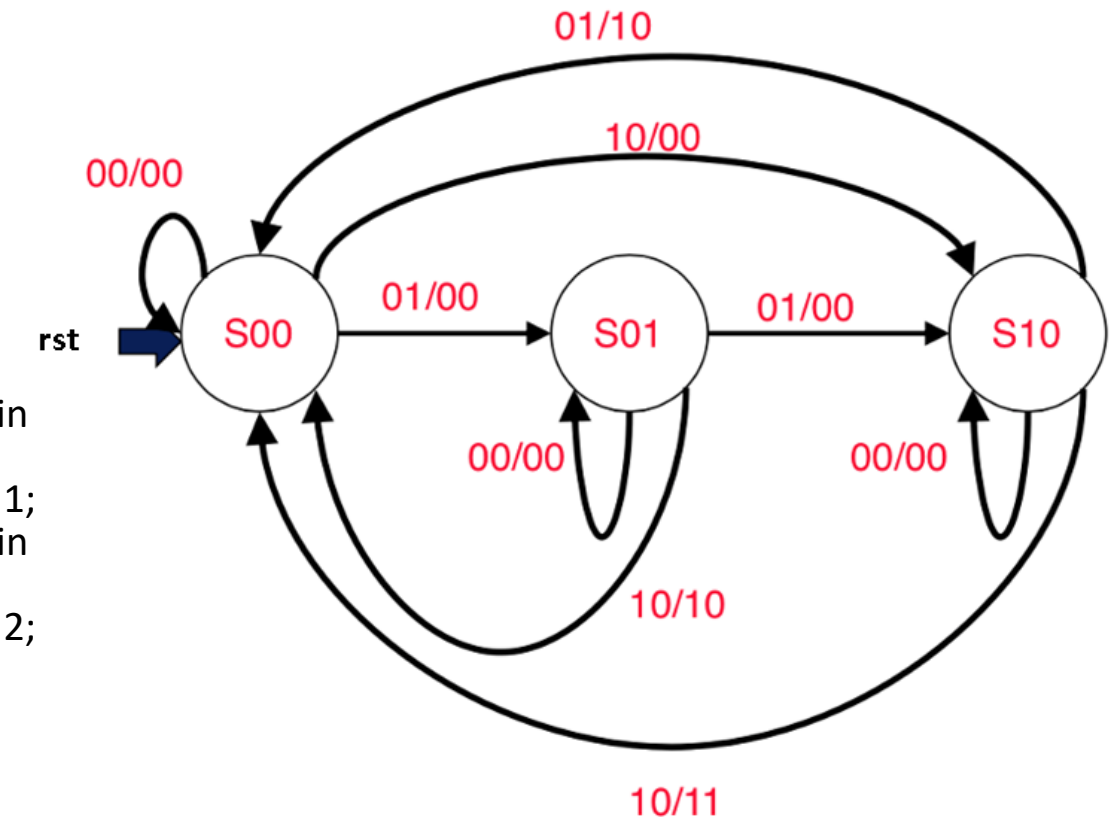
Durum Makinaları

```
module mooreMachine2 (input clk, input in, output reg out)

reg [1:0] state, stateNext;

initial begin
        state = 0;
        stateNext = 0;
        out = 0;
end

always@(posedge clk) begin
        state <= stateNext;
end
```



Durum Makinaları

```
always@(*) begin
        stateNext = state;
        out = 0;
        if(rst) begin
                stateNext = 0;
                out = 0;
        end else begin
                case(state)
                        0: begin
                                if(in == 0)begin
                                        out = 0;
                                end else if(in == 1) begin
                                        out = 0;
                                        stateNext = 1;
                                end else if(in == 2) begin
                                        out = 0;
                                        stateNext = 2;
                                end
                        end             endcase
        end
end

endmodule
```



Durum Makinaları

```
1: begin
            if(in == 0)begin
                        out = 0;
            end else if(in == 1) begin
                        out = 0;
                        stateNext = 2;
            end else if(in == 2) begin
                        out = 2;
                        stateNext = 2;
            end
end
2: begin
            if(in == 0)begin
                        out = 0;
            end else if(in == 1) begin
                        out = 2;
                        stateNext = 0;
            end else if(in == 2) begin
                        out = 3;
                        stateNext = 0;
            end
end
```