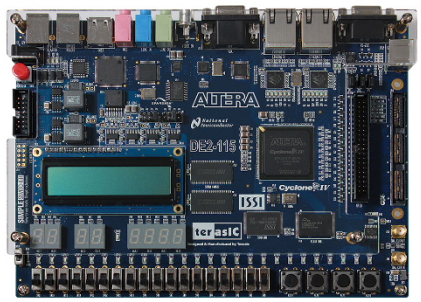
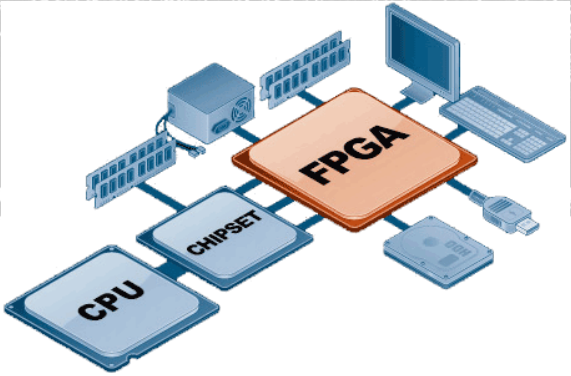
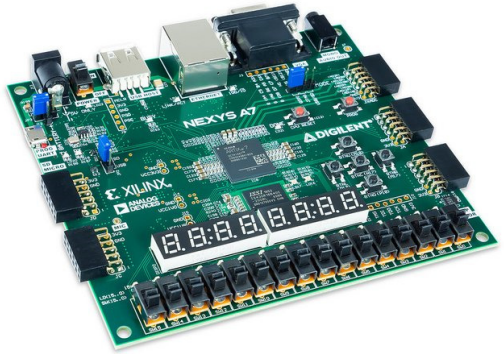
İLERİ SAYISAL SİSTEMLER

1- Sayısal Sistemler ve FPGA







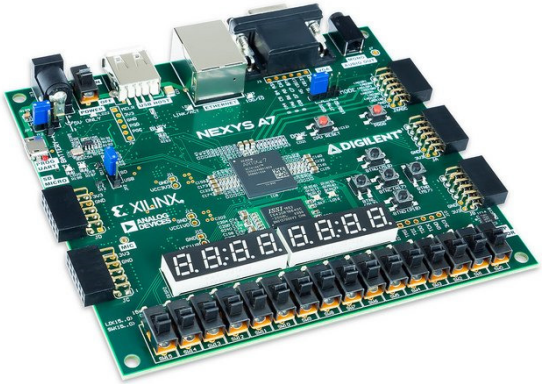
Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

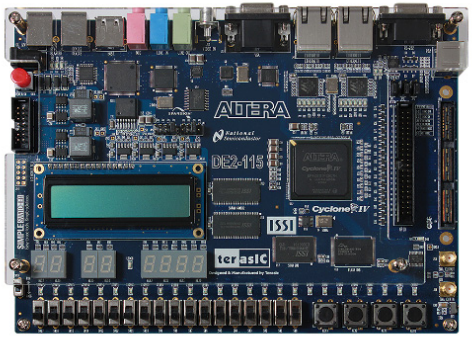
1. SAYISAL SİSTEMLER VE FPGA

- FPGA; "**Sahada Programlanabilir Kapı Dizileri**" anlamına gelen "**Field Programmable Gate Array**" ifadesinin kısaltmasıdır.

- FPGA, üretimden sonra ***istenen fonksiyona göre donanım yapısı kullanıcı tarafından değiştirilebilen entegre (bütünleşmiş) devre*** olarak tanımlanır.



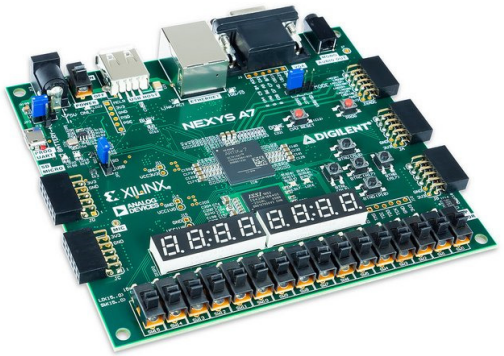
- Herhangi bir üretim alanında kullanılması planlanan bir ürünün, üretimden önceki ***prototipleme*** ve ***test*** işlemleri gibi birçok işlemi ***FPGA***'ler tarafından gerçekleştirilir.



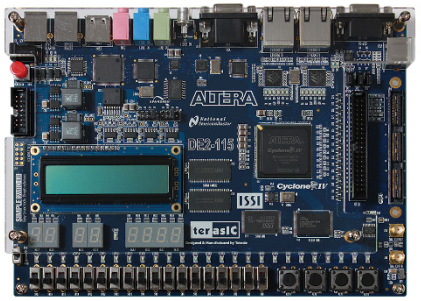
1. SAYISAL SİSTEMLER VE FPGA

- Bu aygıtlar; ***bir elektronik tansiyon aletinin kontrol entegresinden dijital kol saatinin işlemcisine, ses anfisinin sayısal entegresinden cep telefonlarındaki işlemciye kadar*** çok geniş bir alanda kullanılabilir özellikleriyle ön plandadır.

- FPGA'ler; yerine göre bir ***mikroişlemci*** gibi, bir ***şifreleme ünitesi*** gibi veya bir ***grafik kartı*** gibi işlem görebilmektedir.



- Hatta ***bu üç işlemin aynı anda çalışabileceği bir sistemin parçası*** da olabilir.



- Günümüzde ileri teknoloji ürünü olarak akla gelen neler varsa tamamının gelişim süreçleri FPGA'ler ile sağlanmıştır.

1. SAYISAL SİSTEMLER VE FPGA

- Günümüzde önemli gömülü sistem entegreleri olarak görülen ***ARM mikrodenetleyicileri*** FPGA'ler üzerinde geliştirilmiştir.

- Gömülü sistem teknolojileri kapsamına giren ***mikrodenetleyici***, ***mikroişlemci*** ile ***ASIC*** adı verilen özel amaçlar için tasarlanan entegre devrelerin FPGA'den farklılıklarının bilmesi gereklidir.

1.1 FPGA ile ASIC (Uygulamaya Özel Entegre Devre) Karşılaştırması

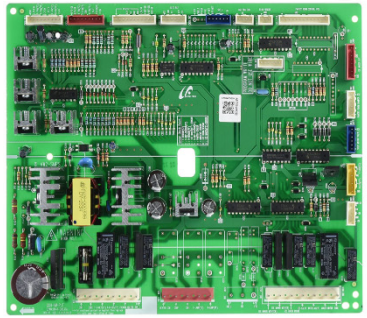
- ASIC, özel uygulamalar için geliştirilen tümleşik bir devredir. ***Sayısal devrelerin*** yanında istenirse ***analog devreler*** ve ***alıcı-verici devreleri*** içebilir.

- ***Tek bir amaç için tasarlanan ASIC'ler ömürleri dolana kadar bu işi yaparlar.***

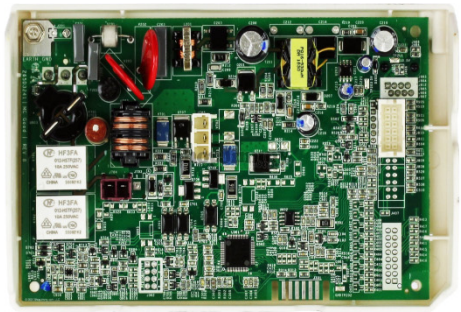
- Örneğin; ***telefonun içindeki işlemci*** veya ***çamaşır makinesindeki kontrol devresi*** bir ASIC olarak işlev görür.

1.1 FPGA ile ASIC (Uygulamaya Özel Entegre Devre) Karşılaştırması

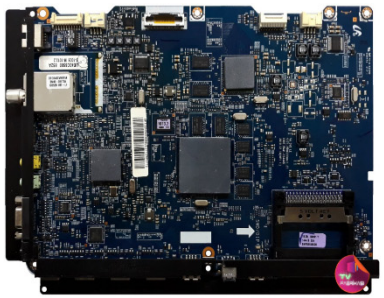
- Yapıları itibari ile silikon seviyede olduklarından dolayı ***işlevleri kesinlikle değiştirilemez.***



Buzdolabı Kontrol Kartı

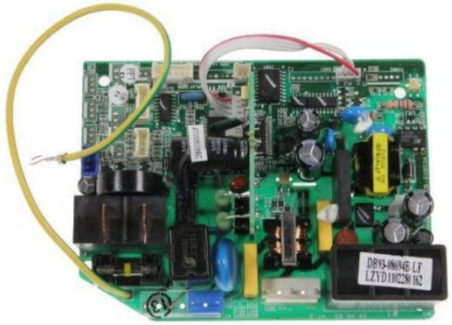


Bulaşık Makinesi Ana Kartı



TV Ana Kartı

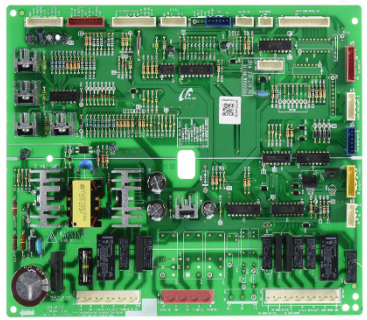
Oksijen Ölçüm Aleti



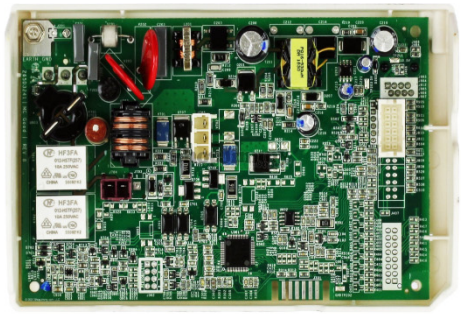
Klima Ana Kartı

1.1 FPGA ile ASIC (Uygulamaya Özel Entegre Devre) Karşılaştırması

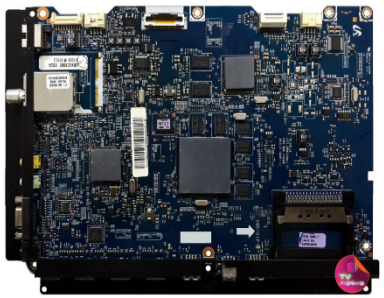
- ASIC'lerin FPGA'ler ile benzer yönü ise ***mantıksal devre tasarımlarının VHDL veya Verilog tarzı dillerle*** geliştirilmesidir.



Buzdolabı Kontrol Kartı

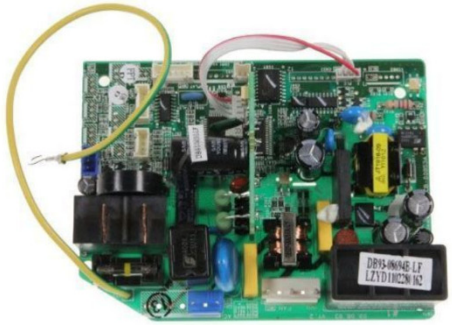


Bulaşık Makinesi Ana Kartı



TV Ana Kartı

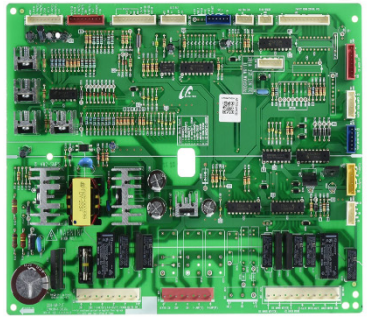
Oksijen Ölçüm Aleti



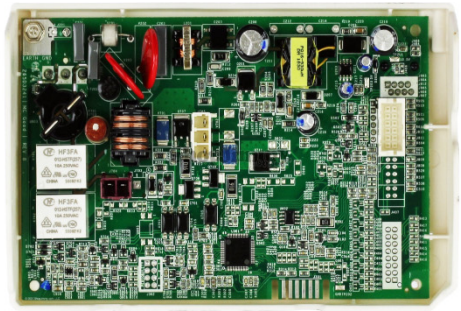
Klima Ana Kartı

1.1 FPGA ile ASIC (Uygulamaya Özel Entegre Devre) Karşılaştırması

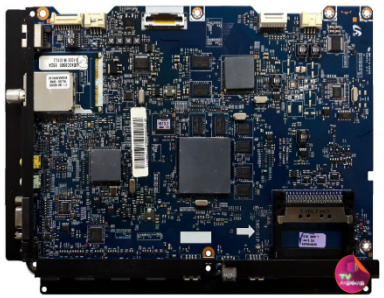
- Ancak ASIC yapılarının üretimi gerçekleştirildikten sonra ***tasarımlarında bir hata tespit edilirse, bütün üretimlerin yeniden toplanması ve yeni üretimlerin piyasaya sürülmesi*** gerekir.



Buzdolabı Kontrol Kartı

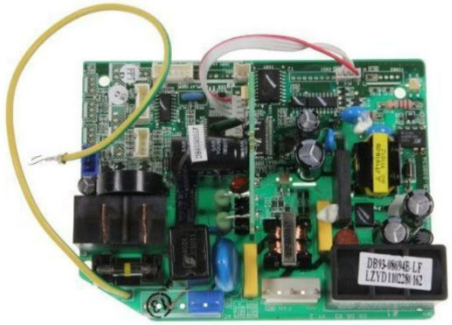


Bulaşık Makinesi Ana Kartı



TV Ana Kartı

Oksijen Ölçüm Aleti



Klima Ana Kartı

1.1 FPGA ile ASIC (Uygulamaya Özel Entegre Devre) Karşılaştırması

- ***Güç tüketimi açısından*** FPGA'lere göre daha az enerji harcayıp daha az yer kaplamakta ve daha hızlı işlem yapabilmektedir.

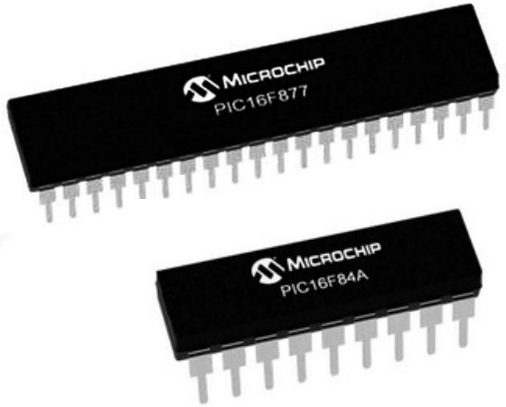
- ***Tasarım maliyetleri açısından*** daha avantajlı hale gelebilmeleri için aynı ASIC devreden milyonlarca basılmalıdır.

- ***ASIC devrelerinin FPGA'lerden en önemli farkı*;** işlevlerinin üretim sonrası değiştirilememesi, yani ***yeniden programlama opsiyonlarının bulunmamasıdır.***

- **ASIC tasarımları genelde FPGA üzerinde gerçekleştirilmekte, fonksiyonellik ve performans test edilmekte, sonrasında tasarlanan sayısal devre ASIC olarak gerçeklenmektedir.**

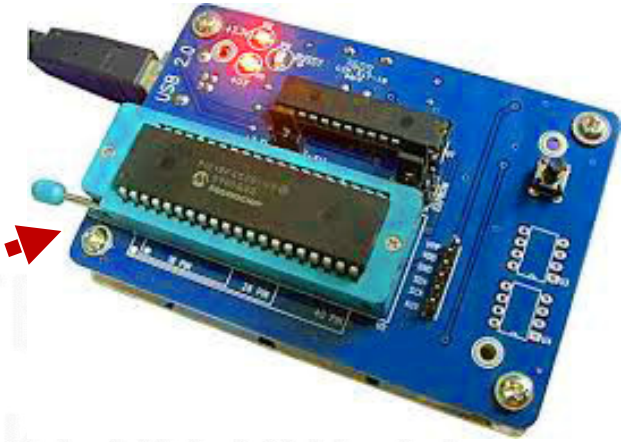
1.2 FPGA ile Mikrodenetleyici Karşılaştırması

- Mikrodenetleyiciler, belirli işlevler için entegre devrelerle güçlendirilmiş mini bir bilgisayar gibidir.



- FPGA'lere göre ***daha az elektrik tüketmekte ve daha ucuza üretilmektedir.***

- Hem mikrodenetleyicilerde hem de FPGA'lerde ***belli amaçlar için kod yazılarak bir çipe aktarılır***.



- Mikrodenetleyiciler hazır devreler üzerinden bu işlemi gerçekleştirirken,

FPGA'lerde tüm işlemlerin ***ayrı ayrı kodlanması*** gerekir.

1.2 FPGA ile Mikrodenetleyici Karşılaştırması

- Bir diğer fark ise komutların işlenme biçiminde saklıdır.

- ***Mikrodenetleyiciler komutları satır satır işlerken, FPGA'ler aynı anda birden fazla satır komutu işleyebilmektedir.*** Bu durum FPGA'ler ve mikrodenetleyiciler arasındaki en önemli ayrımdır.

1.3 FPGA ile Mikroişlemci Karşılaştırması

- FPGA'ler gerekli talimatları, tasarımlarının doğası gereği ***paralel olarak gerçekleştirme imkânına*** sahiptir. ***Belirli program parçaları ile sırayla (ardışıl, sequential) kod işleme yetisi de mevcuttur.***

- Mikroişlemcilerde ise ***komutların sırayla işletilmesi*** gereklidir.

1.3 FPGA ile Mikroişlemci Karşılaştırması

- FPGA temelli bir tasarımın geliştirme süreci, mikrodenetleyici veya mikroişlemci içeren bir sistem tasarımından uzun zaman almaktadır.

- Ancak bu durum, FPGA'lerin ham hali ile tasarımcıya sunulmasından kaynaklıdır.

- Bir FPGA üzerinden bir mikrodenetleyici veya bir mikroişlemci tasarlanabilmesine rağmen, ***tersi durum söz konusu değildir***.

- Tasarım veya programlama açısından inceleme yapılırsa, mikroişlemcilere dair ***komut kümeleri mevcuttur ve bu bilgiler işlemcinin belleğinde saklıdır***.

- ***Komutlar işletilirken bu bellek devreye alınır.***

1.3 FPGA ile Mikroişlemci Karşılaştırması

- FPGA'lerde ise ***elektriksel ilişkisi bulunan mantık hücrelerinin birbirlerine farklı şekillerde bağlanması ile*** tasarım sağlanır.

- Güç tüketimleri temel alınırsa mikroişlemcilerin FPGA'lere göre ***daha tasarruflu*** olduğu görülür.

- Mikroişlemciler ***GHz mertebesinde işlem*** yapmasına karşın, FPGA'ler genellikle ***MHz düzeyinde frekans hızına*** sahiptir.

- Ancak veri işleme noktasında FPGA'lerin mikroişlemcilerden ***çok daha fazla işlem hacmine sahip oldukları*** açıktır.

1.4 Programlanabilir Mantık Cihazları (PLD)

- "Programmable Logic Device" (PLD) olarak bilinen ekipmanlar, yeniden yapılandırılabilir devreler oluşturabilmek için kullanılan elektronik bileşenlerdir.

- PLD'ler sabit bir işleve sahip mantık kapılarının aksine ***üretim esnasında tanımsız işleve*** sahiptir. Bu yüzden ***elektronik devrelerde kullanılmadan önce yapılandırılmaları yani programlanmaları*** gerekir.

- PLD'ler şu anki durumuna, Programlanabilir Salt Okunur Bellek (Programmable Read Only Memory → PROM) birimlerinden başlayan uzun bir gelişim süreci sonrası ulaşmışlardır.

- PROM, PLA, PAL, GAL, SPLD, CPLD ve FPGA yapıları PLD kapsamındadır.

1.4 Programlanabilir Mantık Cihazları (PLD)

PROM (Programmable Read Only Memory):

- Kullanıcı tarafından programlanabilen basit belleklerdir. Bir PROM içerisine; ***bir mikroişlemci programı***, ***basit bir algoritma*** veya ***durum makinesi kodu*** yüklenebilir.

- PROM elemanı oldukça yavaş çalıştığından, ***hız gerektiren tasarımlarda kullanışlı değildir.***

- PROM'lar sadece bir defa programlanabilirken, ***EPROM ve EEPROM gibi türleri silinip tekrar programlanabilmektedir.***

- PROM'lar ***sınırlı sayıda giriş / çıkışı bulunan herhangi bir kombinasyonel devrenin gerçekleştirilmesi için uygun cihazlardır.***

- Ardışıl devreleri PROM türleri ile tasarlamak için, flip-flop veya mikroişlemciler harici olarak tümleşik yapıya eklenmelidir.

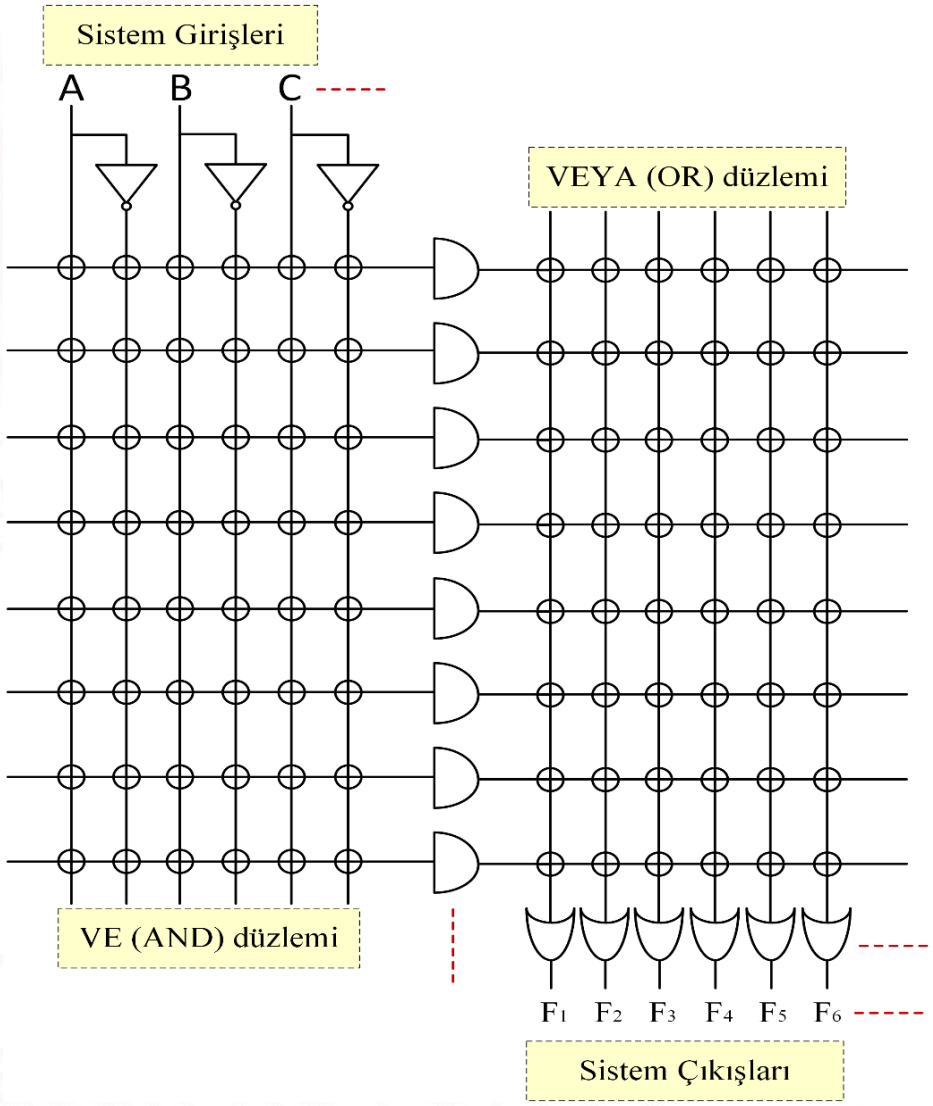
1.4 Programlanabilir Mantık Cihazları (PLD)

PLA (Programmable Logic Array):

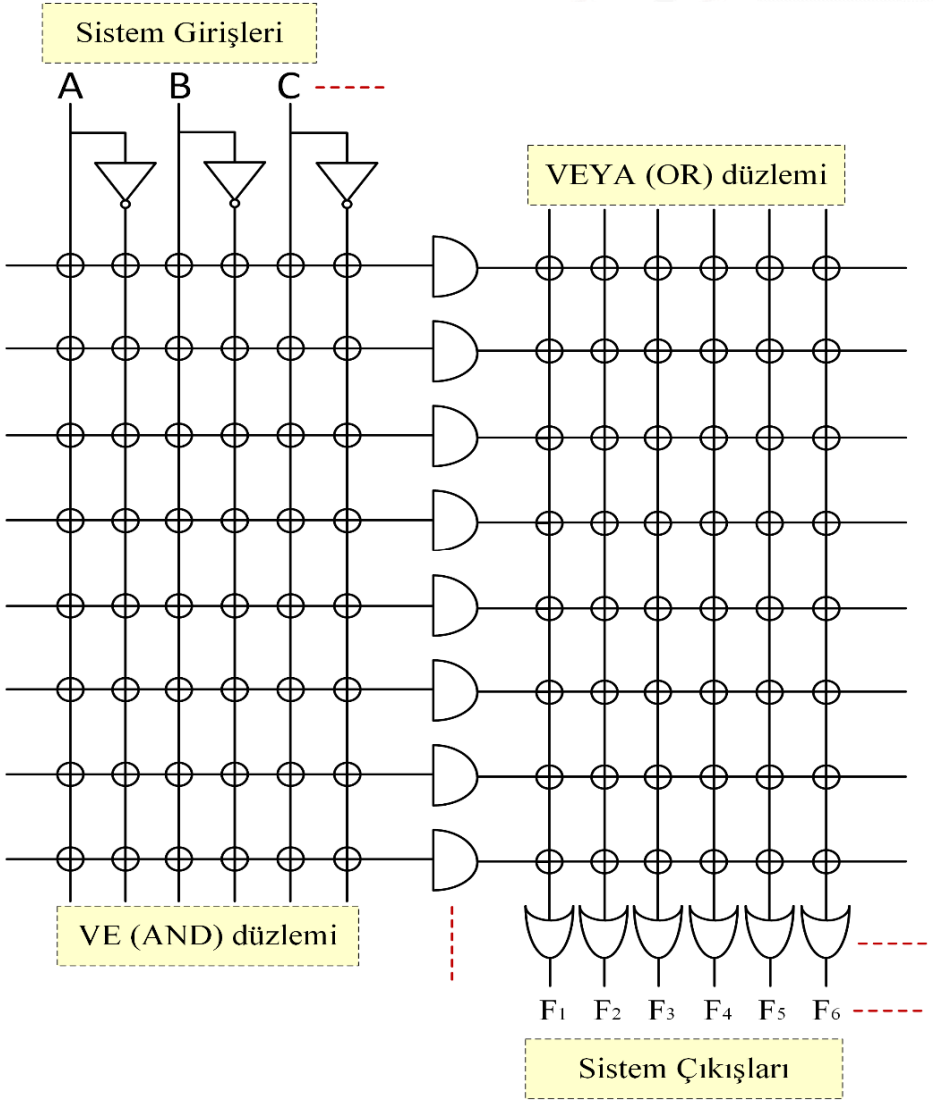
- Programlanabilir mantık dizileri (PLA), ***PROM'lardaki hız ve sınırlı sayıda giriş sorunlarına çözüm olarak geliştirilmişlerdir.***

- PLA yapıları ***çok sayıda girişi desteklemekte ve PROM türlerine göre daha hızlı çalışmaktadırlar.***

- PLA'da girişler, VE (AND) kapılarından oluşan programlanabilir yapıya bağlıdır.



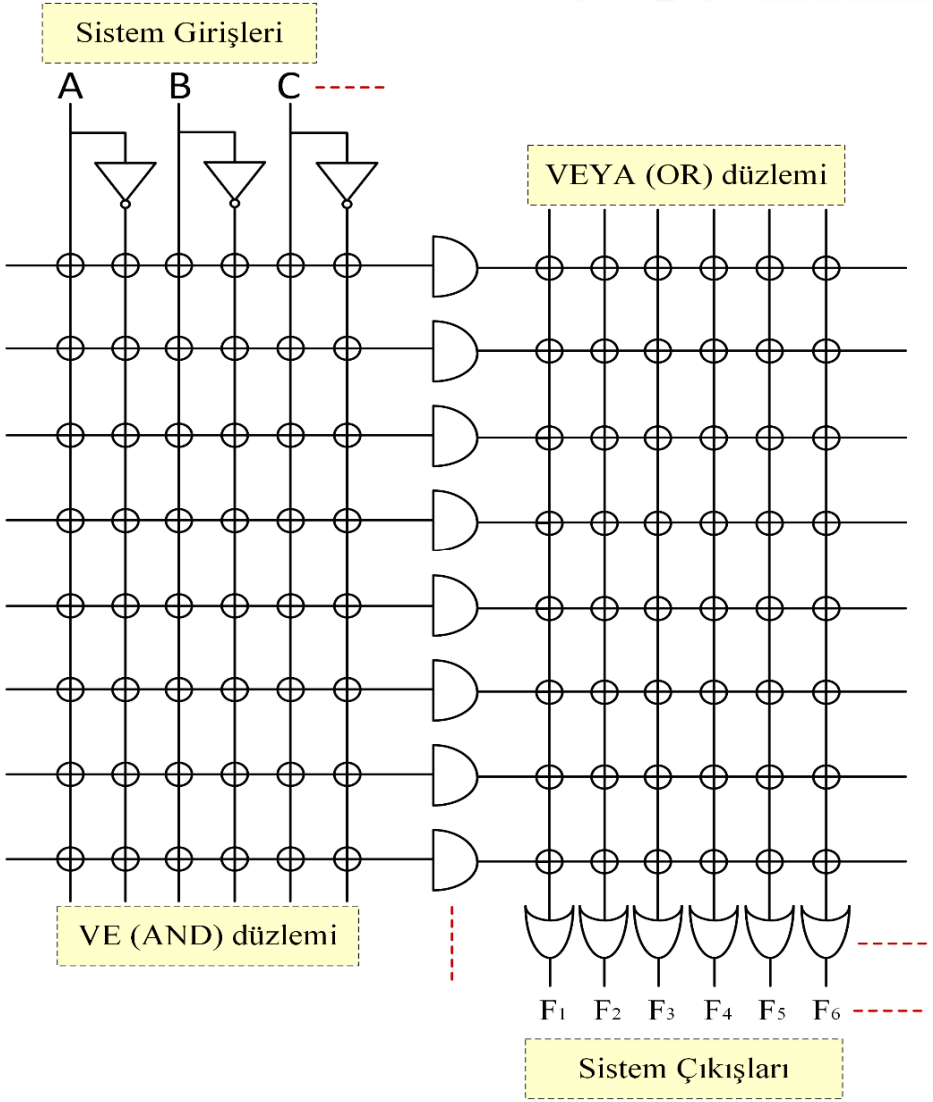
1.4 Programlanabilir Mantık Cihazları (PLD)



PLA (Programmable Logic Array):

- Bu düzlemde tasarıma dair AND işlemleri gerçekleştirilir.
- AND düzleminin çıkışı VEYA (OR) düzlemine bağlanmaktadır.
- OR düzleminde tasarıma ait gereken VEYA işlemleri yapılır ve istenen sistem çıkışları elde edilir.

1.4 Programlanabilir Mantık Cihazları (PLD)

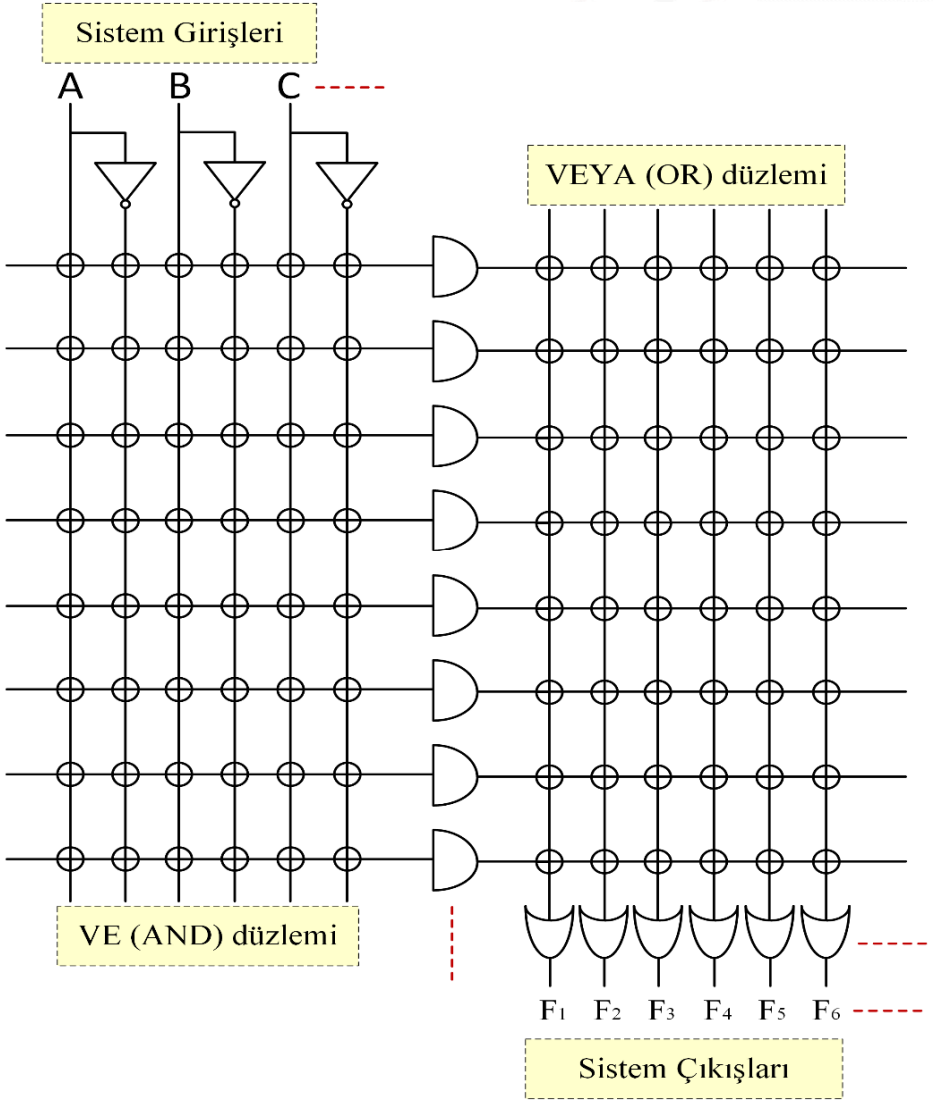


PLA (Programmable Logic Array):

- PROM'daki gibi ***bütün kombinasyonlar gerçekleştirilmez ve sadece gerekli işlemler yapılır.***

- PROM'daki gibi minimum terimler kanonik açılımı işletilerek, ***çarpımların toplamı üzerinden sistem çıkışları üretilir.***

1.4 Programlanabilir Mantık Cihazları (PLD)

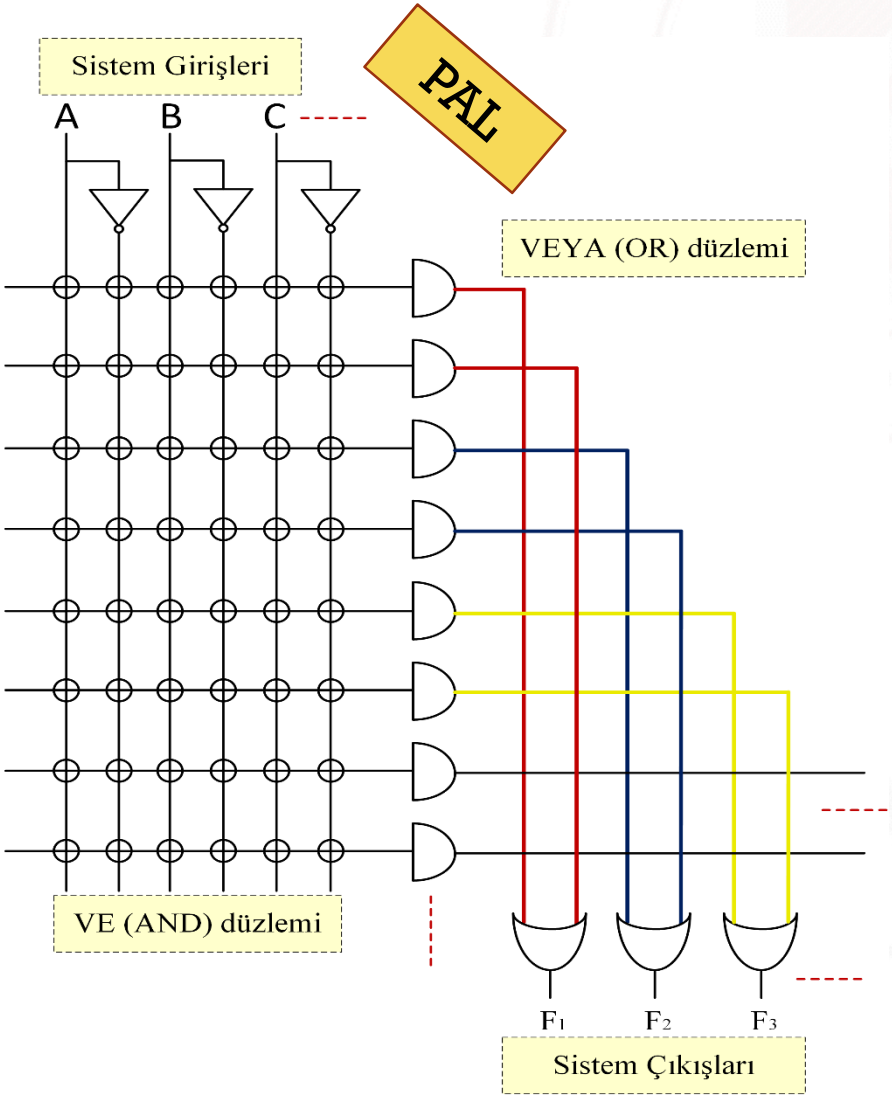


PLA (Programmable Logic Array):

- PLA'ların iki adet programlanabilir düzleme (AND ve OR) sahip olması, ***devre karmaşıklığını artırdığı gibi fazladan kullanılan herbir sigorta bağlantısı daha fazla gecikmeye*** sebebiyet verir.

1.4 Programlanabilir Mantık Cihazları (PLD)

PAL (Programmable Array Logic) ve GAL (Generic Array Logic):

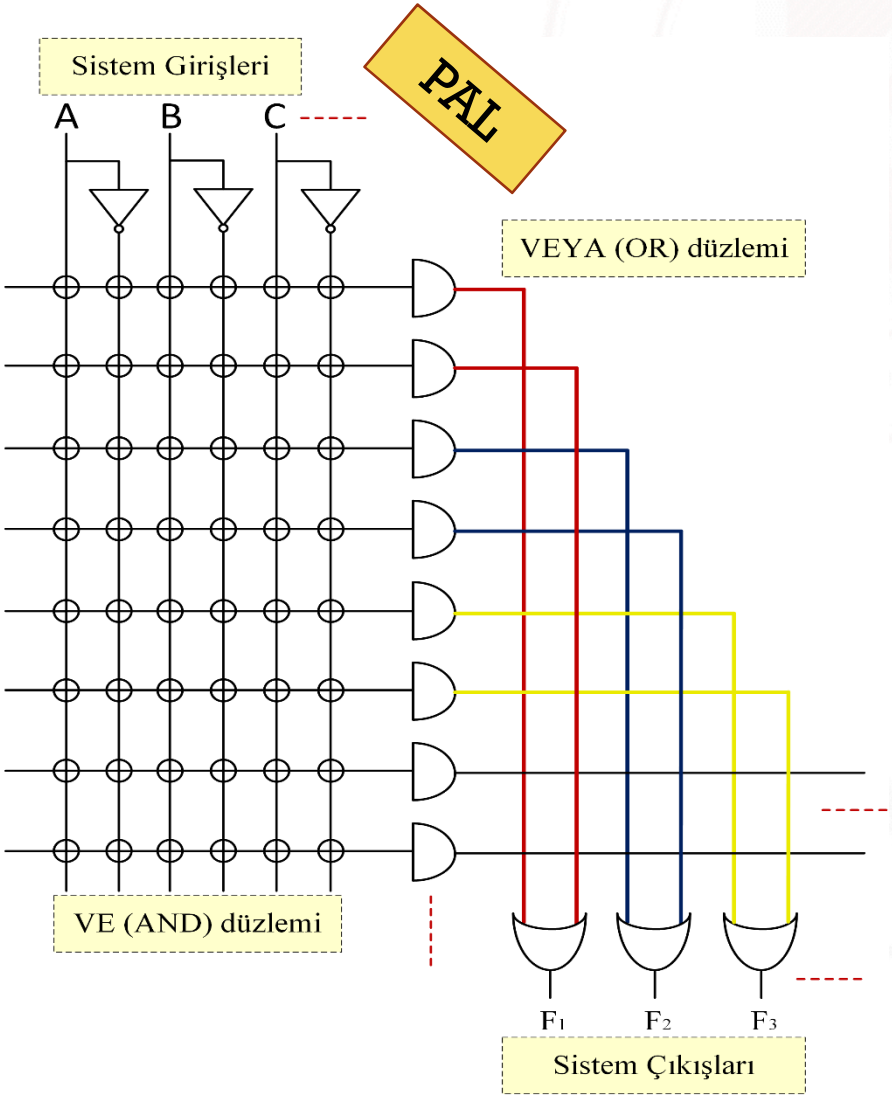


- Programlanabilir dizi mantığı (PAL) ve Genel dizi mantığı (GAL), ***minimum terimler kanonik açılımı ile tasarıma*** dayanır.

- Ancak, ***PAL içindeki AND düzlemi programlanabilmesine rağmen, OR düzlemi sabit tutulmuştur ve programlanamaz.***

1.4 Programlanabilir Mantık Cihazları (PLD)

PAL (Programmable Array Logic) ve GAL (Generic Array Logic):



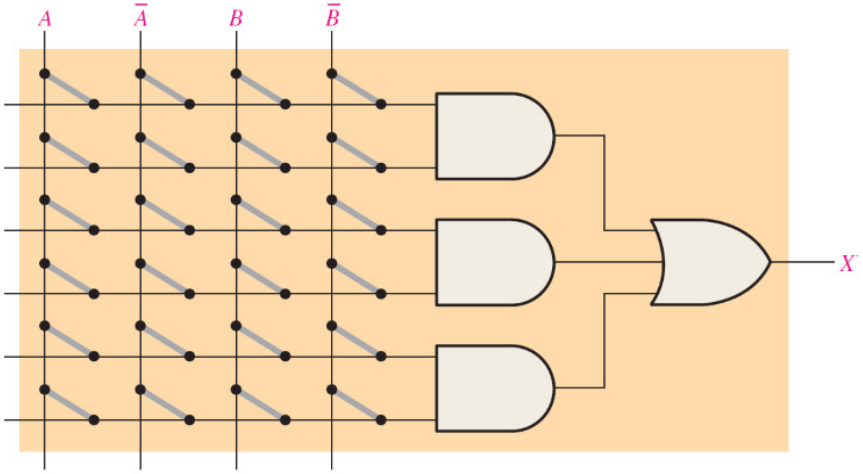
- PAL sistemi, ***ikinci düzleminde sigorta gecikmeleri olmayacağından*** PLA'dan daha hızlı işlem yapar.

- Ancak ***bu durum, PLA'ya göre daha az tasarım esnekliği sağlar.***

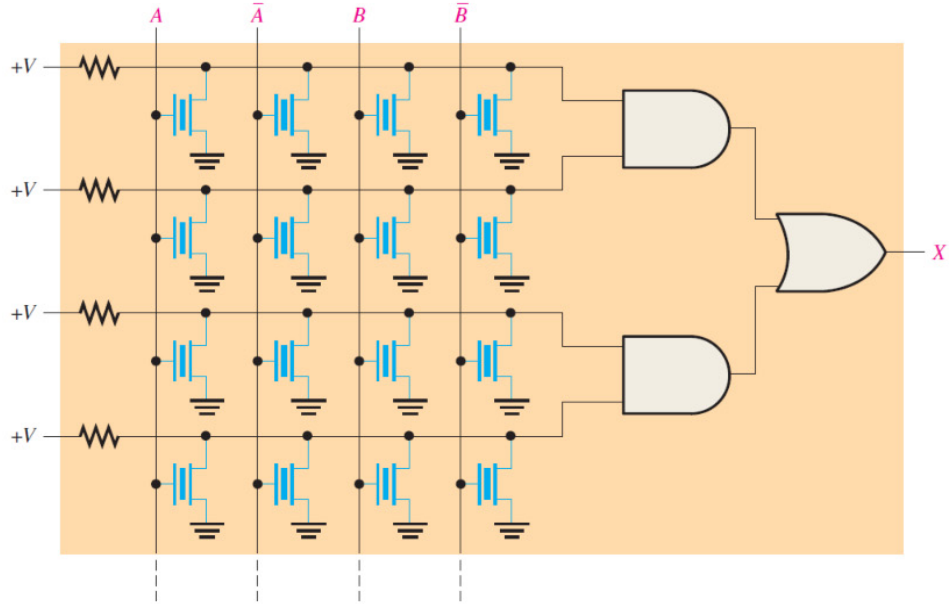
1.4 Programlanabilir Mantık Cihazları (PLD)

PAL (Programmable Array Logic) ve GAL (Generic Array Logic):

- PAL entegreleri ***yalnızca bir kez programlanabilen (OTP → One Time Programmable) yapıya sahiptir.***



İki girişli PAL örneği



İki girişli GAL örneği

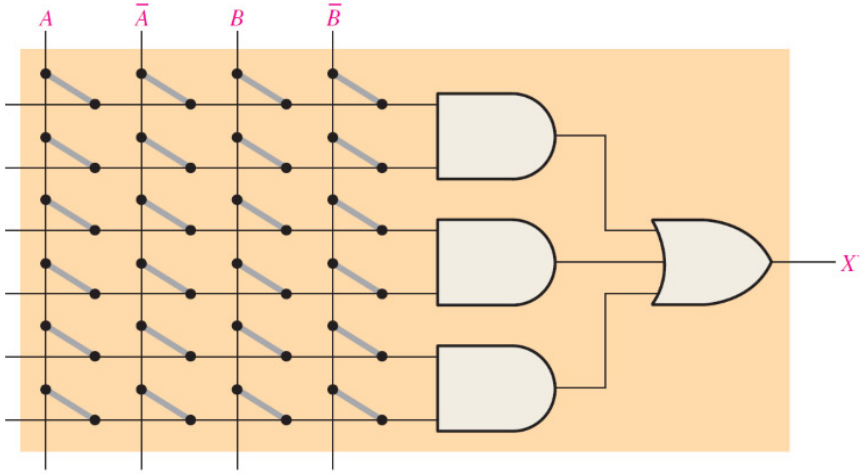
PAL ve GAL yapılarına dair basitleştirilmiş içyapı

1.4 Programlanabilir Mantık Cihazları (PLD)

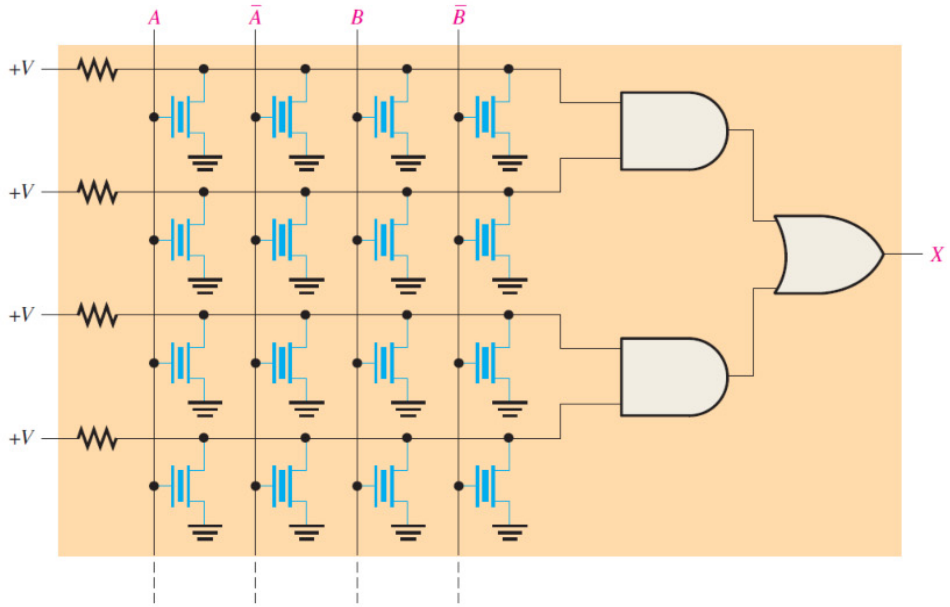
PAL (Programmable Array Logic) ve GAL (Generic Array Logic):

- GAL entegreleri ise ***tekrar programlanabilir bir AND düzlemi içerir.***

- GAL mimarisinin, PAL yapısına göre diğer farkı ise ***kombinasyonel fonksiyonun gerçekleşmesinde kullanılan sigorta yapılarıdır.***



İki girişli PAL örneği



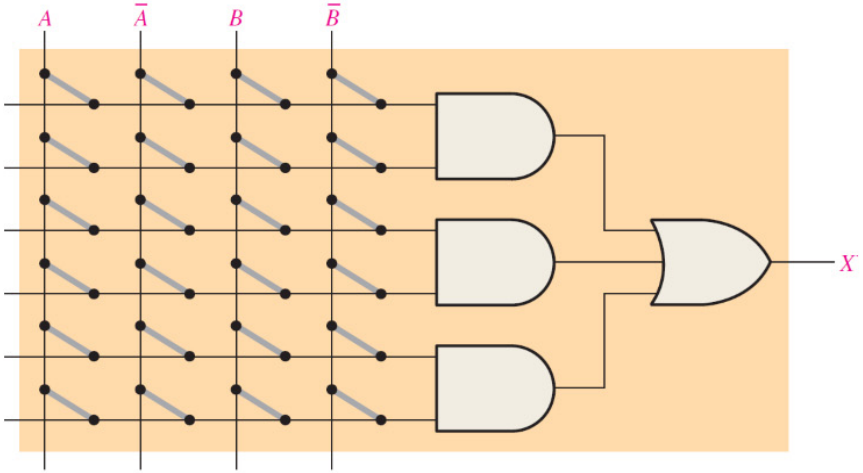
İki girişli GAL örneği

PAL ve GAL yapılarına dair basitleştirilmiş içyapı

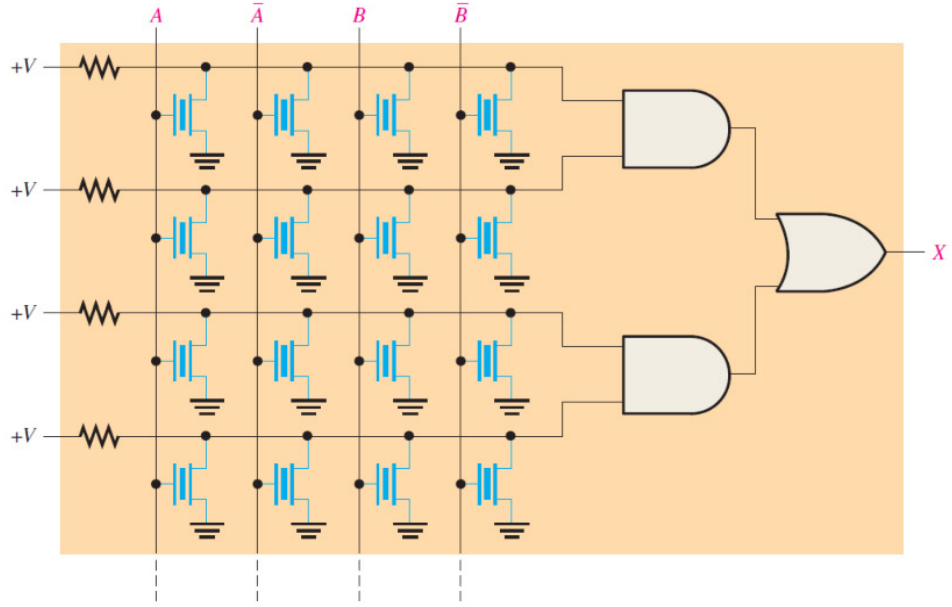
1.4 Programlanabilir Mantık Cihazları (PLD)

PAL (Programmable Array Logic) ve GAL (Generic Array Logic):

- GAL mimarisinin yeniden programlanabilmesinin nedeni, ***sigorta yerine (EEPROM'un programlama mantığına benzer) MOSFET tabanlı yeniden programlanabilen işlem teknolojisinin (E^2CMOS) kullanımıdır.***



İki girişli PAL örneği



İki girişli GAL örneği

PAL ve GAL yapılarına dair basitleştirilmiş içyapı

1.4 Programlanabilir Mantık Cihazları (PLD)

PAL (Programmable Array Logic) ve GAL (Generic Array Logic):

- PAL ve GAL mimarileri içinde OR düzlemi ve ilgili çıkışların bulunduğu birim ***makrosel*** olarak bilinir.

- Makrosel sistem çıkışında, çıktılar sürekli üretilebildiği gibi ***bazı makrosel tiplerinde girdiler de alınabilmektedir***.

- Ayrıca bir makrosel; kombinasyonel mantık, hafıza mantığı veya bu ikisinin kombinasyonunu sağlayacak şekilde yapılandırılabilir.

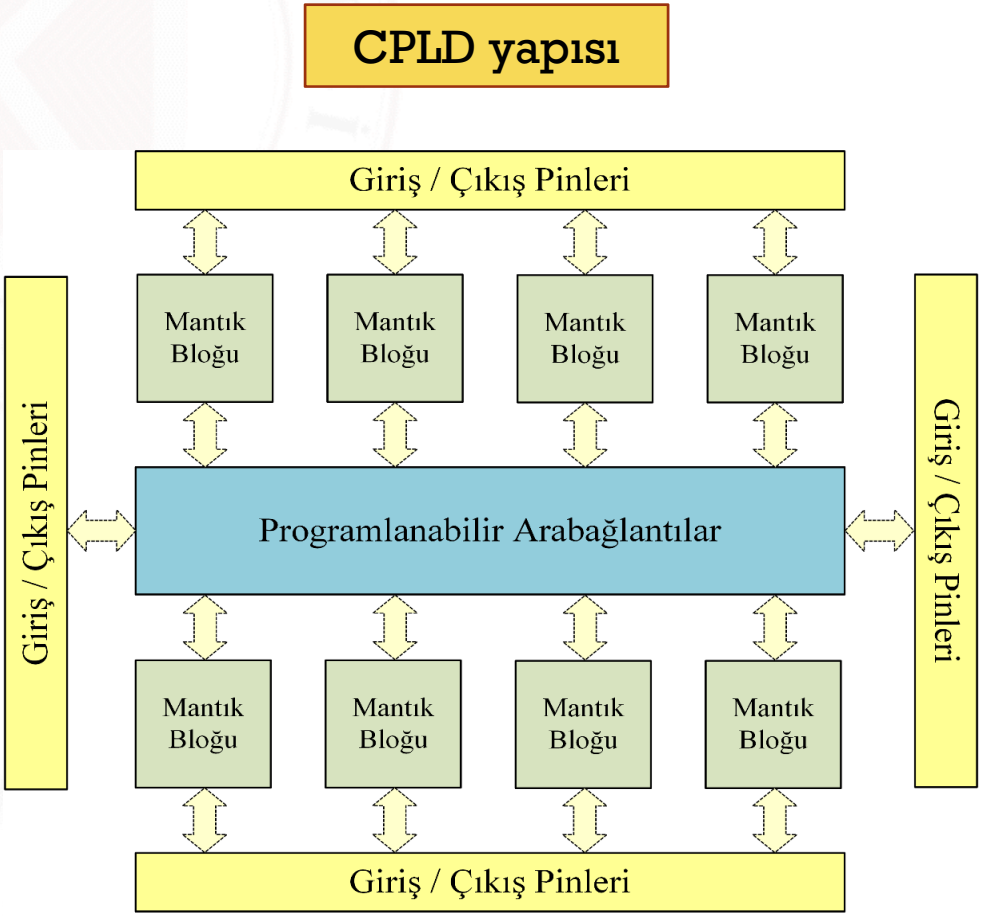
- PAL ve GAL mimarileri, ***SPLD*** (***Simple Programmable Logic Device***) kapsamındadır.

1.4 Programlanabilir Mantık Cihazları (PLD)

CPLD (Complex Programmable Logic Device):

- ***PLA ve PAL yapıları*** küçük devreler için uygulanabilir olup, çok fazla girdi / çıktı gerektiren ***büyük ve kompleks devre tasarımları için uygulanabilir değildir***.

- **Kompleks Programlanabilir Mantık Aygıtı (CPLD)**; karmaşık devre tasarımları için, ***tekil entegre içinde birbiriyle etkileşimli birden fazla PLA'nın işlevini yerine getiren tümleşik entegredir***.

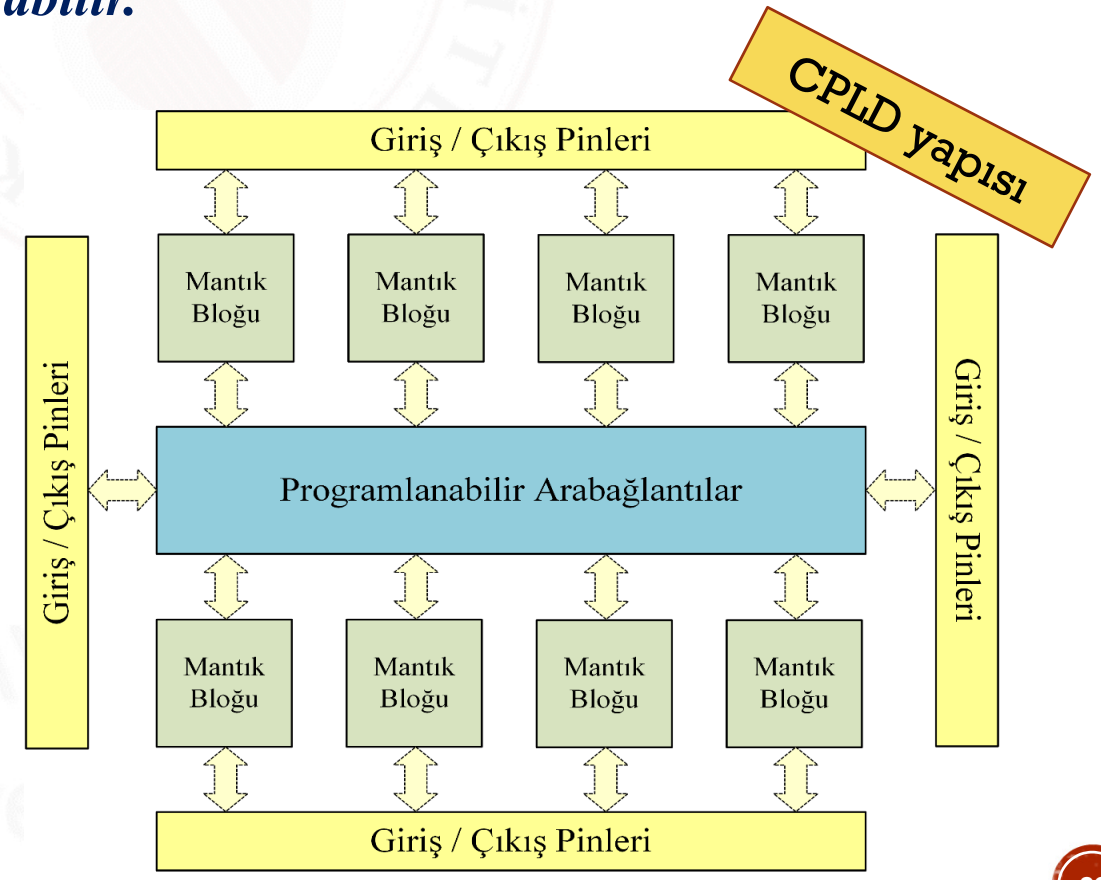


1.4 Programlanabilir Mantık Cihazları (PLD)

CPLD (Complex Programmable Logic Device):

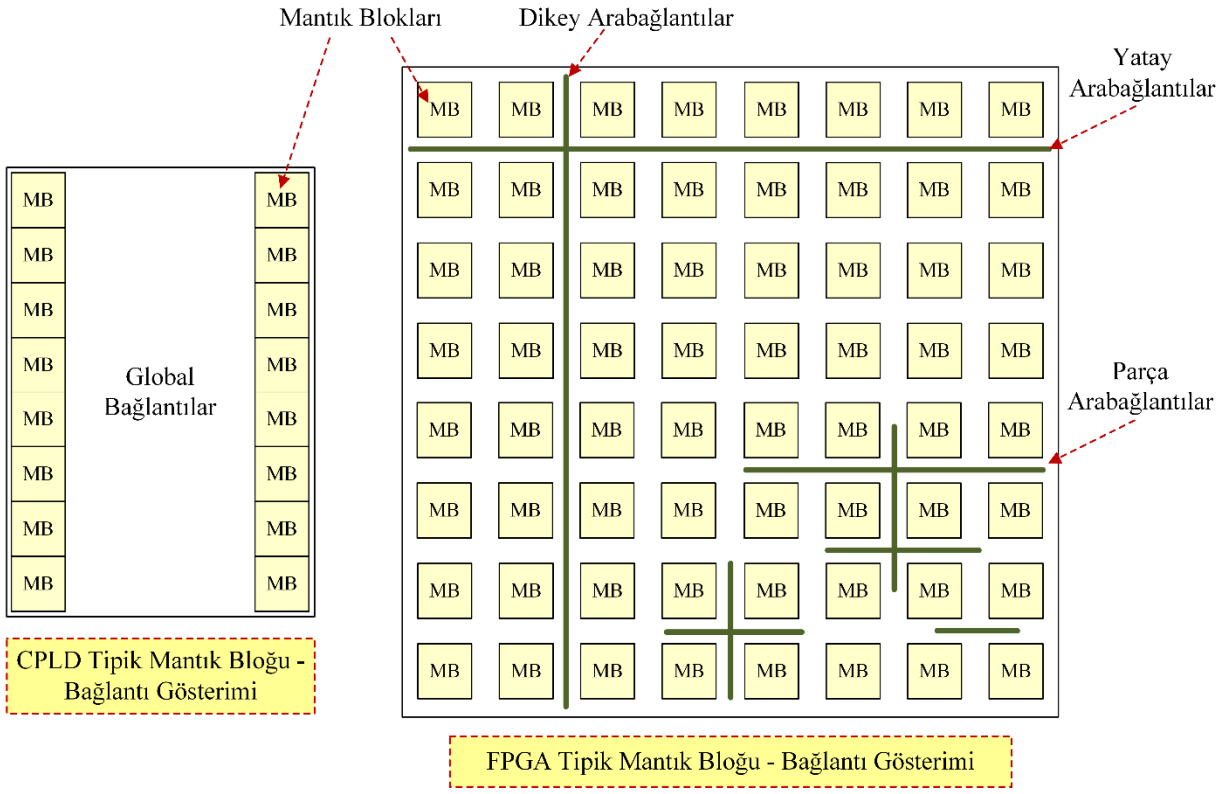
- PAL ve PLA yapıları ile ***yalnızca birkaç yüz mantık kapısı eşdeğerinde olan küçük devreler tasarlanabilir.***

- CPLD ile **yüzbinlerce mantık kapıları eşdeğerindeki komplike devre tasarımları gerçekleştirilebilir.**



1.4 Programlanabilir Mantık Cihazları (PLD)

CPLD (Complex Programmable Logic Device):



CPLD Tipik Mantık Bloğu - Bağlantı Gösterimi

FPGA Tipik Mantık Bloğu - Bağlantı Gösterimi

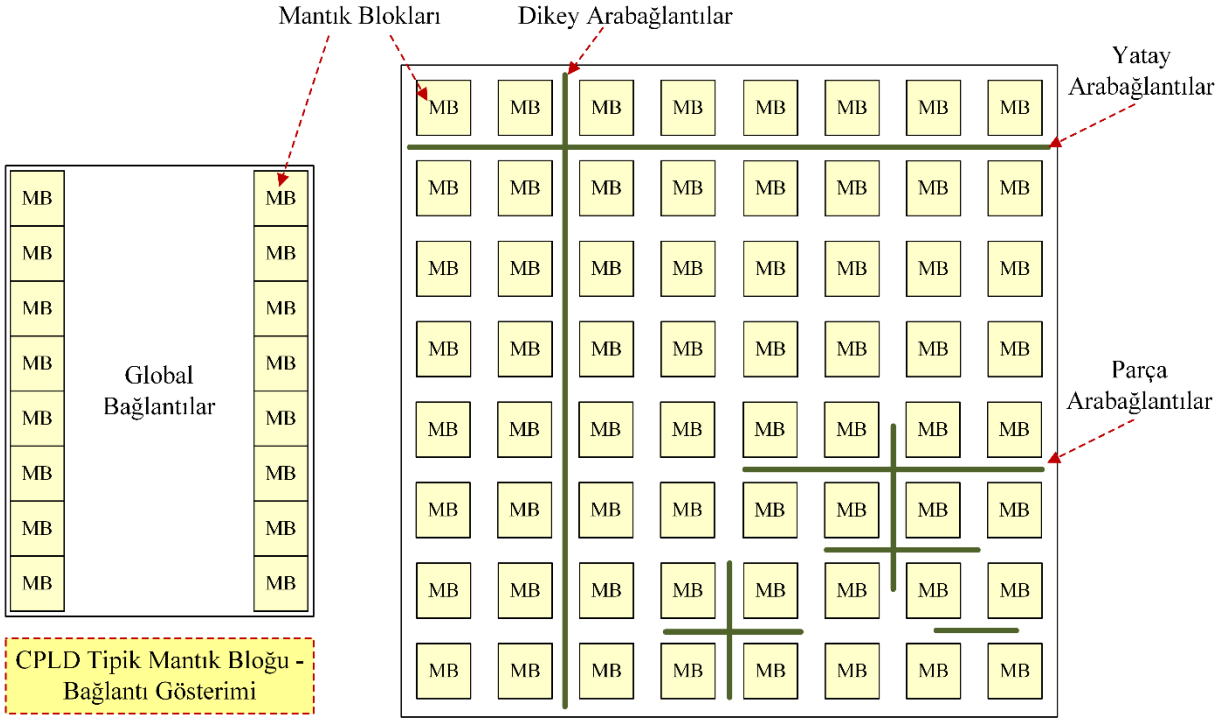
Mantık blokları (hücreleri) ve ara bağlantıların, CPLD ve FPGA yapılarındaki tipik dizilişi

- ***CPLD yapılarında gerekli mantık hücresi sayısı arttıkça***, mantık hücrelerinin diziliş şekli ve tek bir genel ara bağlantının olmasından ötürü, ***hücreler arası bağlantı sayısı da katlanarak artar***.

- Bu durum ise ***çok büyük tasarımlar için önemli bir handikap teşkil eder ve CPLD'leri kullanışsız hale getirir.***

1.4 Programlanabilir Mantık Cihazları (PLD)

CPLD (Complex Programmable Logic Device):



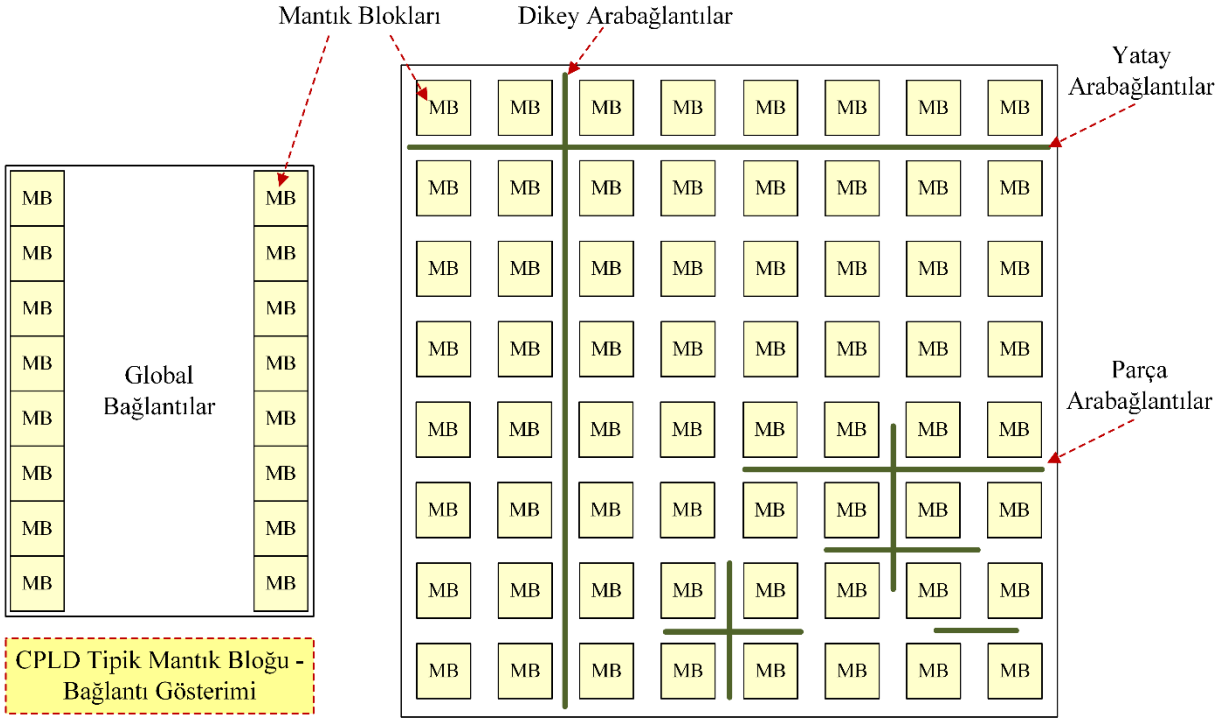
- FPGA yapısında mantık hücreleri, CPLD'lerden farklı olarak ***dizi biçiminde*** yerleştirilir.

- FPGA içindeki mantık hücreleri arası bağlar, ***yatay ve dikey programlanabilir ara bağlantılar*** vasıtasıyla sağlanır.

Mantık blokları (hücreleri) ve ara bağlantıların, CPLD ve FPGA yapılarındaki tipik dizilişi

1.4 Programlanabilir Mantık Cihazları (PLD)

CPLD (Complex Programmable Logic Device):



CPLD Tipik Mantık Bloğu - Bağlantı Gösterimi

FPGA Tipik Mantık Bloğu - Bağlantı Gösterimi

- Mantık hücreleri arası ***gerekli olan bağlantı sayısı, hücre sayısına paralel artış*** gösterir.

- FPGA'lerin bu özellikleri sayesinde, ***büyük tasarımlar için CPLD'lerde karşılaşılan bağlantı sorunları minimize edilir.***

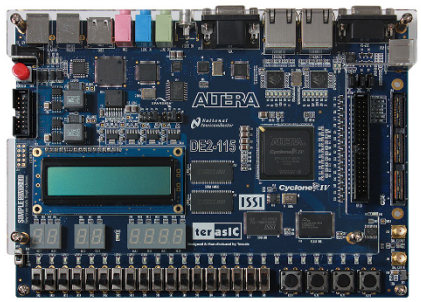
Mantık blokları (hücreleri) ve ara bağlantıların, CPLD ve FPGA yapılarındaki tipik dizilişi

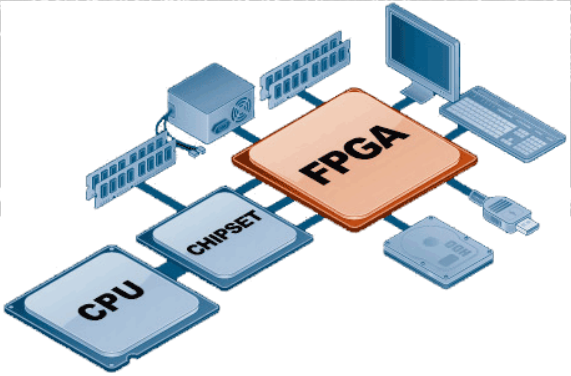
SONRAKİ DERS KONUSU

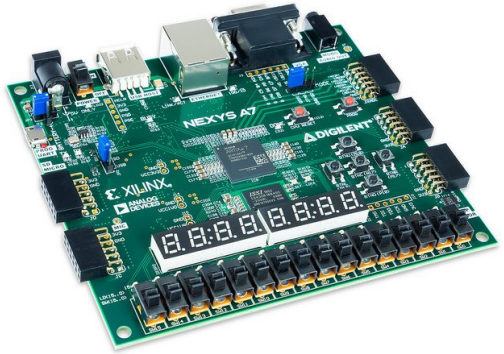
2- FPGA İÇYAPISI VE ÖZELLİKLERİ

İLERİ SAYISAL SİSTEMLER

2- FPGA İçyapısı ve Özellikleri







Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Klasik CPLD mimarisi, ***programlanabilir ara bağlantılara sahip PAL / GAL*** veya ***PLA tipi*** mantık bloklarından oluşur.

- Temel olarak FPGA;

1. Mimaride farklılık gösterir,
2. PAL / PLA tipi dizileri kullanmaz (daha komplike ve kullanışlı birimler mevcut),
3. CPLD'lerden çok daha fazla yoğunluğa sahiptir.

- Tipik bir FPGA, tipik bir CPLD'den çok daha fazla sayıda eşdeğer kapıya sahiptir.

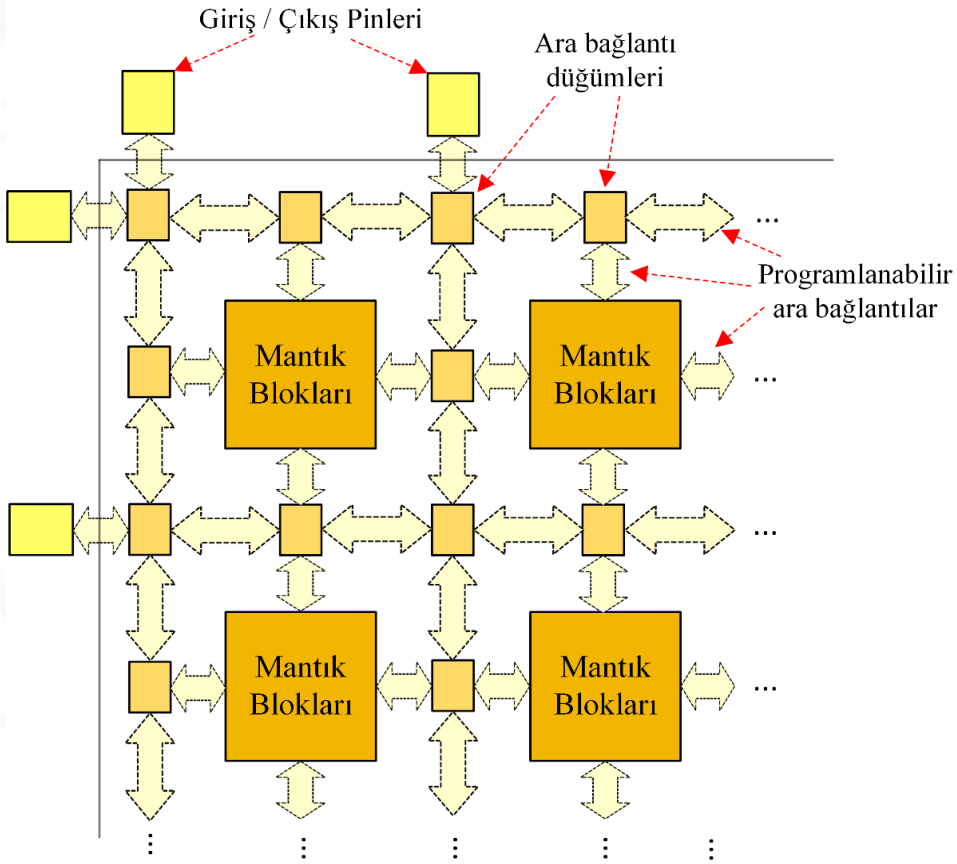
- FPGA'lerdeki mantık üreten unsurlar genellikle CPLD'lerden ***çok daha küçüktür*** ve bunlardan ***çok daha fazlası vardır***.

2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- FPGA'deki programlanabilir bağlantılar, temel bir satır ve sütun (bağlantısı) üzerinden ayarlanır.

- FPGA'ler, ***sayısal devrelerin gerçekleştirilmesi ve test edilmesi için geliştirilen programlanabilir yongalardır.***

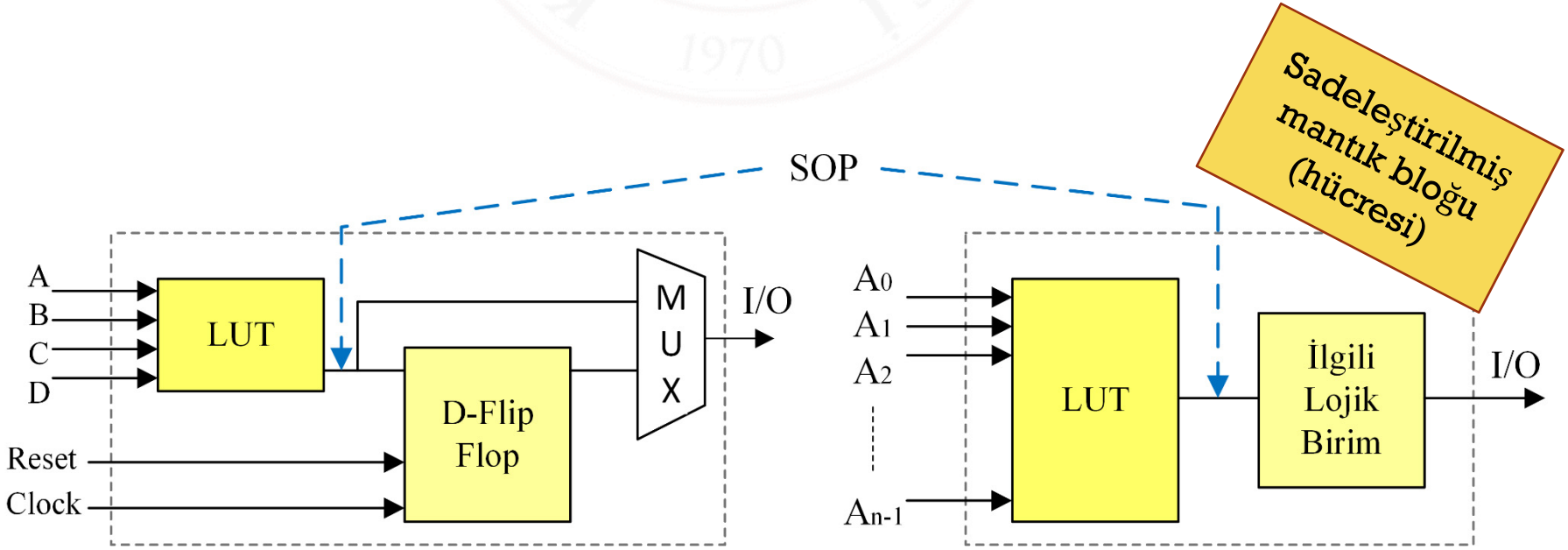
- FPGA yongaları üç bölümden meydana gelir:
  - ***Programlanabilir mantık blokları*** (*Configurable Logic Block→CLB*)
  - Mantık bloklarını birbirine bağlamak için kullanılan ***programlanabilir ara bağlantılar*** (*Interconnect Resources*)
  - FPGA'in dış dünya ile bağlantısını sağlayan ve ***programlanabilen giriş-çıkış birimleri*** (*I/O Elements*)



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Programlanabilir mantık blokları (Configurable Logic Blocks → CLB), FPGA'lerin temel yapısını teşkil eder.

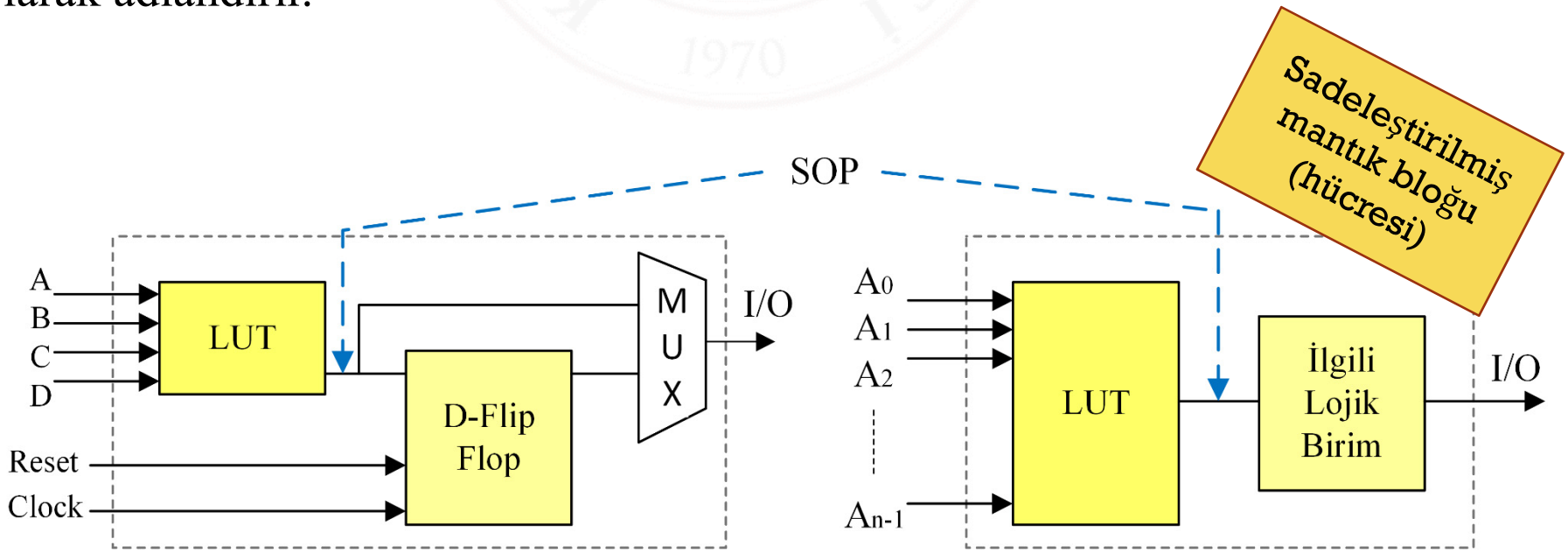
- Bu yapılarda bir veya birkaç adet ***doğruluk tablosu*** (***Look Up Table*** → ***LUT***), ***çoklayıcı*** (***multiplexer***) ve ***saklayıcı*** (***flip flop***) mevcuttur.



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Mantıksal işlemler bu birimlerde gerçekleştirilir ve her üretici firma mantık bloklarını farklı isimlendirmektedir.

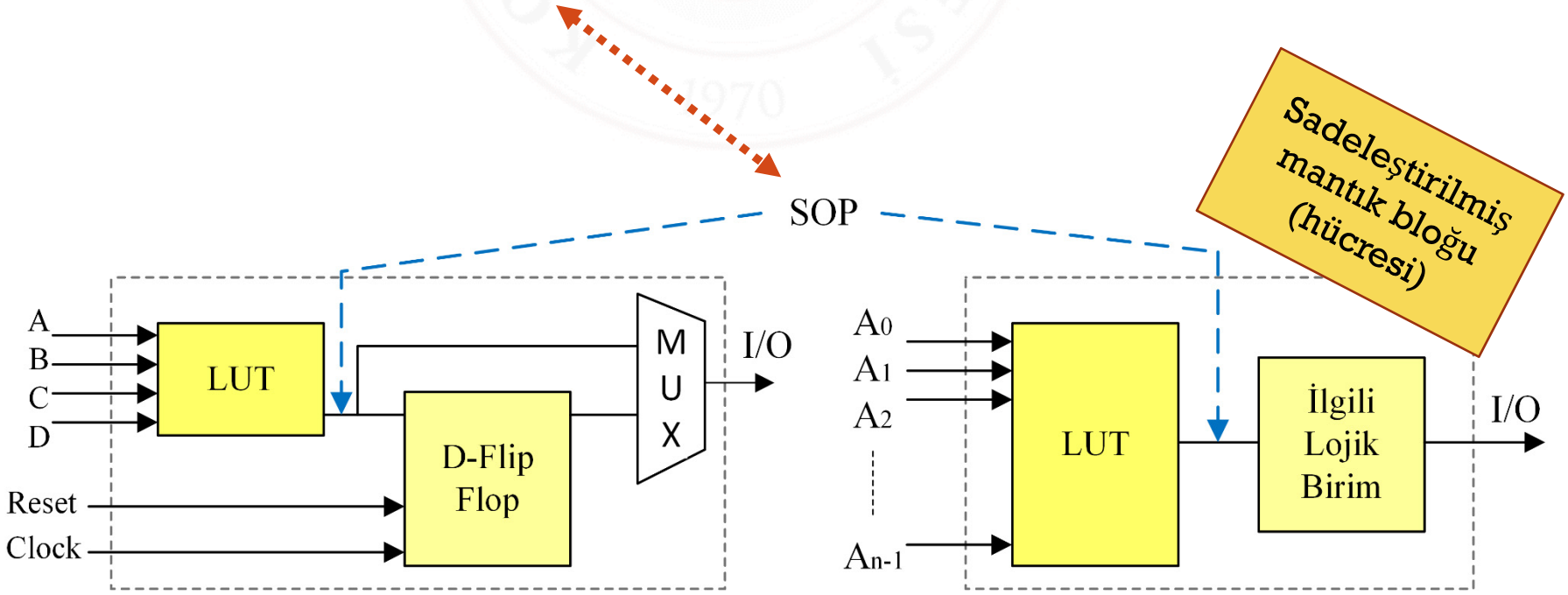
- ***Xilinx*** firması mantık bloklarını ***mantık hücresi*** (***logic cell***) olarak isimlendirirken, ***Altera*** firması bu birimleri ***mantık elemanı*** (***logic element***) olarak adlandırır.



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Programlanabilir mantık blokları, ***minimum terimler kanonik açılımı (çarpımların toplamı)*** temelli çalışır.

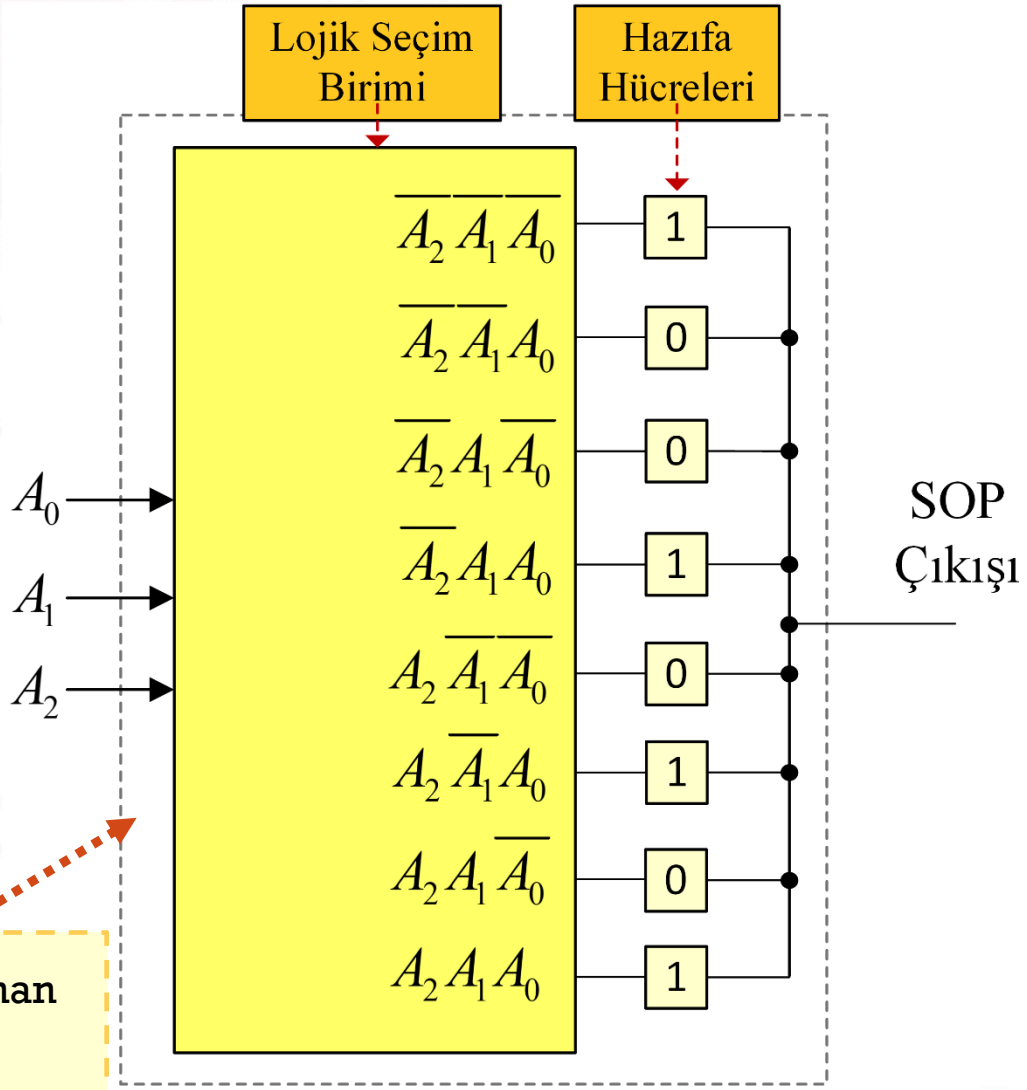
- Bu teorem ***SOP*** (***Sum of Products***) olarak da bilinir.



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- LUT yapısında ***lojik seçim birimi*** ve ***hafıza hücreleri*** yer almaktadır.

- Buna göre;
  - Giriş değişken sayısı '***n***' olarak belirtilirse, LUT' da bulunan hafıza hücrelerinin sayısı 2^n olacaktır.
  - 3 girişli bir LUT örneği yan tarafta görüldüğü gibidir.



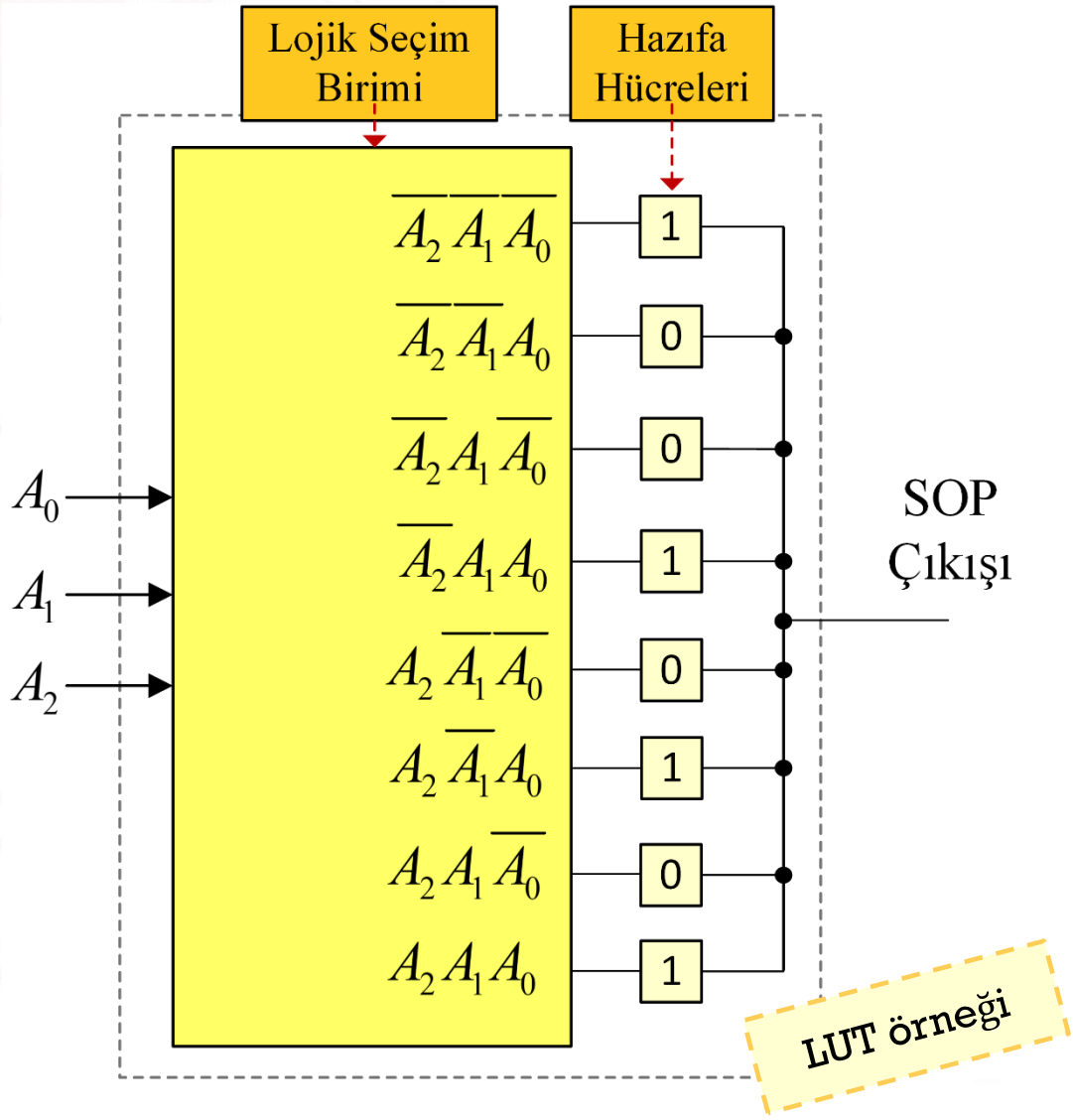
Belirli bir SOP çıkışı için programlanan LUT yapısı örneği

2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Hafıza hücrelerindeki herbir '0' terimin kullanılmadığını belirtir ve herbir '1' ise terimin SOP çıkışında olacağını gösterir.

- Buna göre 3 girişe karşılık, 8 farklı hafıza hücresi olmalıdır.

- SOP çıkışı ise 8 terime kadar farklı durumu barındırabilir.

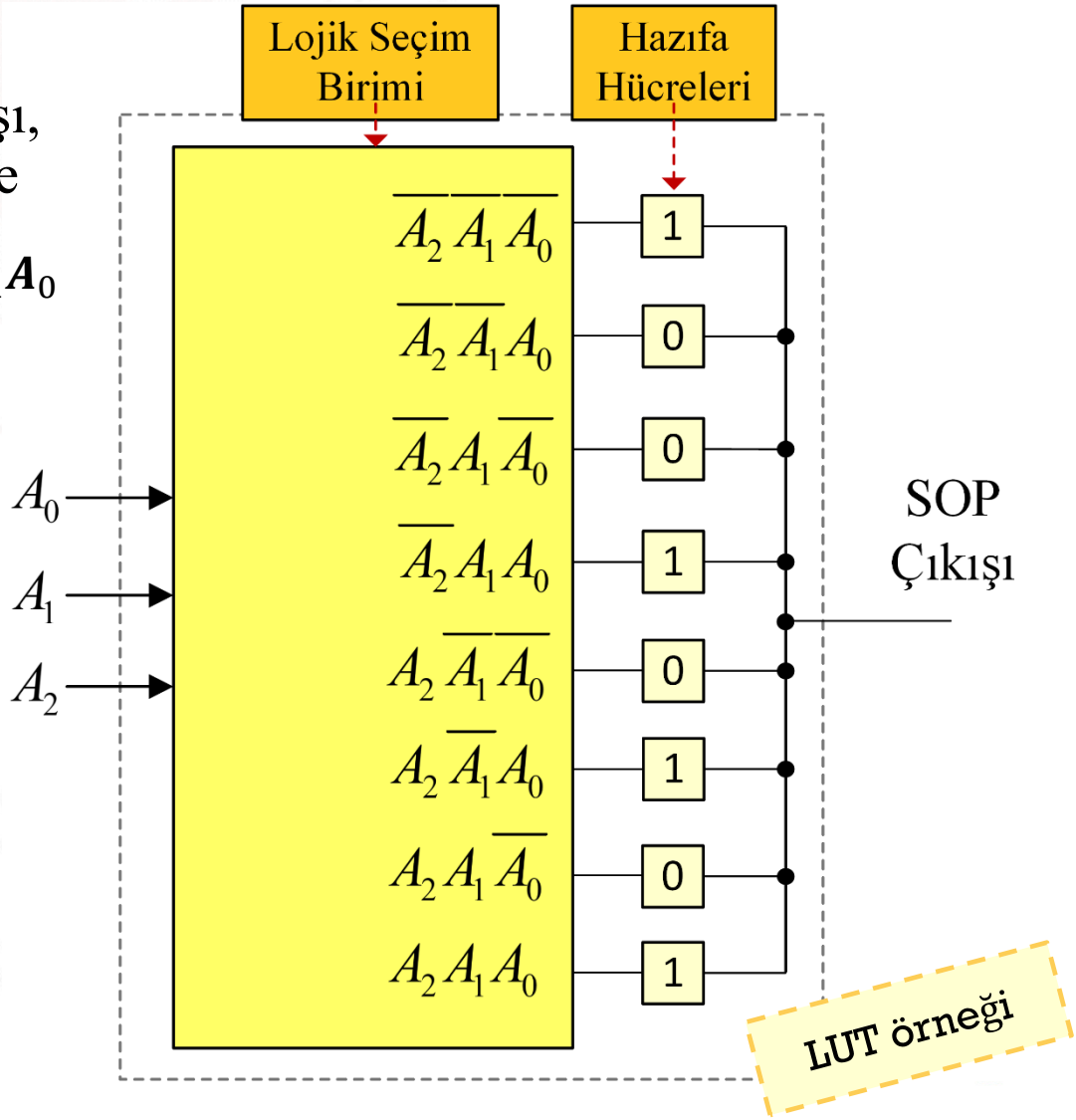


2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Örnek içinde etkin SOP çıkışı, hafıza hücrelerinin durumuna göre

$\overline{A_2}\overline{A_1}\overline{A_0} + \overline{A_2}A_1A_0 + A_2\overline{A_1}A_0 + A_2A_1A_0$

şeklinde elde edilir.



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- FPGA içindeki mantık blokları 4 farklı işlevde programlanabilmektedir:
  - Normal mod
  - Genişletilmiş LUT modu
  - Aritmetik mod
  - Paylaşılan aritmetik mod

- Bir ***mantık bloğu***, bu dört moda ek olarak ***sayıcı*** ve ***ötelemeli register*** oluşturmak için bir ***register dizisi*** olarak kullanılabilir.

- Normal mod, ilk olarak kombinasyonel lojik fonksiyonların üretilmesi için kullanılır.

- Bir ***mantık bloğu***; ***iki adet LUT birimi*** ile ***bir veya iki kombinasyonel çıkış fonksiyonu*** sağlayabilir.

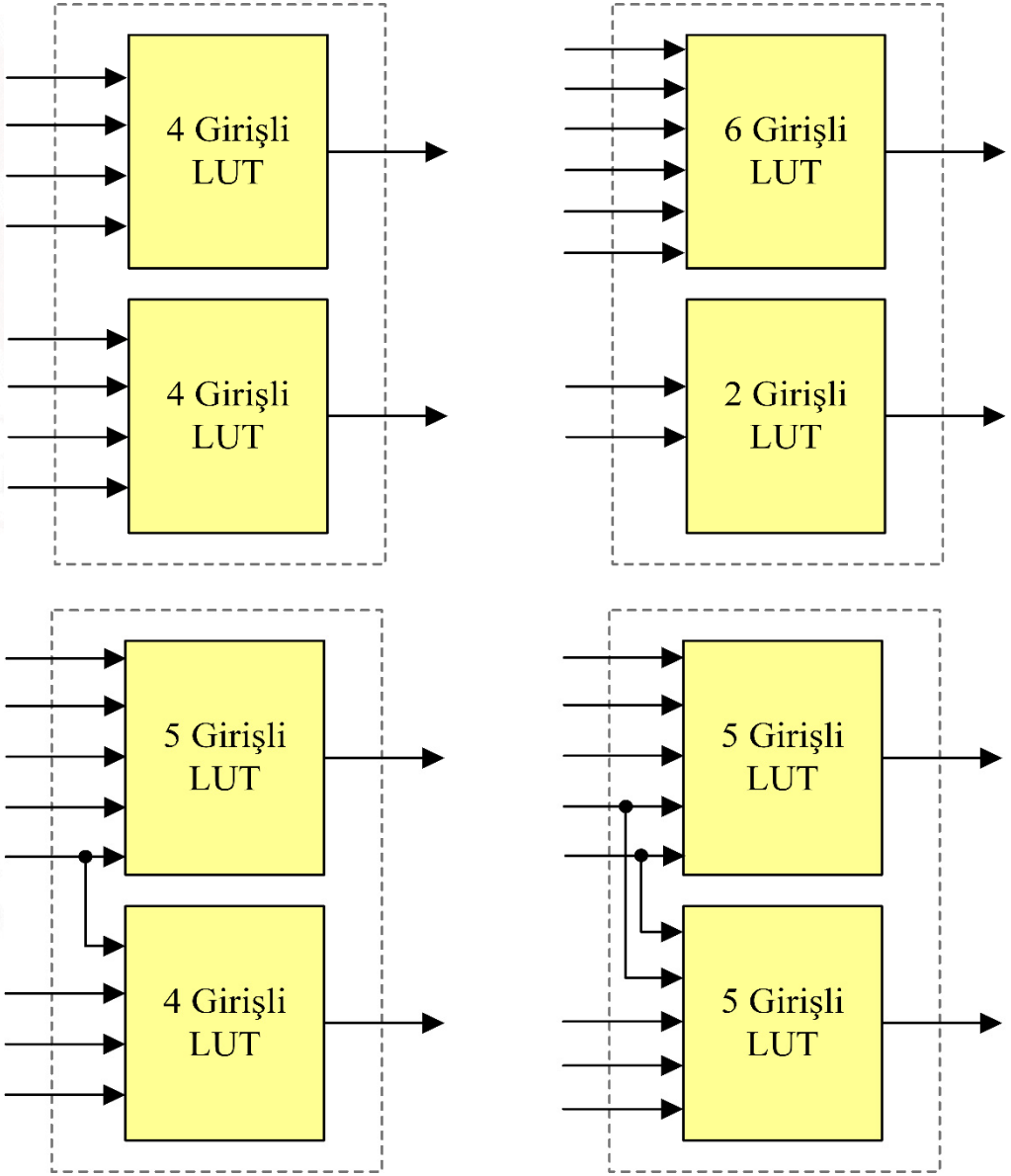
2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Bir ***mantık bloğu***; ***iki adet LUT birimi*** ile ***bir veya iki kombinasyonel çıkış fonksiyonu*** sağlayabilir.

- ***Normal mod kullanımı şu kurallara bağlıdır;***

- Herbir mantık bloğunda, LUT'lardaki girişlere göre iki farklı SOP çıkışı elde edilebilir.

(4+3'lük iki LUT içeren mantık bloğunda $\mathbf{SOP_1} = \overline{A_3}A_2A_1A_0$, $\mathbf{SOP_2} = \overline{A_2}A_1A_0$ gibi)

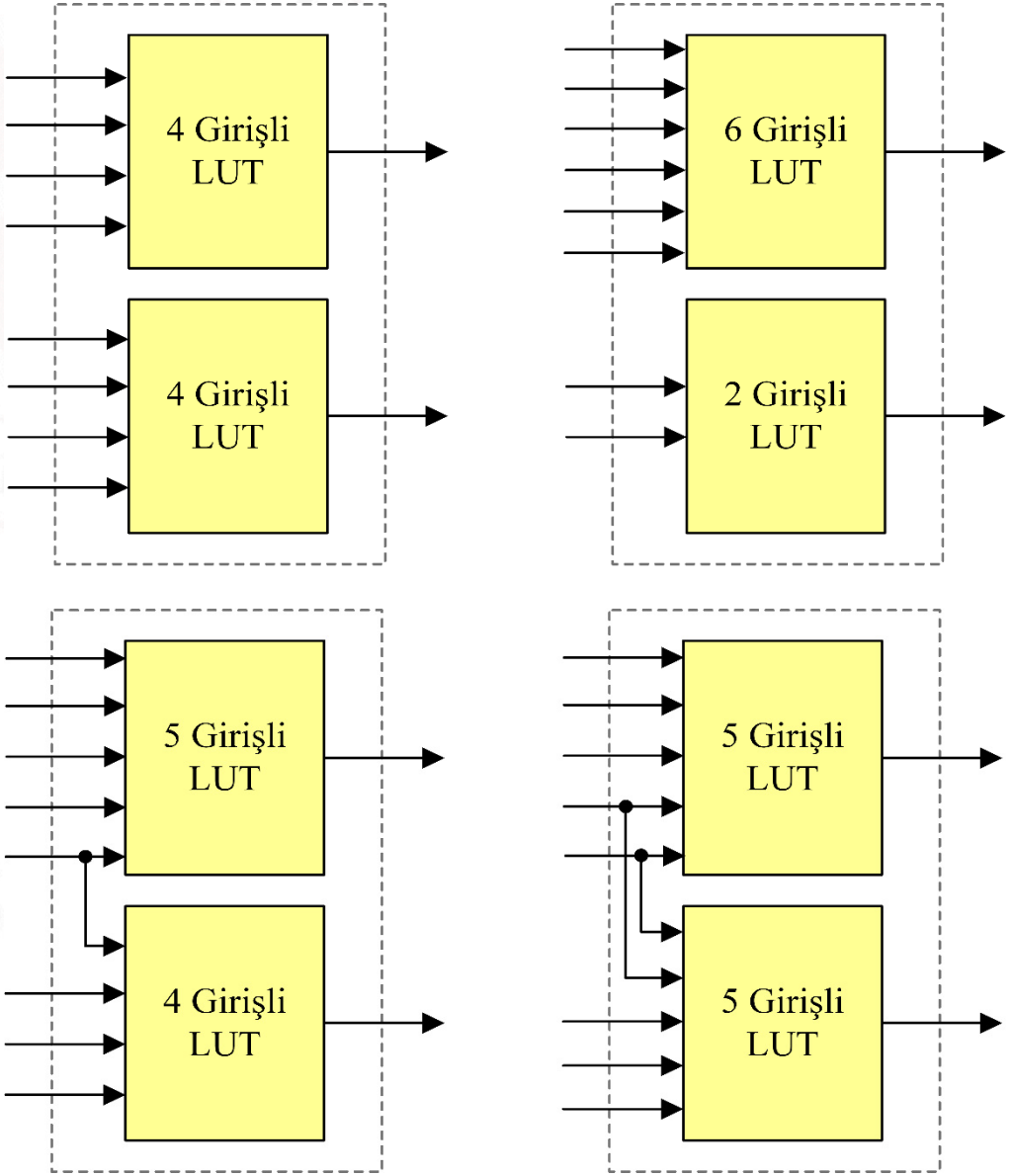


2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- ***Normal mod kullanımı şu kurallara bağlıdır;***

- Bir LUT çıkışında en fazla 6 değişkenli terim bulunabilir. Diğer bir deyişle, bir LUT yapısı en fazla 6 girişe sahip olabilir.

($\mathbf{SOP_1} = \overline{A_5}A_4\overline{A_3}A_2A_1A_0$ gibi)

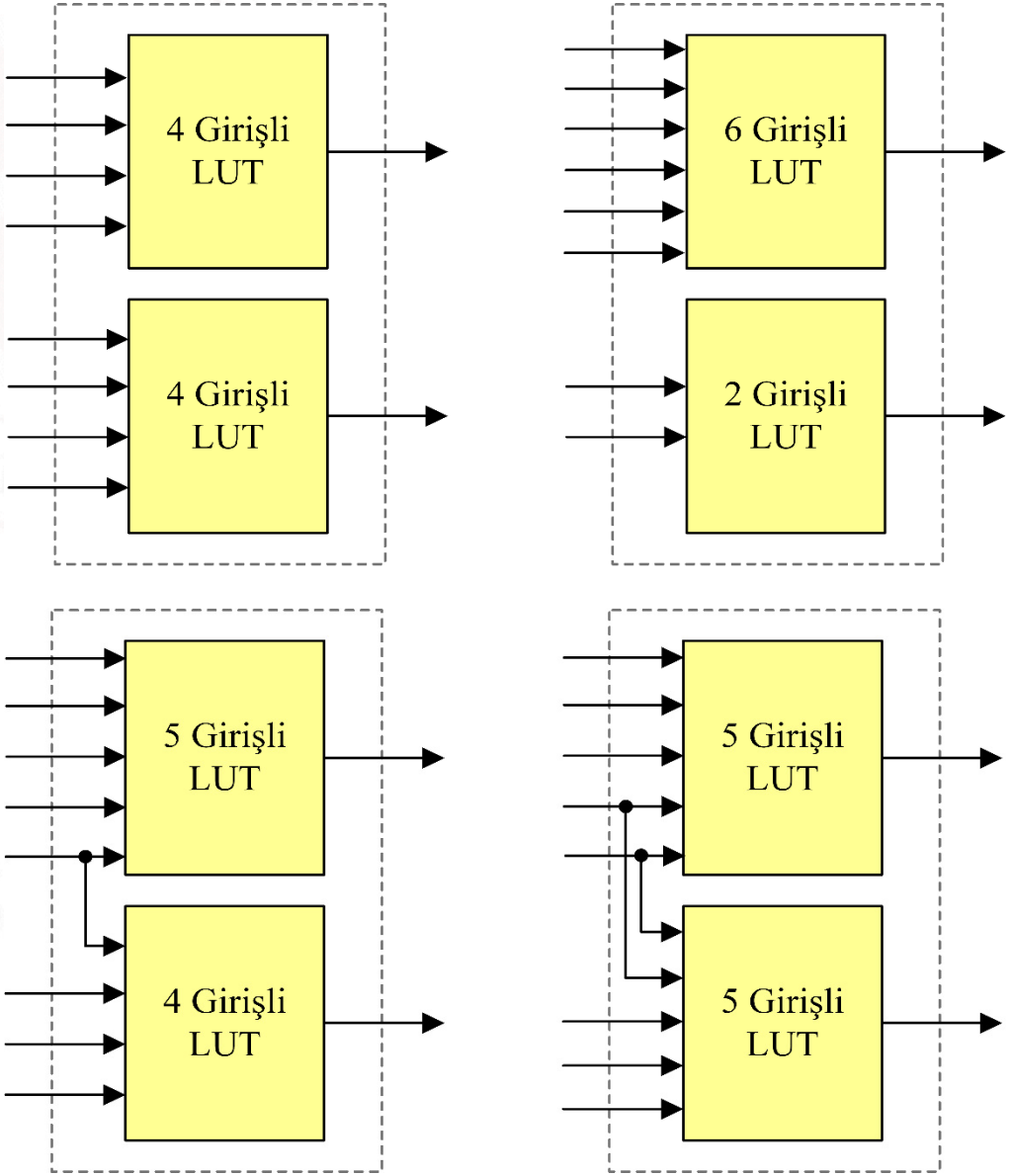


2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- ***Normal mod kullanımı şu kurallara bağlıdır;***

- Her iki LUT girişlerine totalde maksimum 8 değişken sunulabilir.

$(A_7, \ldots, A_1\, A_0)$

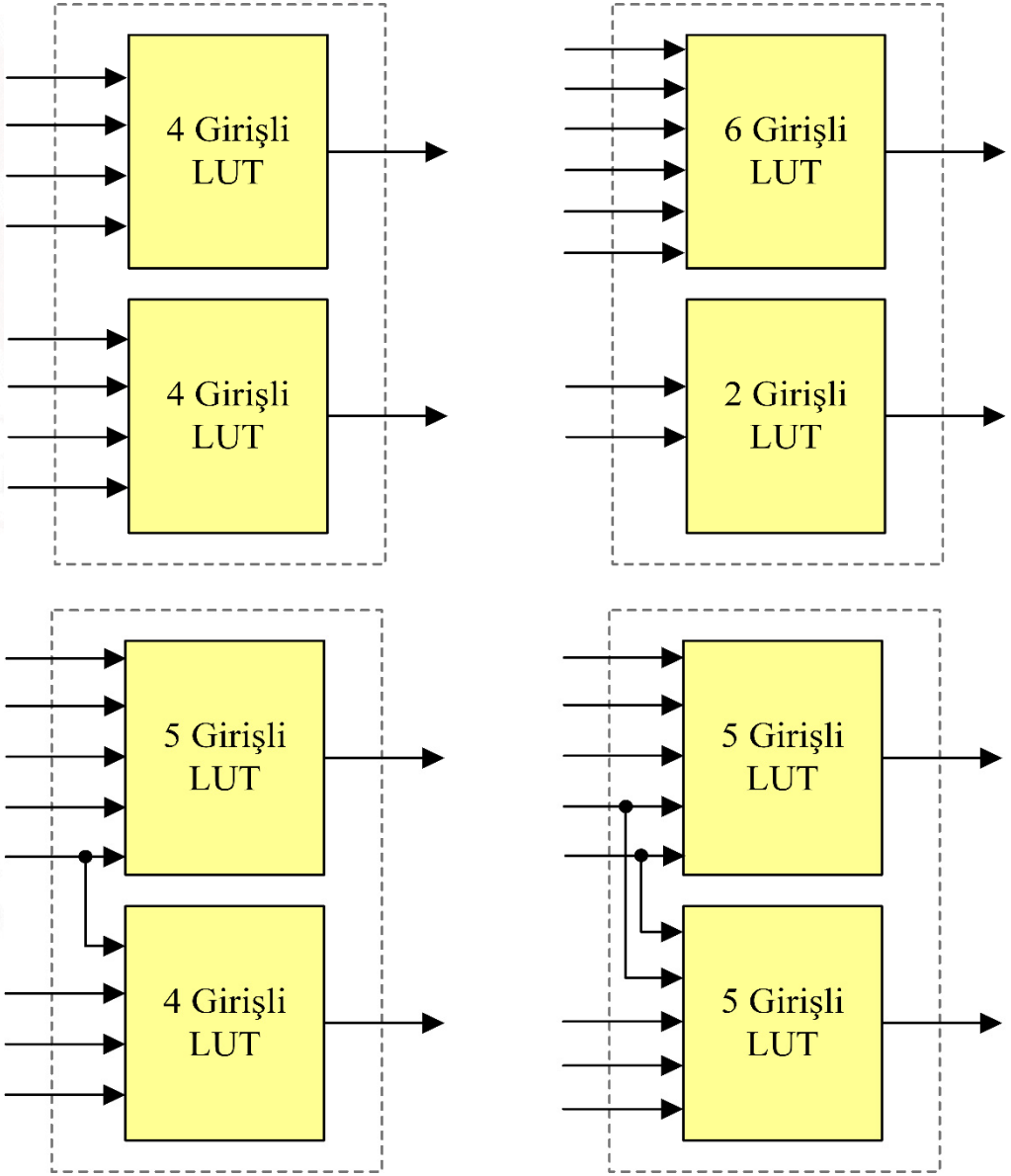


2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- ***Normal mod kullanımı şu kurallara bağlıdır;***

- **Genelde** ***4 veya daha az değişken içeren iki SOP fonksiyonu*** (toplamda 8 farklı veya daha az sayıda giriş değişkeni içeren iki LUT) ***için LUT'larda girişlerin paylaşımına gerek duyulmaz.***

- ***Ancak ortak kullanılacak değişken var ise paylaşım gerekir.***



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

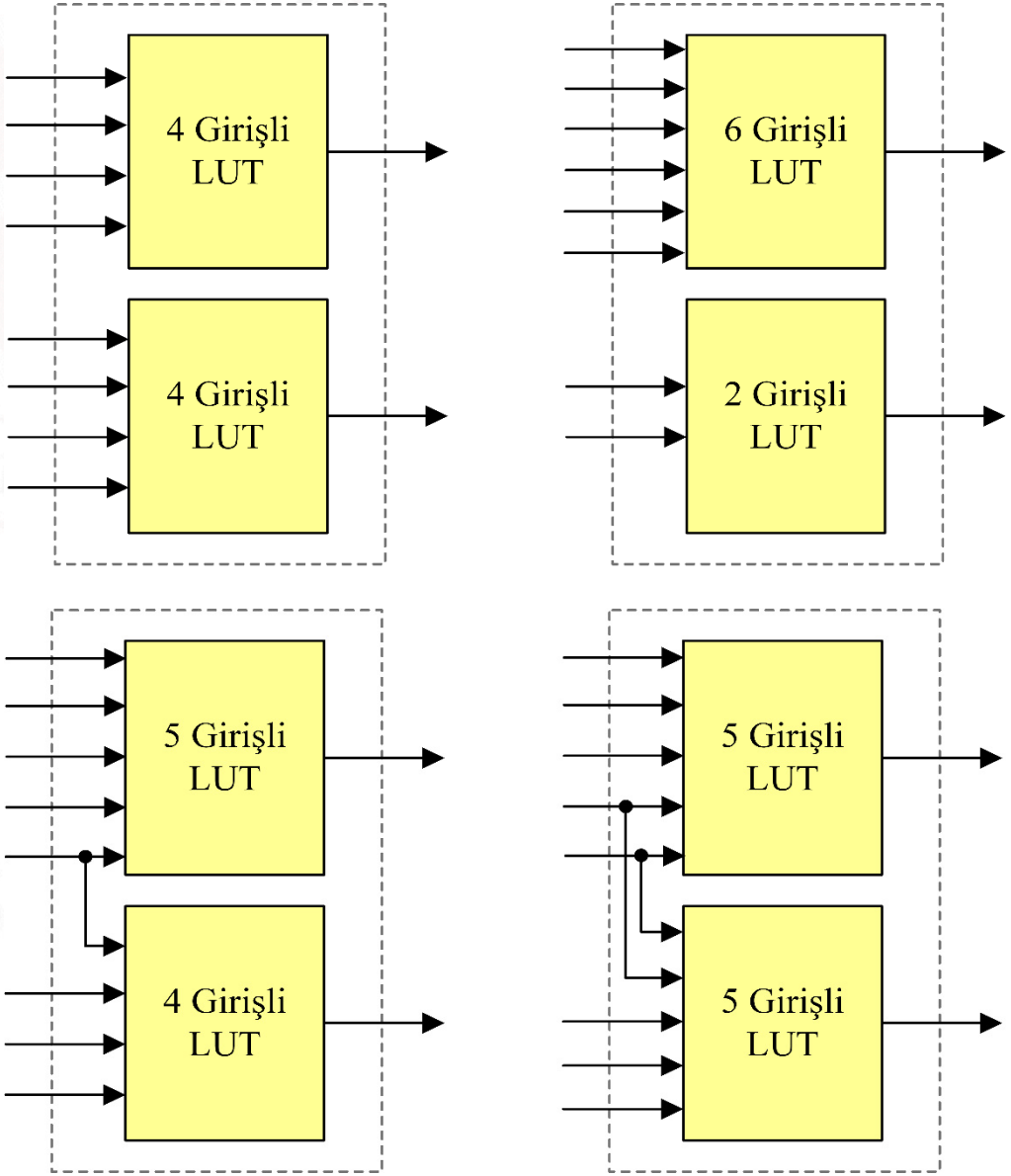
- ***Normal mod kullanımı şu kurallara bağlıdır;***

Örneğin; iki 4-değişkenli fonksiyon = 4+4 iki LUT,

bir 4-değişkenli ve bir 3-değişkenli fonksiyon = 4+3 iki LUT,

bir 3-değişkenli ve bir 3-değişkenli LUT = 3+3 iki LUT

Diğer bir deyişle, ***5 ve üzeri değişkenin kullanılacağı bir LUT için diğer LUT girişi kontrol edilmeli, toplam giriş sayısı 8'i geçerse paylaşım yapılmalıdır.***

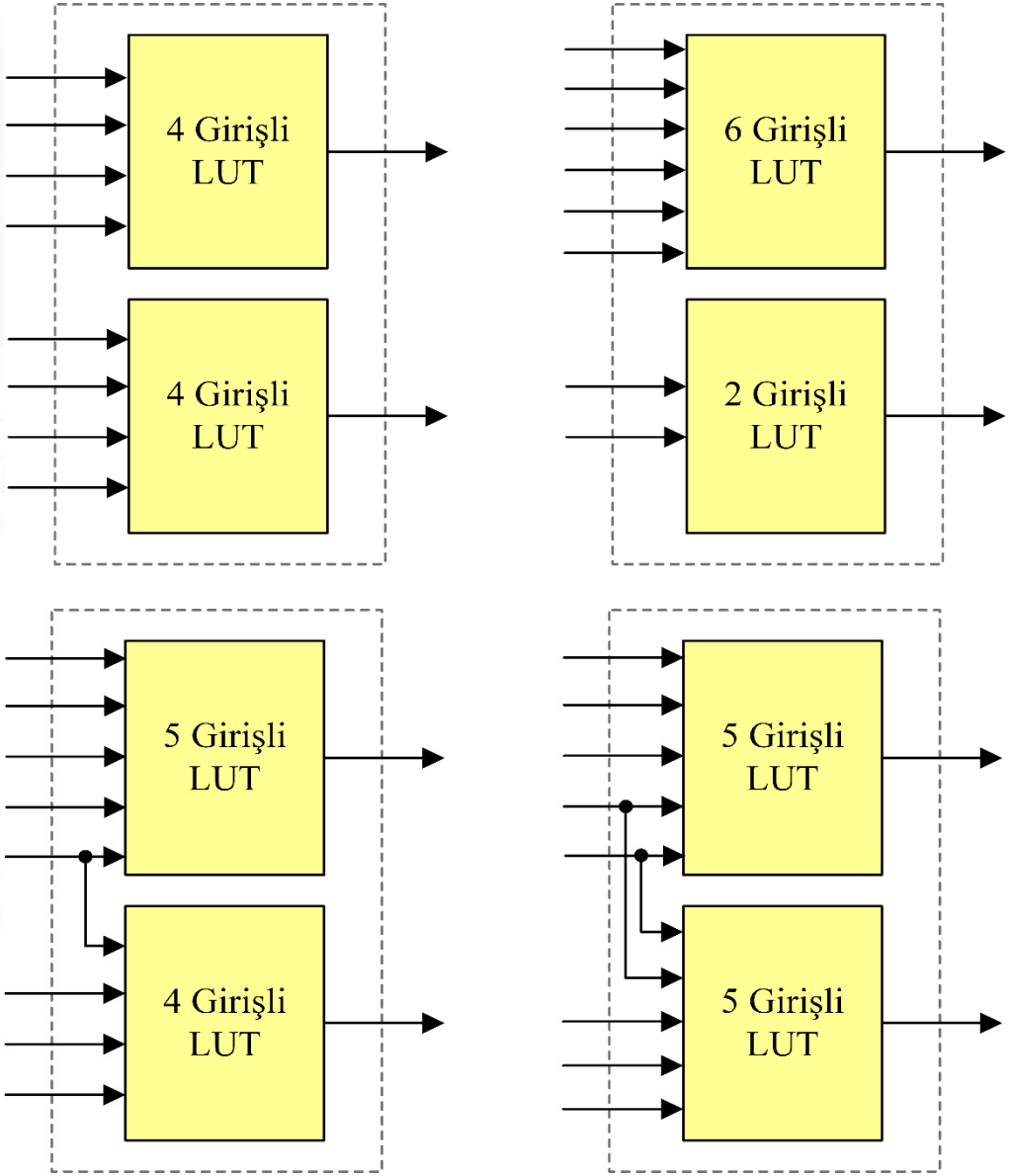


2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- ***Normal mod kullanımı şu kurallara bağlıdır;***

- Giriş paylaşımı yapıldığında maksimum ***8 farklı girişi destekleyecek farklı kombinasyonlarda bağlantı*** yapılabilir.

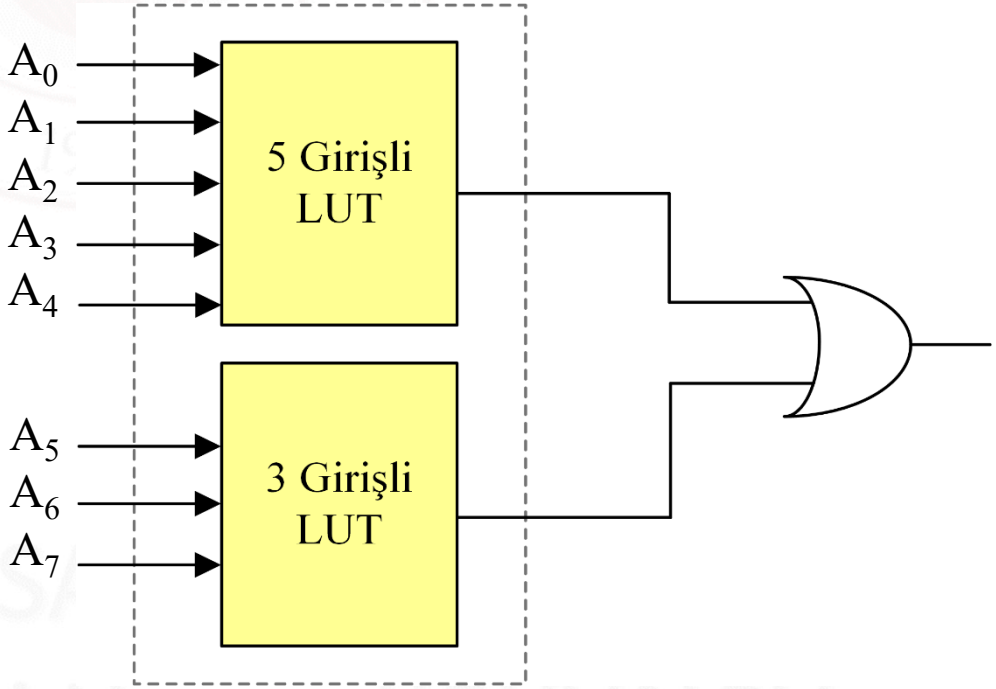
- Paylaşım yapıldığında ***herbir LUT yapısında maksimum 6'şar terimli fonksiyonlar*** (6+6 iki LUT) kullanılabilir.



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

Örnek 2.1: Normal modda işletilen bir mantık bloğunda, çıkışta 5-değişkenli (A_4,A_3,A_2,A_1,A_0) ve 3-değişkenli (A_7,A_6,A_5) terimlerin üretileceği SOP fonksiyonu için gerekli şemayı çiziniz, çalışma prensibini kısaca izah ediniz.

- İstenilen giriş değişkeni sayısı toplamda 8 olduğundan ve her SOP çıkışında farklı değişken çarpımları sağlanacağından, LUT'lar arası giriş paylaşımı yapılması zorunlu değildir.

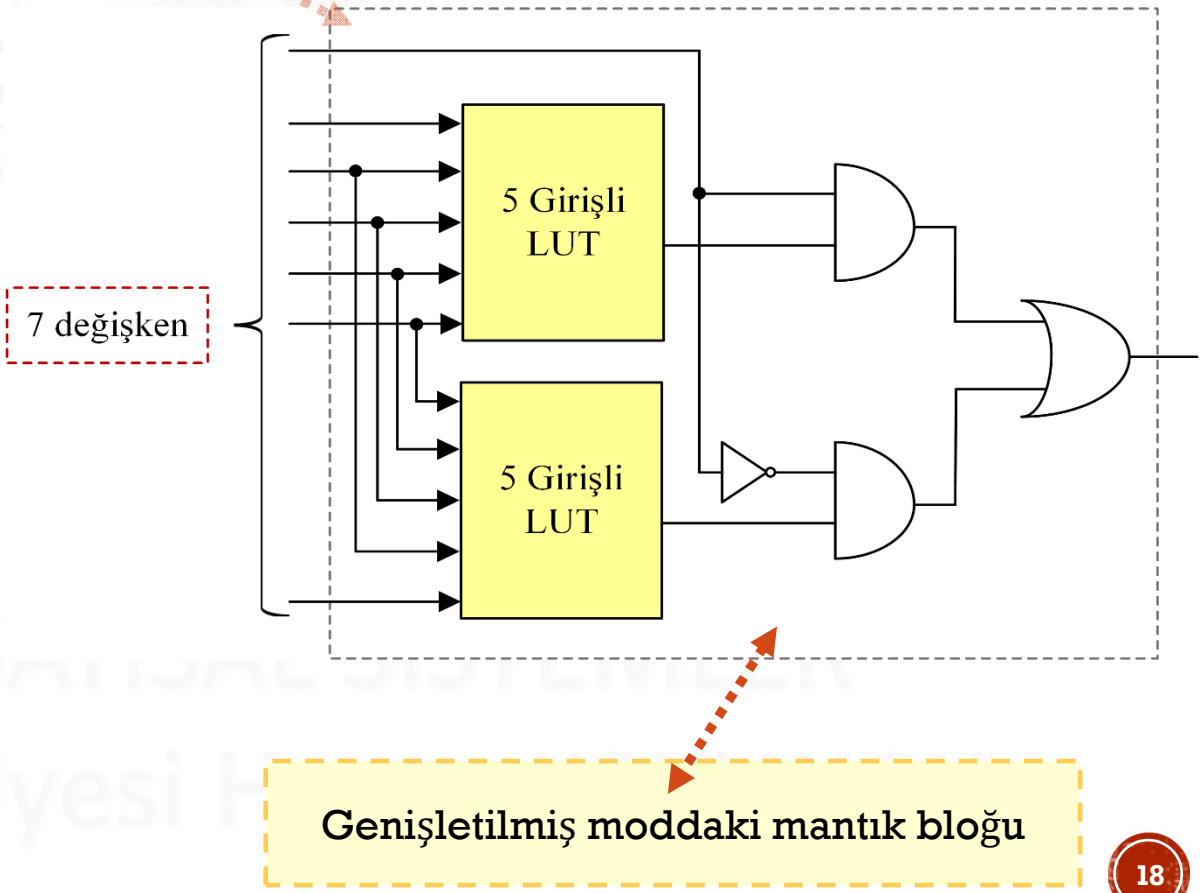


2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Genişletilmiş LUT modu, giriş paylaşımı yapılan 5+5 iki LUT ve bir harici giriş kullanarak, 7 farklı giriş değişkeninin işletildiği özelleşmiş bir seçenektir.

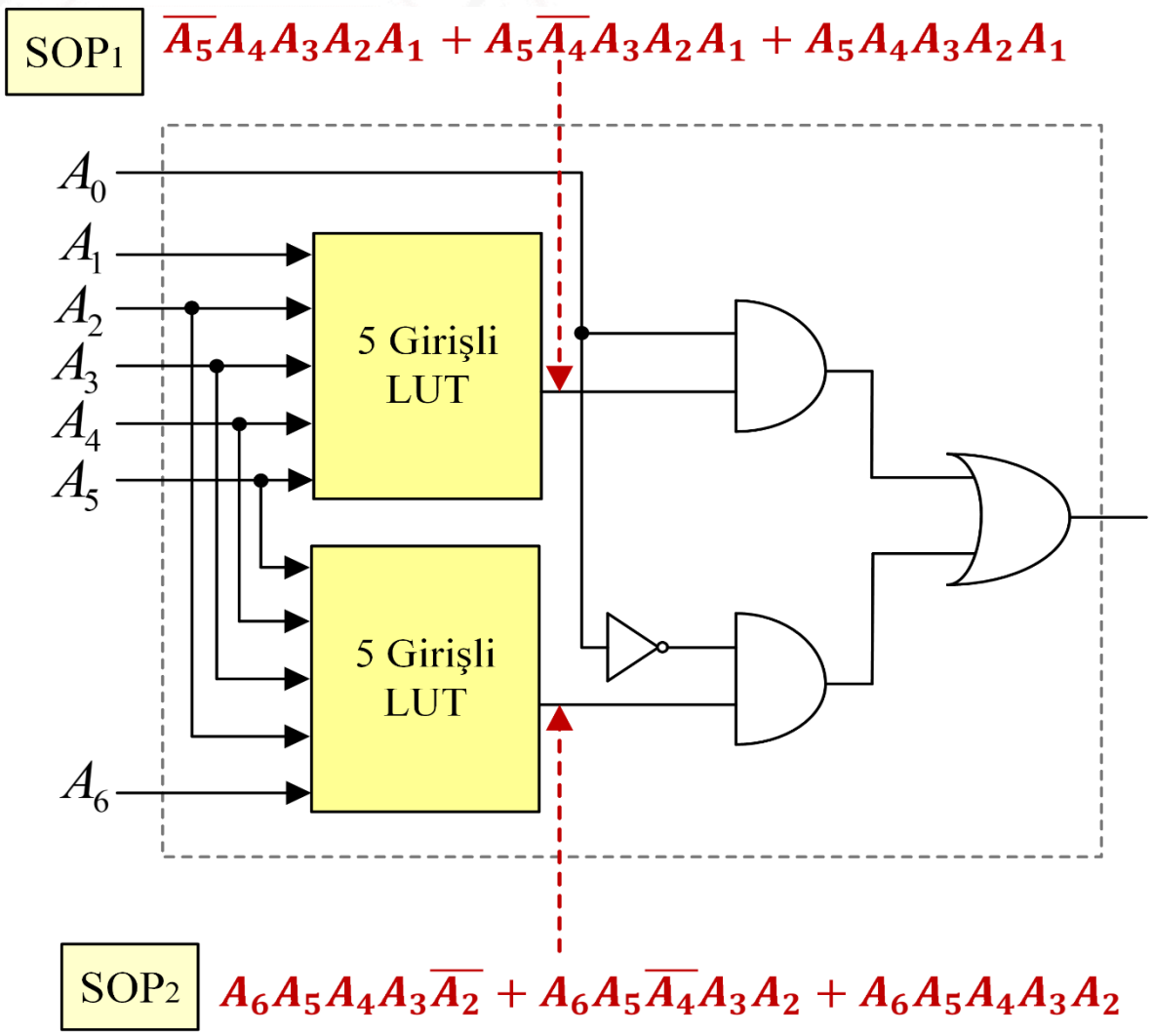
- Harici giriş normal ve tümleyen olarak herbir LUT çıkışındaki terimler ile çarpılmaktadır.

- Herbir AND kapısı çıkışında normal modda olduğu gibi en fazla 6 değişkenli terim içeren fonksiyonlar yer alır.



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

Örnek 2.2: Blok çıkışı = $\overline{A_5}A_4A_3A_2A_1A_0 + A_5\overline{A_4}A_3A_2A_1A_0 + A_5A_4A_3A_2\overline{A_1}A_0 + A_6A_5A_4A_3\overline{A_2}\,\overline{A_0} + A_6A_5\overline{A_4}A_3A_2\overline{A_0} + A_6A_5A_4A_3A_2\overline{A_0}$ olarak elde edilmek istendiği durumda; ***genişletilmiş mod mantık bloğu için*** herbir LUT çıkışındaki SOP fonksiyonlarını belirtiniz.



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- FPGA yapılarında farklı programlama teknolojileri kullanılabilmektedir. Bunlar;
  - SRAM (Static RAM) tabanlı
  - SRAM / Flash tabanlı
  - Anti-fuse tabanlı
  - EPROM / EEPROM tabanlı
  - Flash tabanlı olarak sıralanabilir.

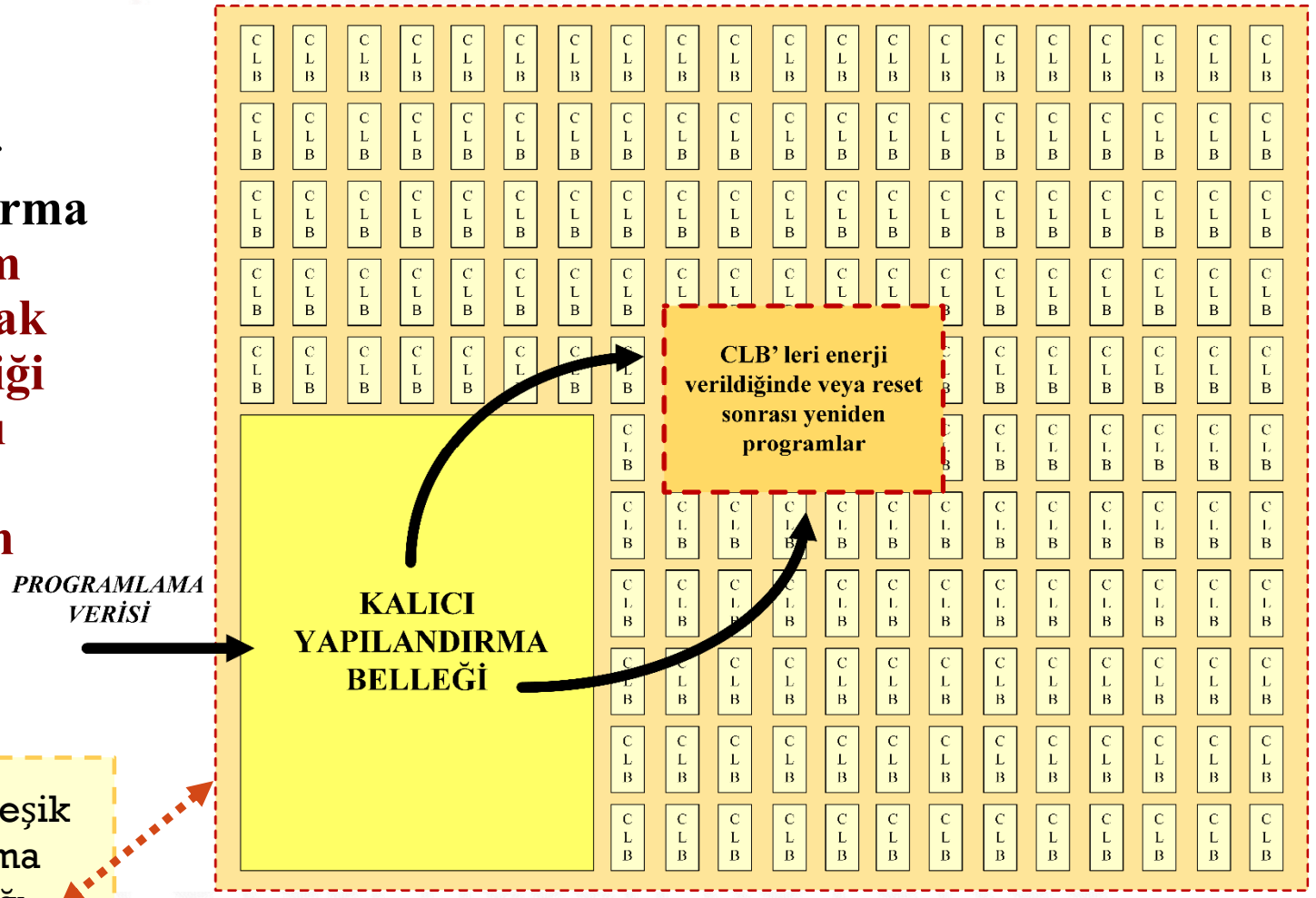
- ***En sık kullanılanı SRAM***, yani Durağan Rasgele Erişimli Bellek (SRAM → Static Random Access Memory) ***teknolojisi tabanlı FPGA yapılarıdır.***

- Bu yapılarda, CLB bloklarına yazılan program verisi enerji kesildiği durumda kaybolur.

2. FPGA İÇYAPISI VE ÖZELLİKLERİ

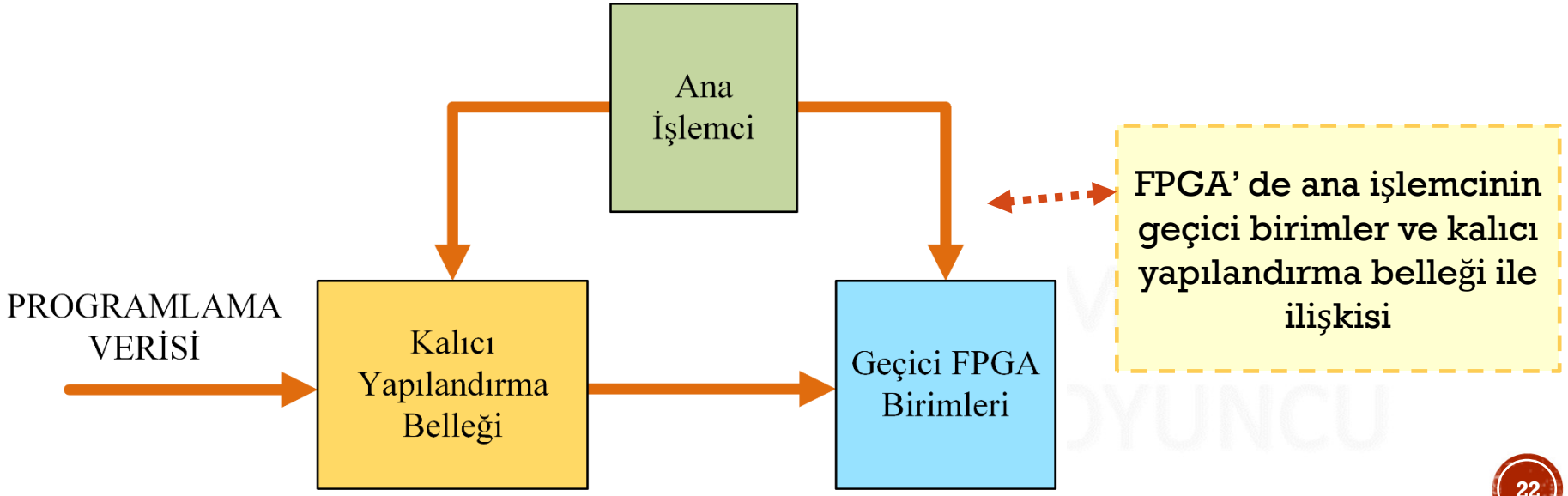
- SRAM tabanlı FPGA türlerinde iki farklı program verisi kaydetme seçeneği vardır:

- Yonga üzerine yerleştirilmiş bir **kalıcı yapılandırma belleği**, **program verisini saklamak** ve **enerji verildiği durumda cihazı yeniden düzenlemek için** kullanılır.



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- SRAM tabanlı FPGA türlerinde iki farklı program verisi kaydetme seçeneği vardır:
  - Yonga üzerine yerleştirilmiş bir kalıcı yapılandırma belleği, program verisini saklamak ve enerji verildiği durumda cihazı yeniden düzenlemek için kullanılır.
  - **Harici bellek birimi ile veri transferinin kontrolünü sağlayan ana işlemci kullanılır.**



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- FPGA yapıları, son kullanıcı tarafından her türlü mantıksal tasarımın programlanabileceği boş bir sayfa olarak görülebilir.

- Bazı FPGA türleri ***sert çekirdek (hard-core)*** tasarımları içerir. Bu tasarımlar ***FPGA içindeki lojik bir bölümdür*** ve ***üretici tarafından belirli bir fonksiyonun sağlanması için eklenir***.

- ***Sert çekirdek tasarım yeniden programlanamaz.***

Örneğin; bir müşteri sistem tasarımının bir bölümünde küçük bir mikroişlemciye ihtiyaç duyarsa, **bu birim kullanıcı tarafından programlanabilir** veya **üretici tarafından sert çekirdek tasarım olarak FPGA içine eklenebilir.**

- Eğer ***gömülü fonksiyon programlanabilir özelliklere sahipse***, bu tasarım ***yumuşak çekirdek (soft-core)*** olarak bilinir.

2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Sert çekirdek yaklaşımın temel avantajları;
  - kullanıcı tarafından sahada programlanacağı kapasiteden daha az kapasite ile bir fonksiyonun programlanması,
  - yonga üzerinde daha az yer harcanması,
  - daha kısa sürede temin olarak sıralanabilir.

- Sert çekirdek yaklaşımının temel dezavantajı ise ***teknik özelliklerin üretim esnasında sabit olması*** ve ***kullanıcının üretilecek yapıya müdahale edemeyecek şekilde (olduğu gibi) kullanması zorunluluğu***dur. **Çünkü bu yapılarda yeniden düzenleme opsiyonu yer almaz.**

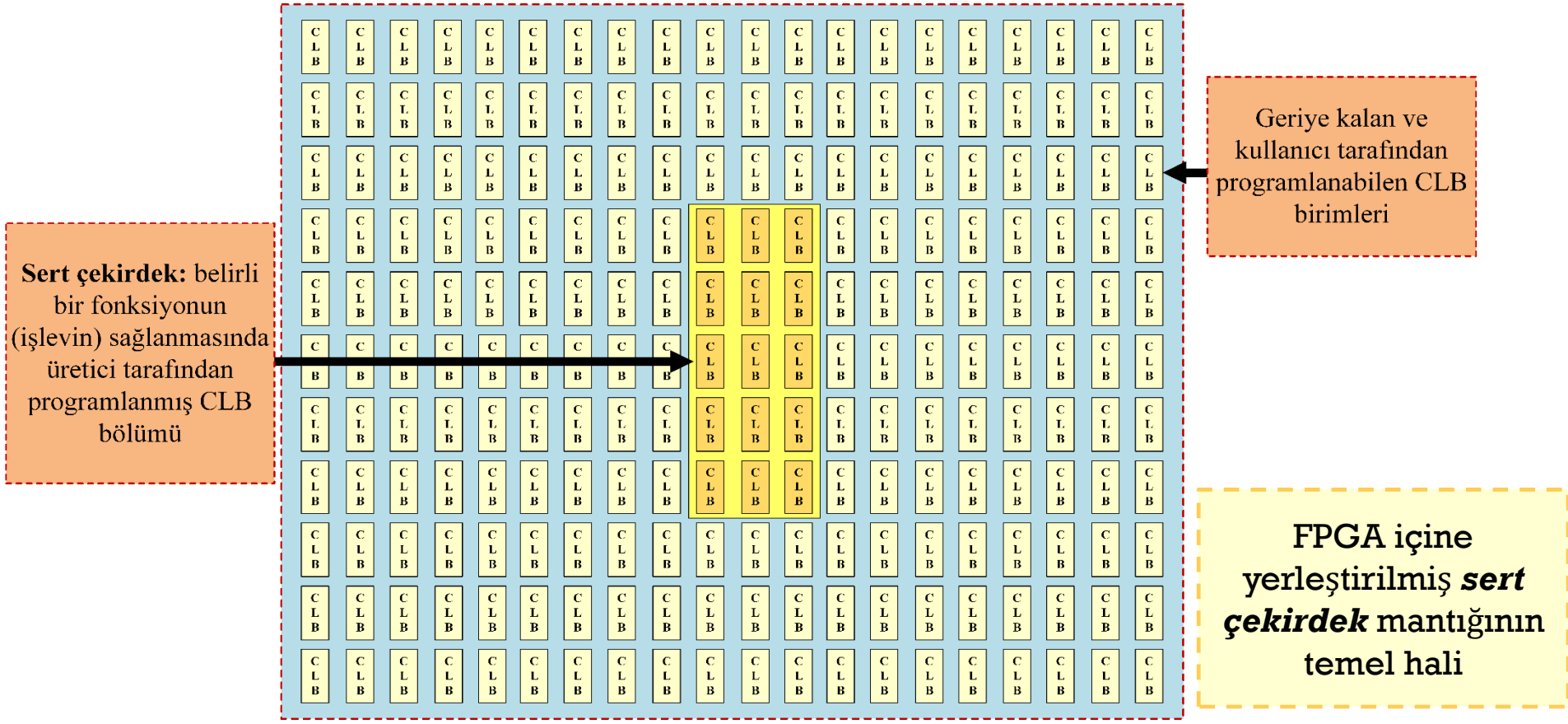
- **Sert çekirdek tasarımlar**, genelde sık kullanılan dijital sistemler (mikroişlemci, standart giriş / çıkış birimleri, dijital sinyal üreteçleri, v.b. yapılar) için mevcuttur.

- Birden fazla sert çekirdek fonksiyonu FPGA içinde yer alabilmektedir.

2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Şekilde kullanıcı tarafından programlanabilen mantık birimleri ile çevrelenmiş ***sert çekirdek yaklaşımı*** görülmektedir.

- Bu örnek, temel seviye bir gömülü sistemdir. Çünkü sert çekirdek fonksiyonu kullanıcı tarafından programlanabilen mantık birimleri içine yerleştirilmiştir.



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Sert çekirdek tasarımların geliştirilmesi ve fikri mülkiyeti ***FPGA üreticisine aittir.***

- Üretici tarafından geliştirilen tasarımlar ise **fikri mülkiyet** (**IP → Intellectual Property**) olarak bilinir.

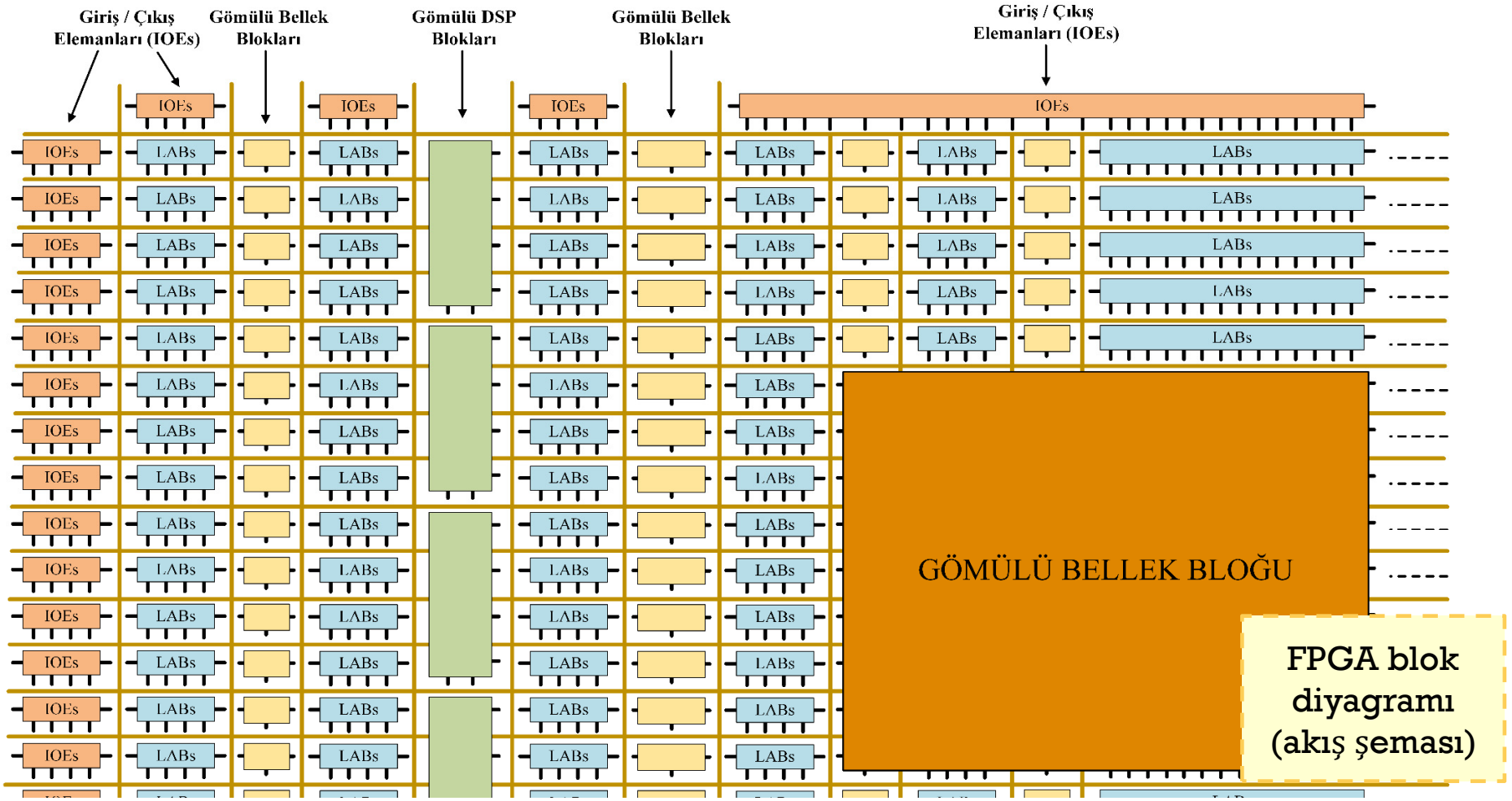
- ***FPGA üretici firma, IP dâhilindeki birimlerin tiplerini kendi web sitesinde paylaşır.***

- IP birimler yalnızca sert çekirdek tasarımları içermez, aynı zamanda bu tasarımların yumuşak çekirdek tasarımlar ile birleştirildiği durumlar da bulunur.

- ***Parametre seçimi ve ayarlaması esnek olabilen bir işlemci birimi bu duruma örnek olarak verilebilir.***

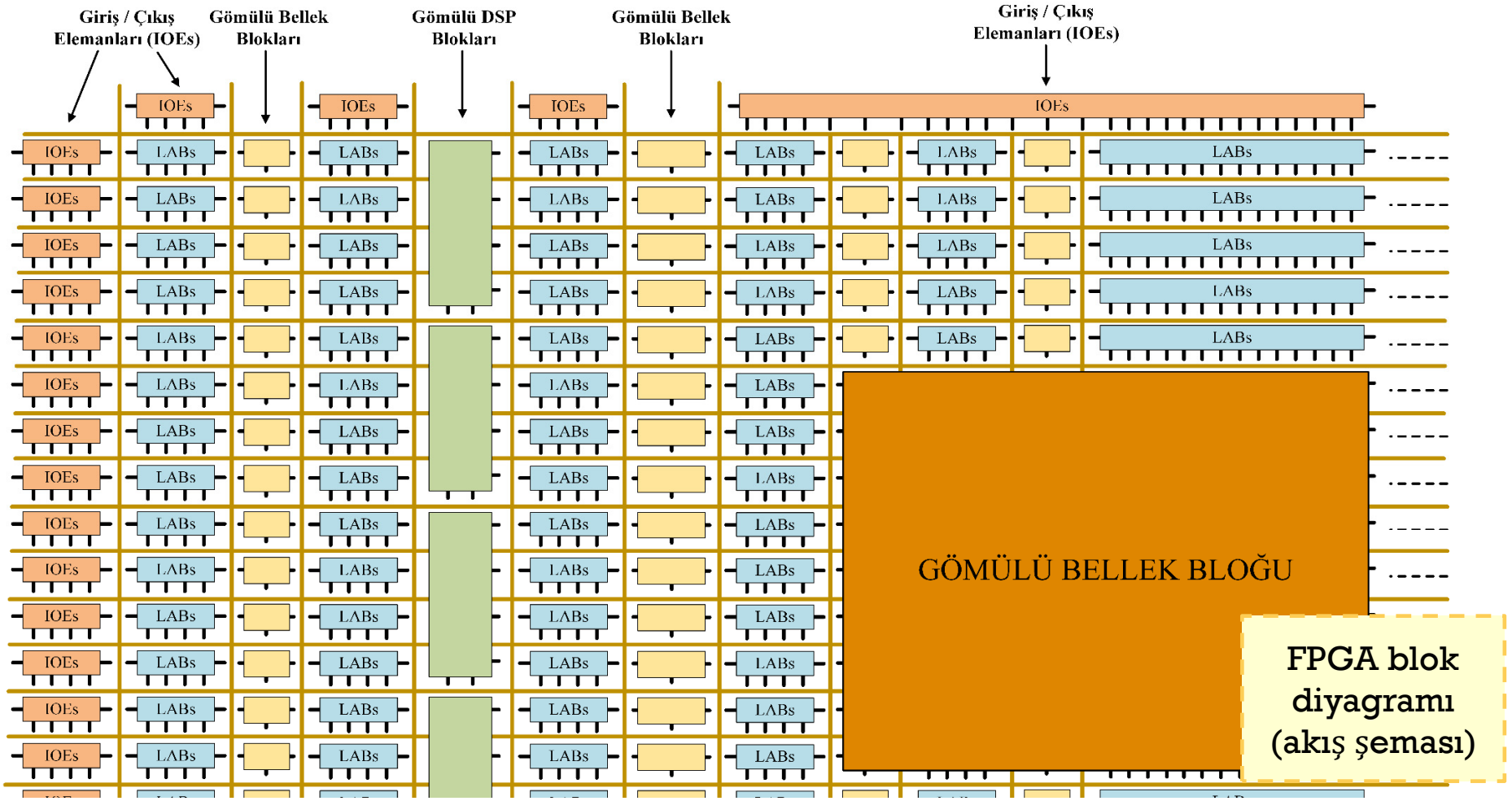
2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- FPGA, gömülü bellek fonksiyonlarının yanısıra dijital sinyal işleme (DSP → Digital Signal Processing) fonksiyonlarını barındırır.



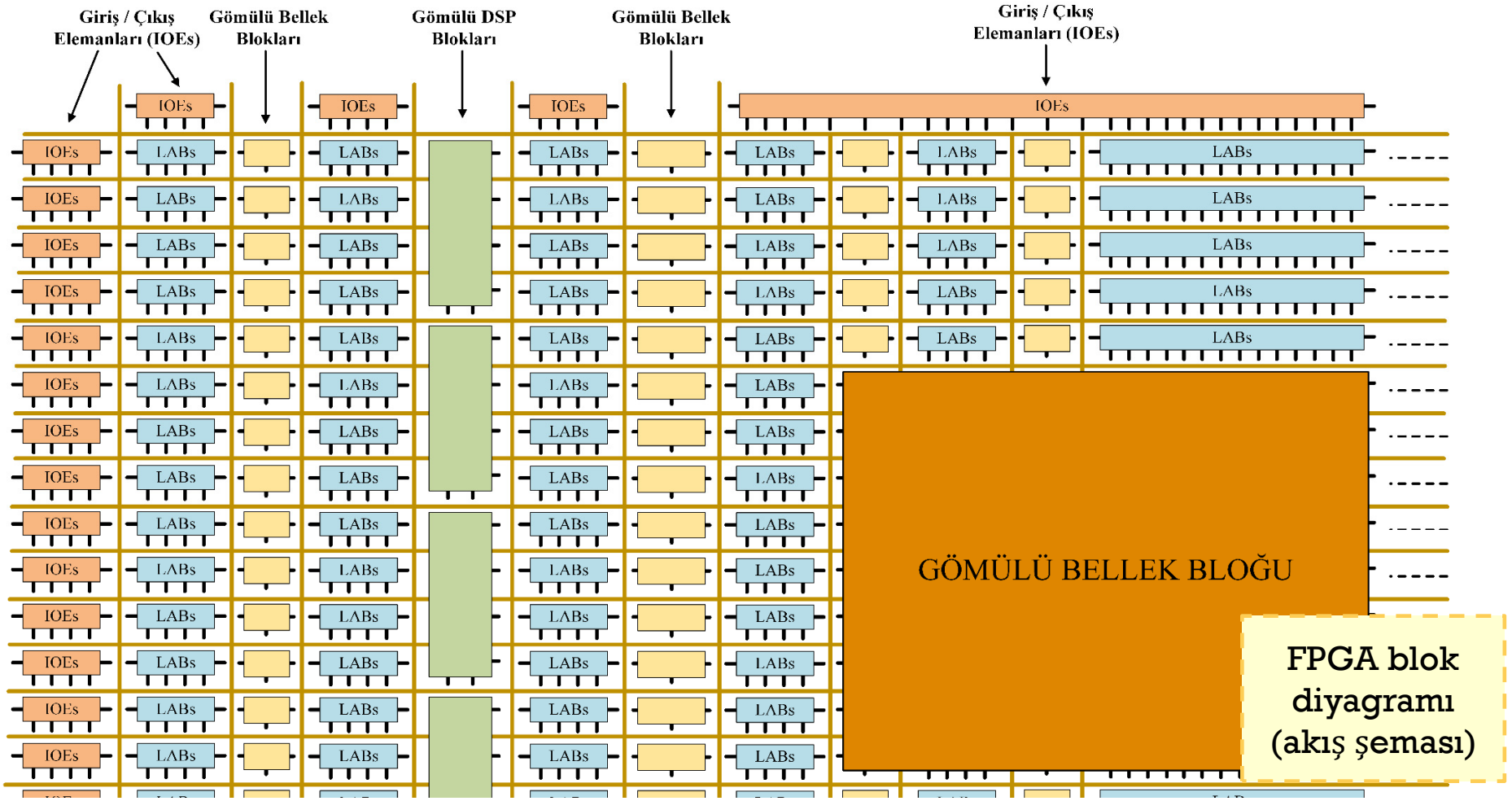
2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- DSP fonksiyonları; dijital filtreler, sinyal işleme ve kontrol sistemleri gibi birçok uygulamada kullanılmaktadır.



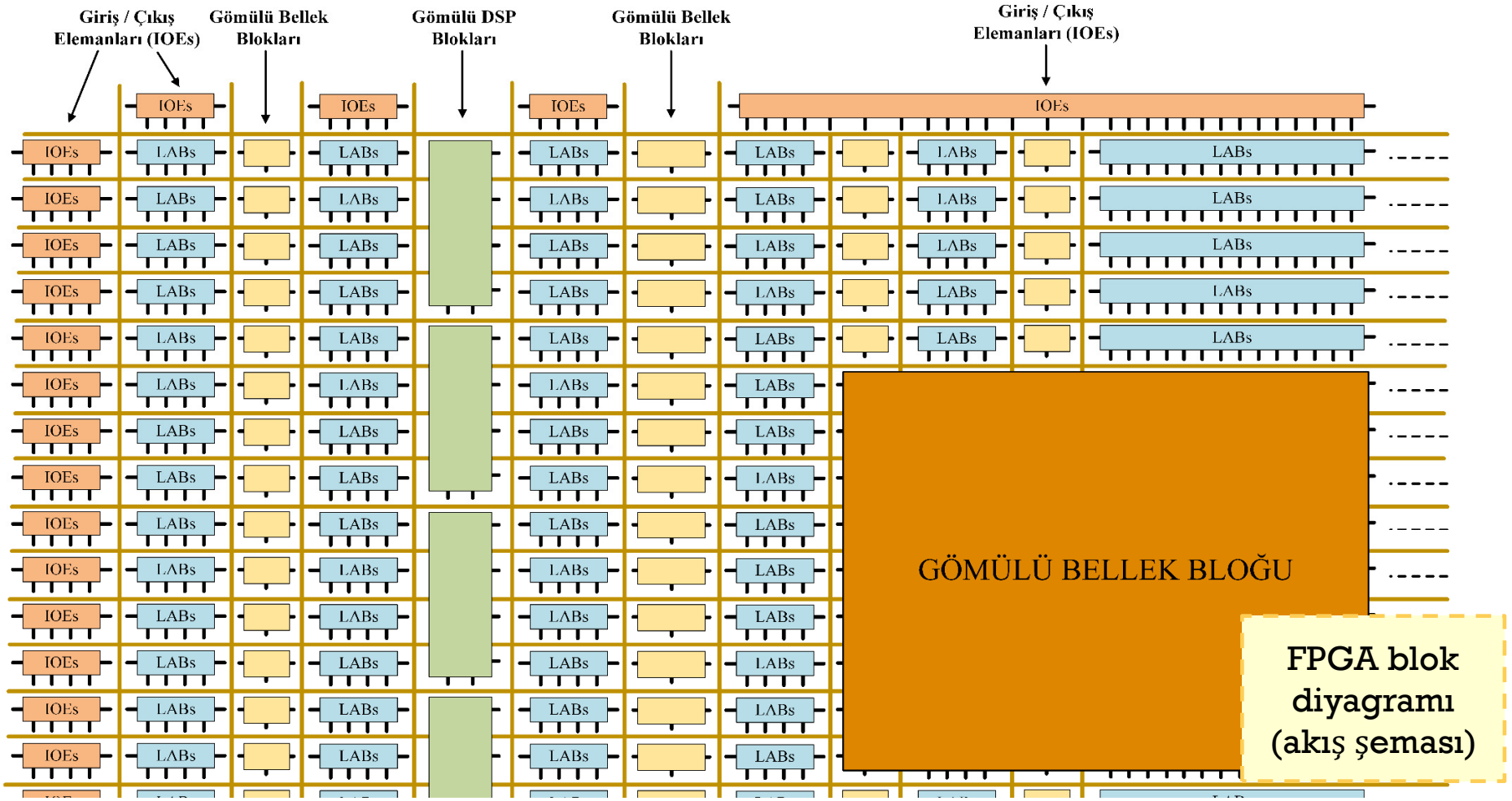
2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Gömülü bloklar FPGA arabağlantı matrisi içinde baştan sona bulunur ve giriş / çıkış birimleri (IOEs → Input / Output Elements) FPGA dış bölümünü sarmaktadır.



2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- LABs (Logic Array Blocks) birimleri ise CLB yapılarının bir araya getirilmesi ile elde edilir.



FPGA blok diyagramı (akış şeması)

2. FPGA İÇYAPISI VE ÖZELLİKLERİ

- Başlıca FPGA üreticileri;
  - Xilinx,
  - Altera,
  - Lattice,
  - Microsemi,
  - Quicklogic,
  - SiliconBlue firmalarıdır.

- Bu firmalardan pazarın büyük çoğunluğuna sahip olanlar ***Xilinx*** ve ***Altera*** firmaları olarak bilinir.

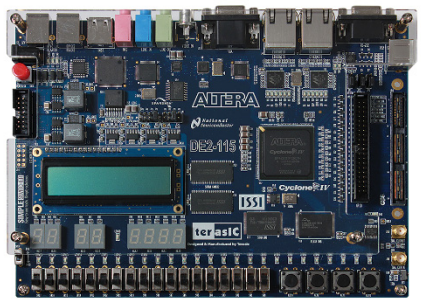
- Xilinx firması FPGA' yı ilk üreten ve şu anda dünyadaki en büyük FPGA üreticisi olan şirkettir.

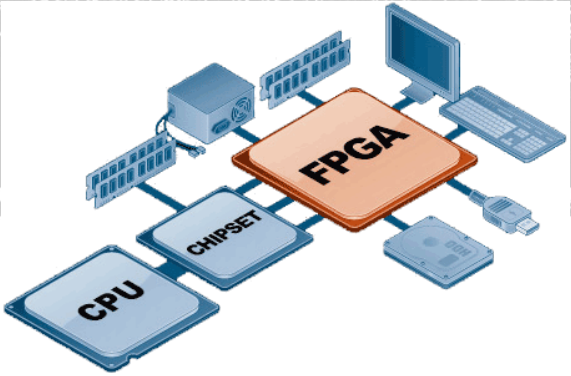
SONRAKİ DERS KONUSU

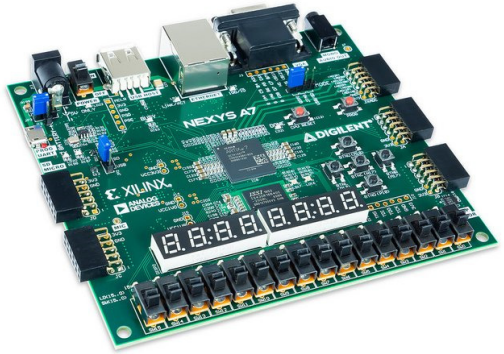
3- VHDL

İLERİ SAYISAL SİSTEMLER

3- VHDL' e Giriş







Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

3. VHDL' E GİRİŞ

- VHDL, ***Yüksek Hızlı Tümleşik Devre Donanım Tanımlama Dili*** (Very High Speed Integrated Circuit Hardware Description Language) olarak bilinir.

- VHDL, donanım parçalarını modellemek için kullanılır.

- ***VHDL***'in yanısıra kullanılan bir diğer donanım tanımlama dili de ***Verilog***'dur.

VHLD	Verilog
Ada diline benzer	***C*** diline benzer
Tasarımcı kendi veri yapılarını tanımlayabilir	Veri yapıları önceden belirlidir ve yeni veri yapısı tanımlanamaz
Tasarlanan prosedür ve fonksiyonlar kütüphaneye dönüştürülerek kullanılabilir	Paket ve kütüphane yapıları bulunmaz
I/O bağlantıları ve devrenin işleyişi ayrı bloklar olarak yazılır	I/O bağlantıları ve devrenin işleyişi aynı blok içinde tanımlanmalıdır

3. VHDL' E GİRİŞ

- VHDL programlama dilinde kütüphane ve paket yapılarının oluşturulabilmesi, ***karmaşık sistemlerin modellenmesini*** ve ***yönetimini kolaylaştırır***.

- Bu nedenle ***karmaşık sistem tasarımlarında daha çok VHDL dili kullanılır.***

- Verilog dili öğrenilmesi daha kolay olduğundan genelde başlangıç seviyesinde programcılar tarafından tercih edilir.

- VHDL dili ABD Savunma Bakanlığı tarafından geliştirildiğinden ***daha bürokratik bir yapıdadır*** ve ***kütüphane geliştirme imkânı sunduğundan ileri seviye tasarımcılar tarafından tercih edilir.***

3.1 VHDL Tasarımda Metodolojiler

- VHDL tasarım metodolojileri 4 alt başlıkta incelenir. Bunlar; ***soyutlama***, ***modülerlik***, ***hiyerarşik tasarım*** ve ***modelleme teknikleri*** şeklinde sıralanır.

3.1.1 Soyutlama

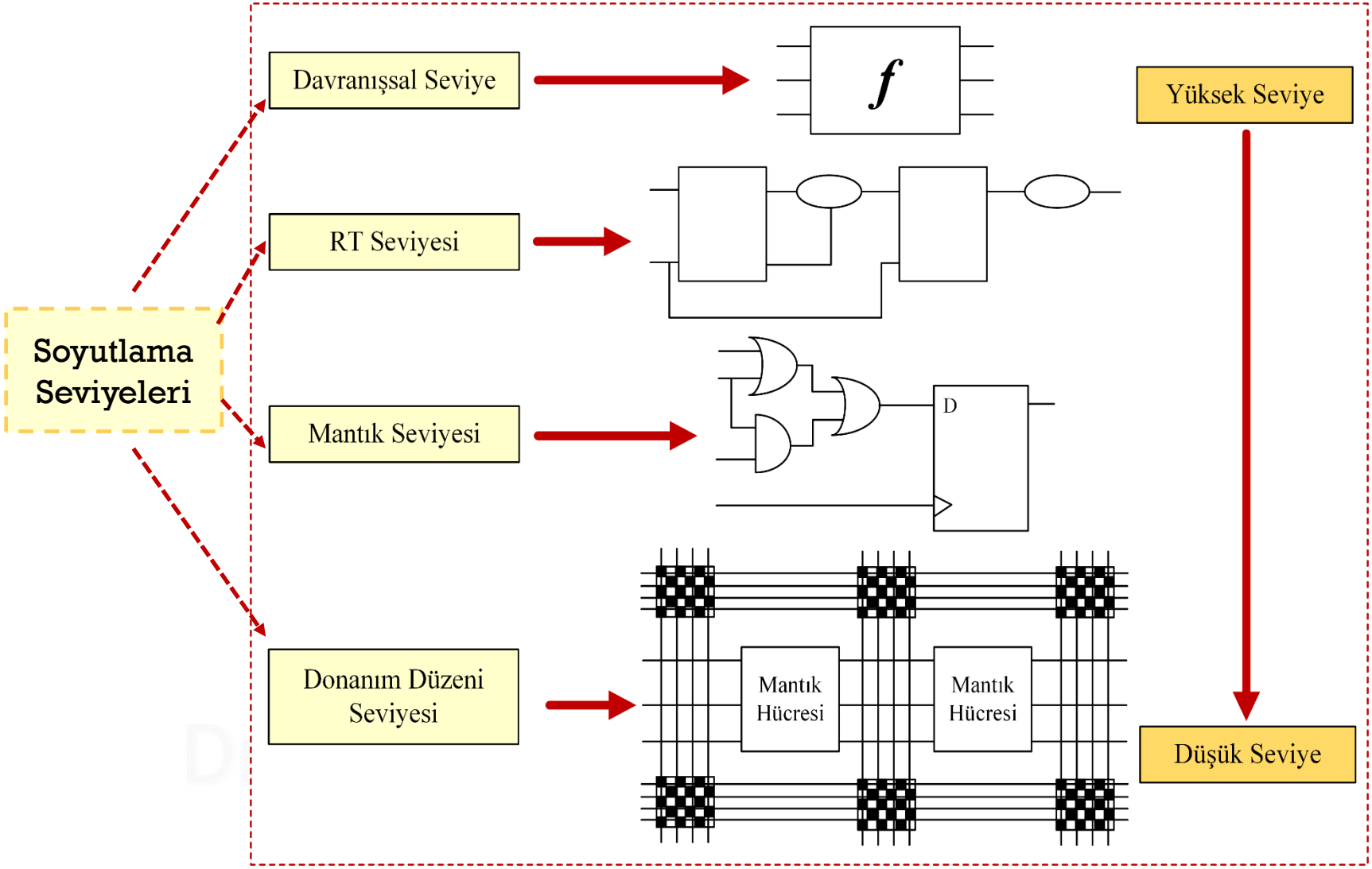
- Günümüzde çok karmaşık olan ve fazlaca eleman içeren yongaların doğrudan işlenmesi zorlaşmaktadır.

- Bu karmaşıklığı daha kolay yönetebilmek adına ***çeşitli soyutlama seviyeleri*** ve ***bu seviyelere dair ana hatlar*** belirlenmiştir.

- Soyutlama, ***tasarımdaki gereksizlikleri atmak*** ve ***veriyi yönetilebilir seviyede tutmak*** için kullanılır.

3.1.1 Soyutlama

- Soyutlama ile komplike sistemler daha basit bir modele indigenebilir.

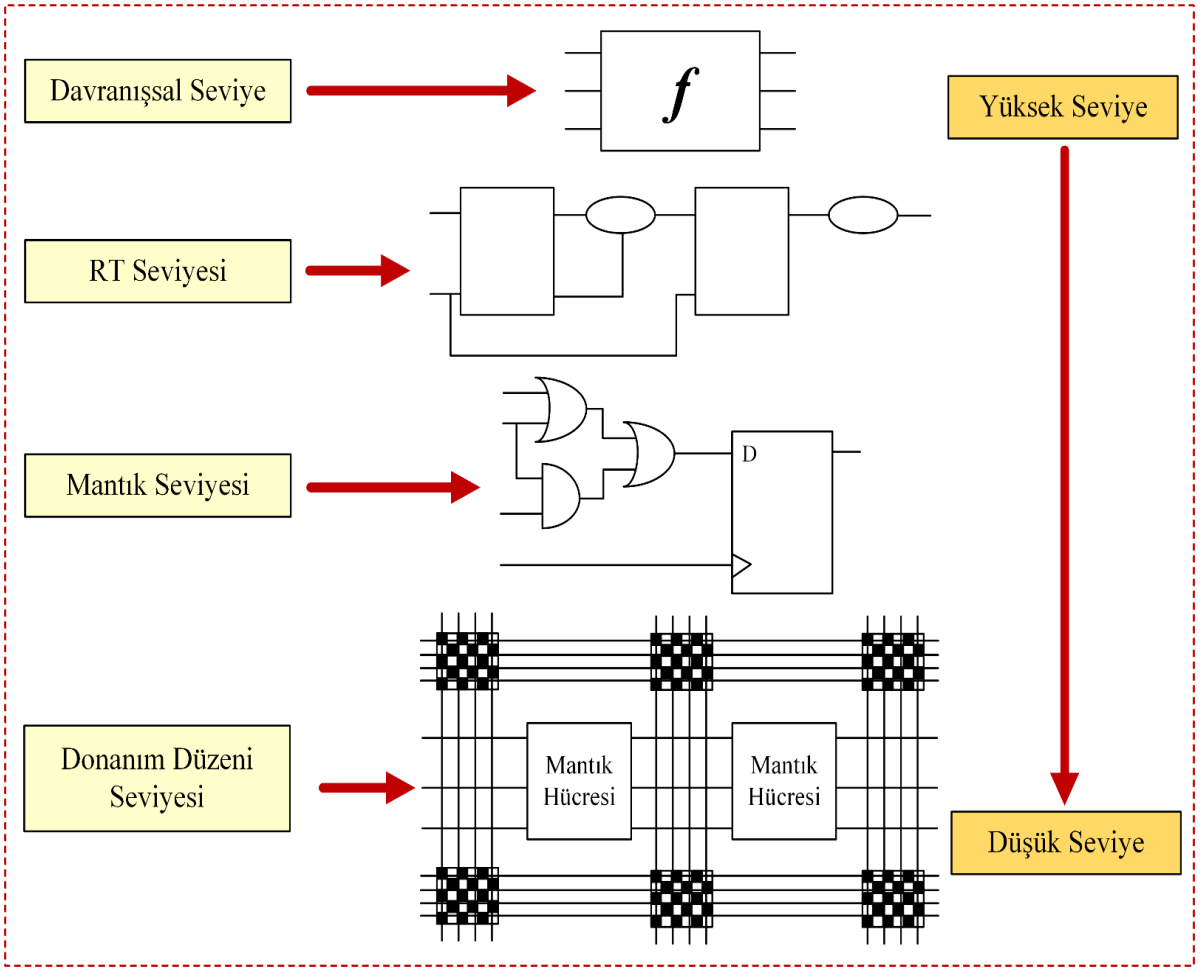


3.1.1 Soyutlama

- Yüksek seviye soyutlama ***çok önemli veriler üzerine yoğunlaşırken***, düşük seviye soyutlama ***ayrıntı içerir*** ve önceki aşamalarda ***göz ardı edilen bilgiler gereklidir.***

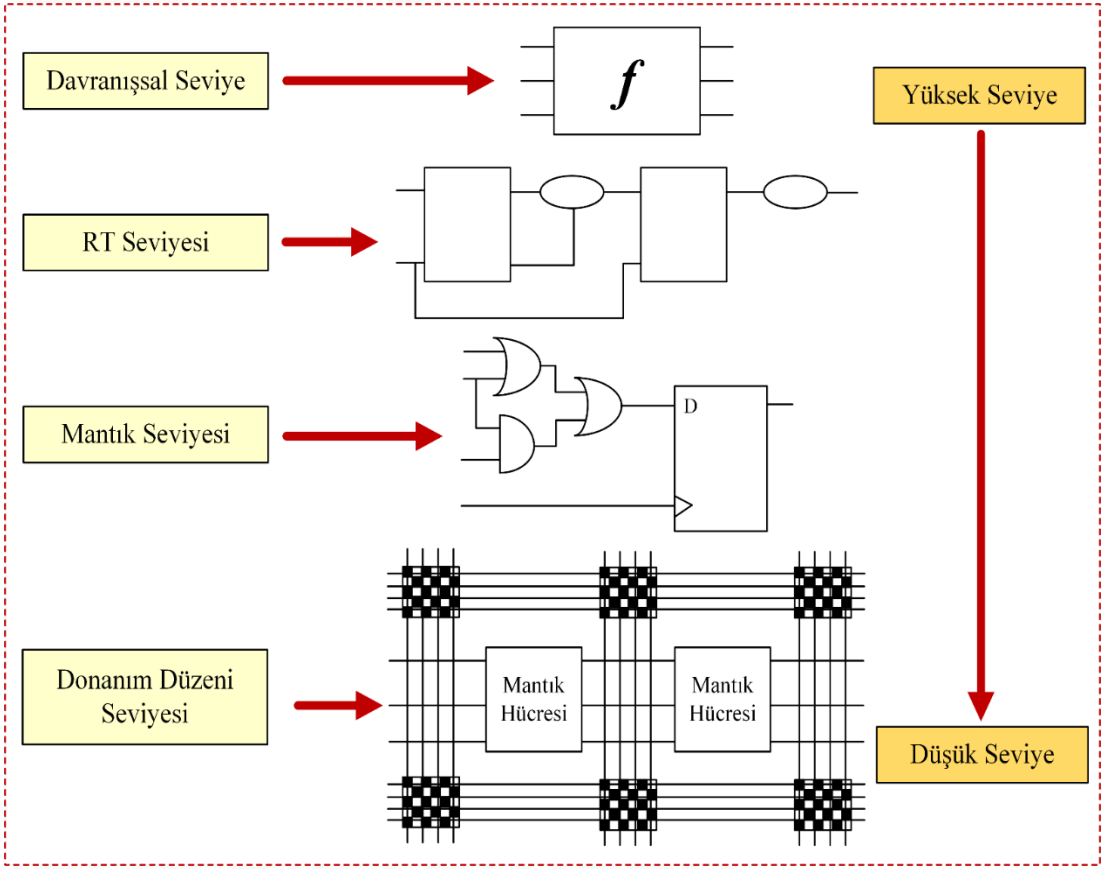
- Yüksek seviye soyutlama için gerekli tasarım, düşük seviye soyutlamadaki tasarımdan ***daha hızlıdır.***

- Düşük seviye soyutlamada tasarlanan yapılar ***daha karmaşıktır***, ancak bu seviye temelli oluşturulan devreler ***tasarlanması gereken devreye daha yakındır*** ve ***daha doğru sonuç üretir.***



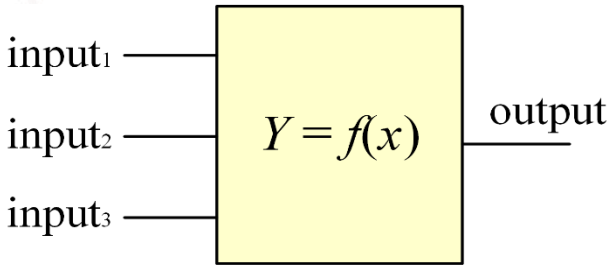
3.1.1 Soyutlama

- Ayrıntılı sistem tasarlanırken ilk olarak;
  - **yüksek seviye soyutlamadan giriş yapılır,**
  - **sistemdeki vazgeçilemezler belirlenir,**
  - **sistem daha iyi anlaşıldıkça detaylar artırılarak düşük seviye soyutlamaya gidilir.**



3.1.1 Soyutlama

Davranışsal Seviye: Modelin fonksiyonel tanımı yapılarak ana hatları belirtilir.



output=input1+(input2*input3)

Davranışsal Seviye Örneği

- Saat (Clock) sinyalinin kullanılmadığı durumda sinyal geçişleri asenkron olacaktır ve bu geçişler anahtarlama zamanına bağlıdır.

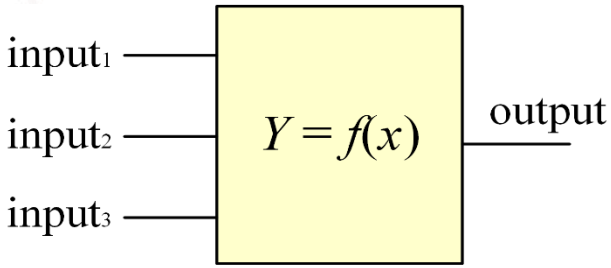
- Davranışsal modelleme, devre davranışını tanımlamada ***en basit yoldur***.

- Davranışsal seviye içinde tanımlanan tasarımlar ***simülasyon amaçlı işletilir***.

- Bu tasarımların içinden ***yalnızca çok basit olanları sentezlenebilir.***

3.1.1 Soyutlama

Davranışsal Seviye: Davranışsal seviye ***çok büyük tasarımlar için mümkün görülmez.***



output=input1+(input2*input3)

Davranışsal Seviye Örneği

- Ancak, veri yolu (Bus) sistemleri veya karmaşık algoritmalar ***sentezlenebilirlik kaygısı olmadan tanımlanabilir.***

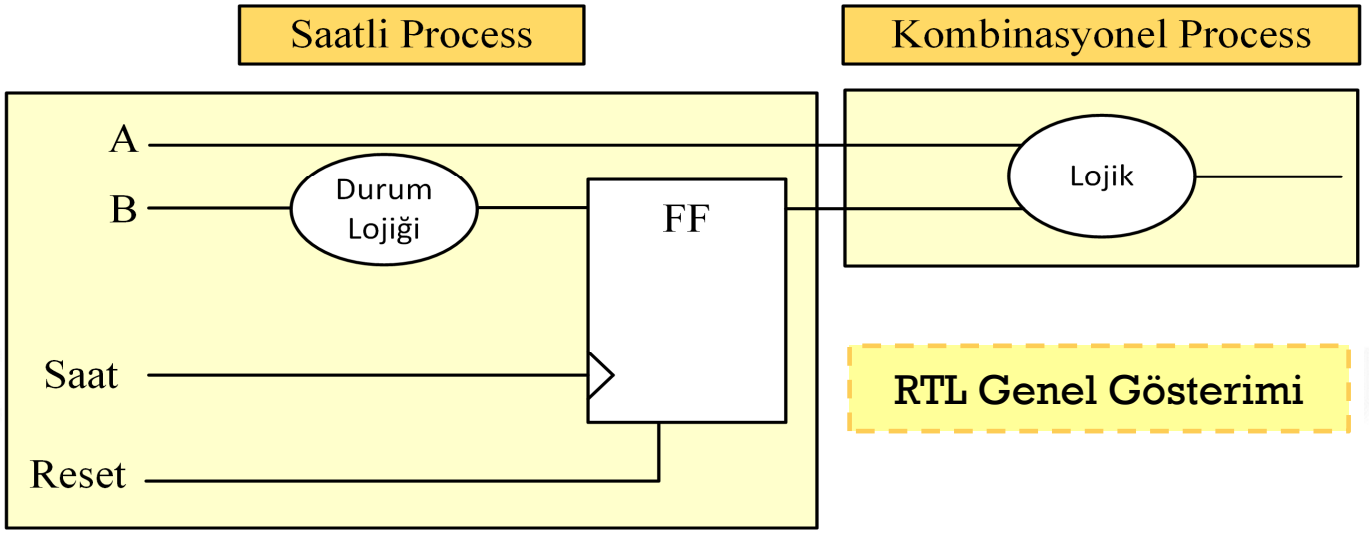
- Test bench' de **hafıza birimleri içeren bir modelin simülasyonu için giriş (uyarıcı) tanımlamalarının belirlenmesi** davranışsal seviye tasarıma örnektir.

3.1.1 Soyutlama

RT Seviyesi (RTL): RTL (Register Transfer Level), **saat işaretine göre veri aktarımı yapan ardışıl devreler** ile **bu devreleri destekleyen kombinasyonel lojik yapıların tanımlandığı** tasarım düzeyidir.

- **Toplayıcı ve karşılaştırıcı gibi fonksiyonel birimleri**, **register gibi hafıza birimlerini** ve **çoklayıcı gibi veri seçici elemanları** içerir.

- RTL seviyesinde **verinin nasıl işleneceği** ve **iletimin nasıl sağlanacağı** belirlenir.

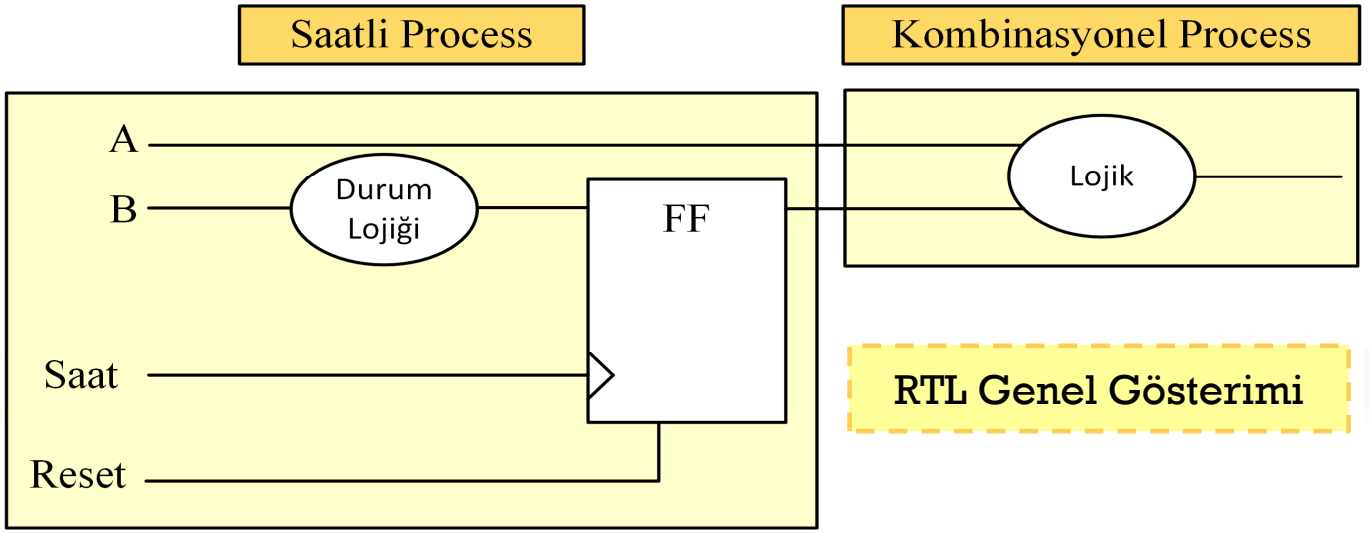


3.1.1 Soyutlama

RT Seviyesi (RTL): RTL seviyesinin en önemli noktası tasarlanan yapı içinde ***ortak clock sinyaline göre işlem yapılmasıdır.***

- Bu noktada veri sinyalleri yalnızca **clock sinyalinin tetik kenarında işleme tabi tutulur.**

- Bu nedenle bu seviyede, **asenkron sistemlerde ardışıl bağlanan hafıza birimlerinde olduğu gibi büyük sinyal gecikmeleri meydana gelmez**, ayrıca **anlık atlamalar (glitch) gerçekleşmez.**

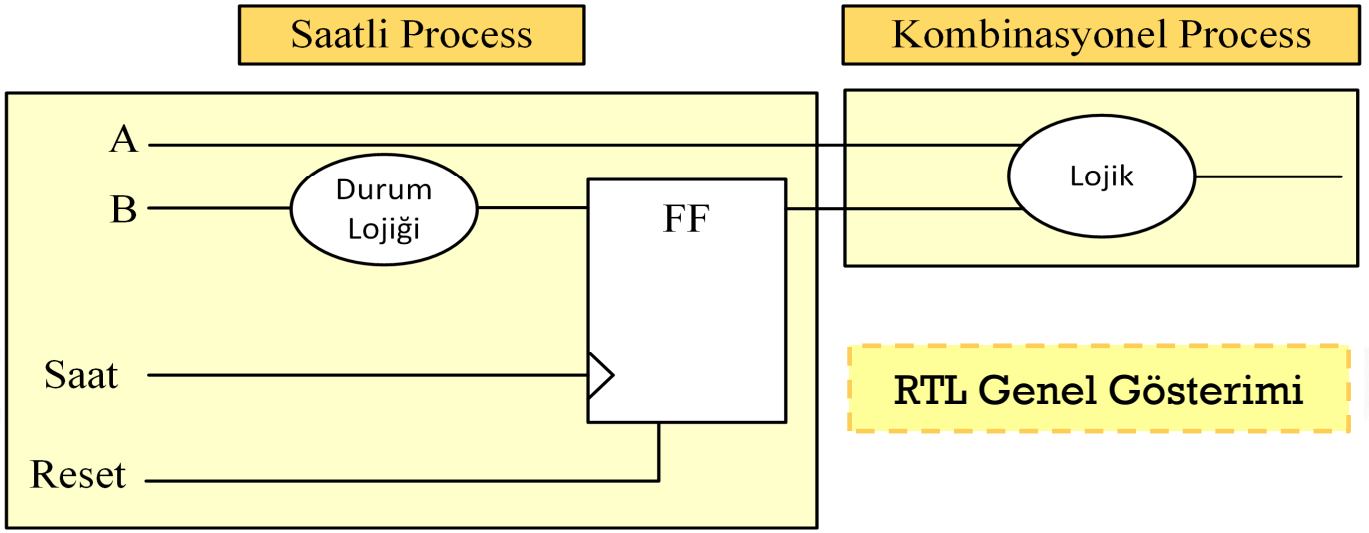


3.1.1 Soyutlama

RT Seviyesi (RTL): RT seviyesindeki bir simülasyon için **tek bir clock periyodu işletildiğinde**, **bütün sinyallerin denge değerlerine ulaşıp ulaşmadığı hakkında kesin bir şey söylenemez.**

- Yani bu seviyede yapılan simülasyonlar, **sistemin gerçek zamanlama davranışı ile ilgili bilgi vermez.**

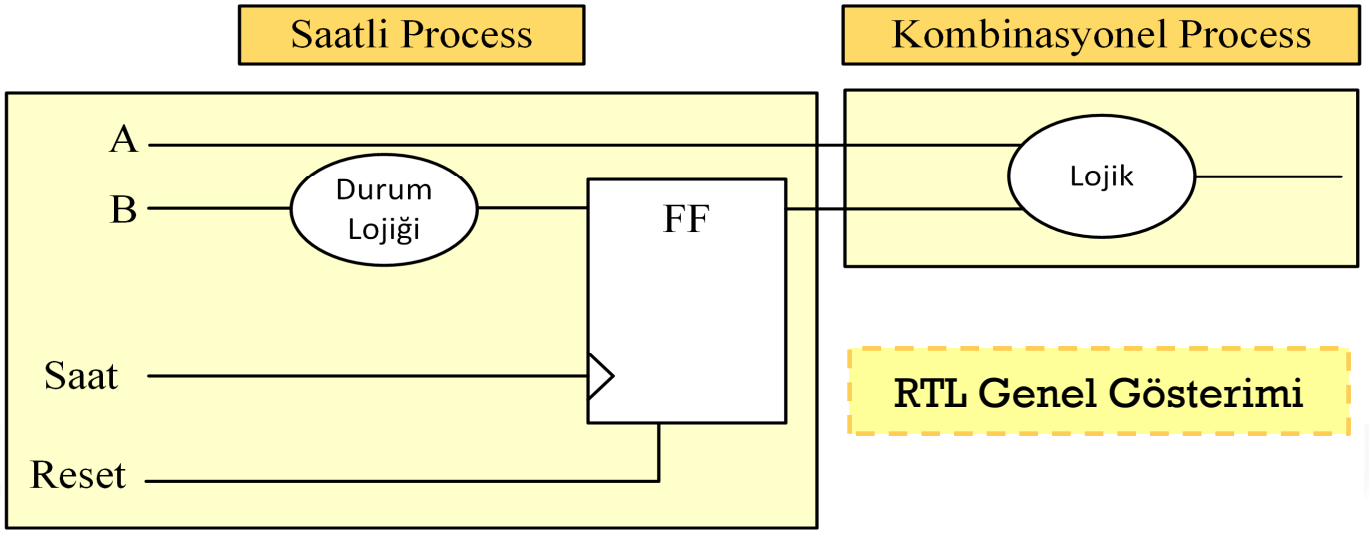
- Diğer bir deyişle, **yalnızca sisteme dair senkron davranış modellenir.**



3.1.1 Soyutlama

RT Seviyesi (RTL): VHDL'de fonksiyonel davranışlar ***process*** ile modellenir.

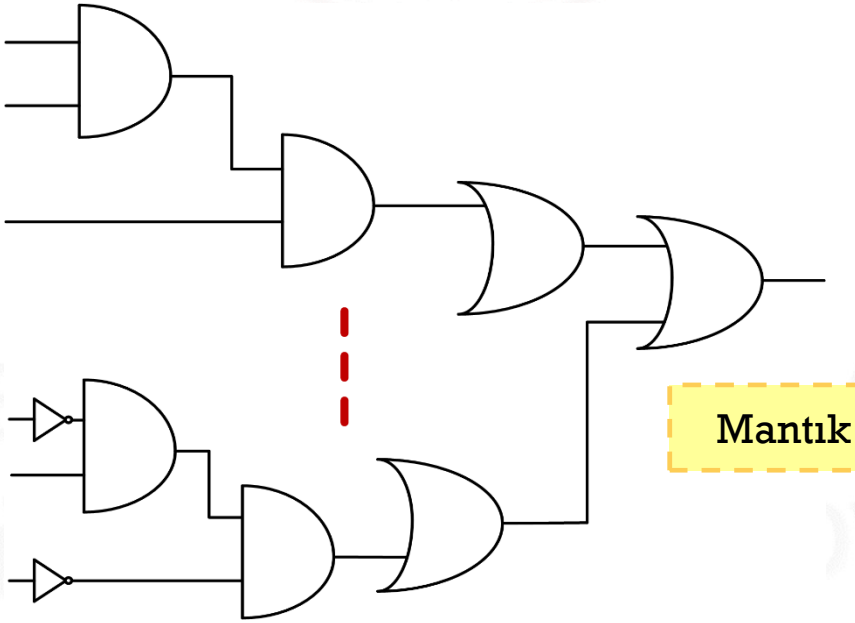
- RT seviye tasarımda **kombinasyonel** ve **saatli (clock içeren)** olmak üzere iki tip process mevcuttur.



3.1.1 Soyutlama

Mantık Seviyesi: Mantık seviyesinde amaçlanan tasarım, **lojik kapılar ve hazıfa (depolama) elemanlarından oluşan bir ağ yapısı şeklindedir.**

- Herhangi bir tasarım mantık seviyesinde nitelendiğinde, **tasarımda kullanılan kapı gecikmeleri simülasyona uygulanabilir.**



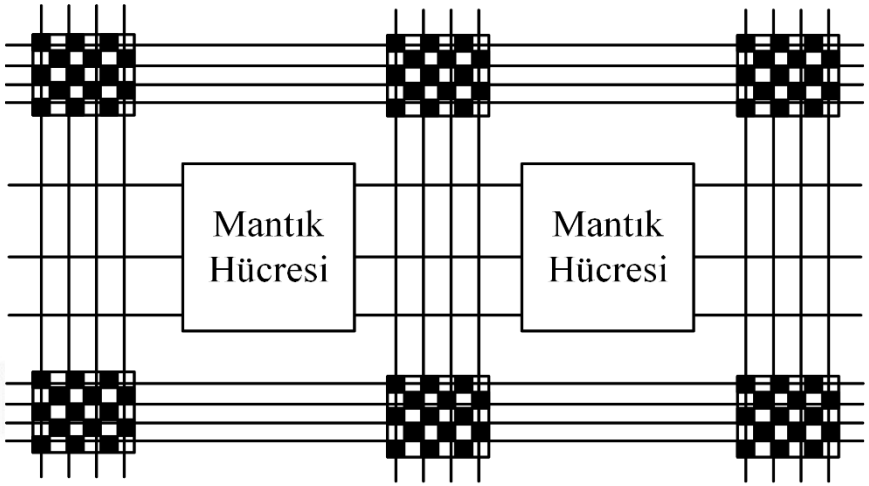
Mantık Seviyesi Örneği

3.1.1 Soyutlama

Donanım Düzeni (Layout) Seviyesi: Soyutlamanın en alt kısmı donanım düzeni seviyesidir.

- Bu seviyede gerekli tasarım **farklı hücreler üzerine aktarılır** ve **hücreler arasındaki bağlantılar yapılır.**

- **Donanım düzeni tamamlandıktan sonra devre üretime hazır hale gelir.**

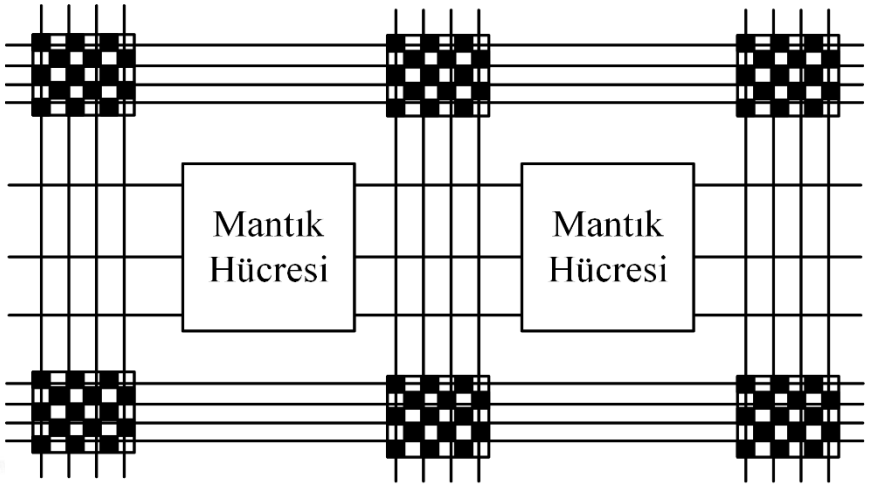


Layout Seviyesi Örneği

3.1.1 Soyutlama

Donanım Düzeni (Layout) Seviyesi: Donanım düzeninin tasarlanması sonrası **bağlantı uzunlukları**, yani **sinyallerin gecikmeleri belirlenir.**

- ***Mantık birimlerindeki gecikmelerin hesaba katılmasından sonra bütün devrenin zamanlama davranışı bulunur.***



Layout Seviyesi Örneği

3.1.2 Modülerlik

- Modülerlik, büyük bir tasarımın alt parçalara (birimlere) ayrılarak ele alınması ve programlanması anlamına gelir.

- Bu tip tasarıma ***modüler tasarım*** denilir.

- VHDL kapsamında oluşturulan modüler tasarımlar, **hem kolay tasarım yapılmasını hem simülasyonun kolaylaştırılmasını hem de eşzamanlı farklı kullanıcıların çalışabilmesini sağlar.**

3.1.3 Hiyerarşik Tasarım

- Tasarlanacak olan büyük bir sistemin alt modüllerden oluşmasını sağlar (modülerliğin tümleşmiş halidir).

3.1.3 Hiyerarşik Tasarım

- Tasarım hiyerarşisinin her ayrı bölümü **farklı bir soyutlama seviyesinde modül içerebilir.**

- **Böylelikle tasarım daha kolay yönetilebilir hale gelir.**

- Hiyerarşik tasarım, **hem yazılımcılar hem de donanımcılar tarafından kullanılır.**

- Hiyerarşik tasarım ve modülerlik; VHDL içindeki **kütüphane**, **bileşen** ve **paket** kavramları üzerinden sağlanır.

3.1.4 Modelleme Teknikleri

- VHLD tasarım kapsamında temel olarak üç tip modelleme tekniği mevcuttur:

1-) Veri akış (Dataflow),

2-) Davranışsal (Behavioral),

3-) Yapısal (Structural).

- **Veri akış (Dataflow) modeli,** sistemin girişine sunulan bir sinyalin çıkışa kadarki eş zamanlı akışını modelleyen tasarım tekniğidir.

- Bu model ***alt tasarım düzeyi olmaktadır*** ve devredeki AND, OR, XOR gibi ***yerleşik bileşenler arası giriş çıkış bağlantıları gösterilerek modelleme yapılmaktadır.***

3.1.4 Modelleme Teknikleri

- VHLD tasarım kapsamında temel olarak üç tip modelleme tekniği mevcuttur:
  - **1-) Veri akış (Dataflow),**
  - **2-) Davranışsal (Behavioral),**
  - **3-) Yapısal (Structural).**

- **Davranışsal modellemede** geliştirilecek olan modelin giriş ve çıkış ilişkisi (işlevi) **davranışsal olarak ele alınır.**

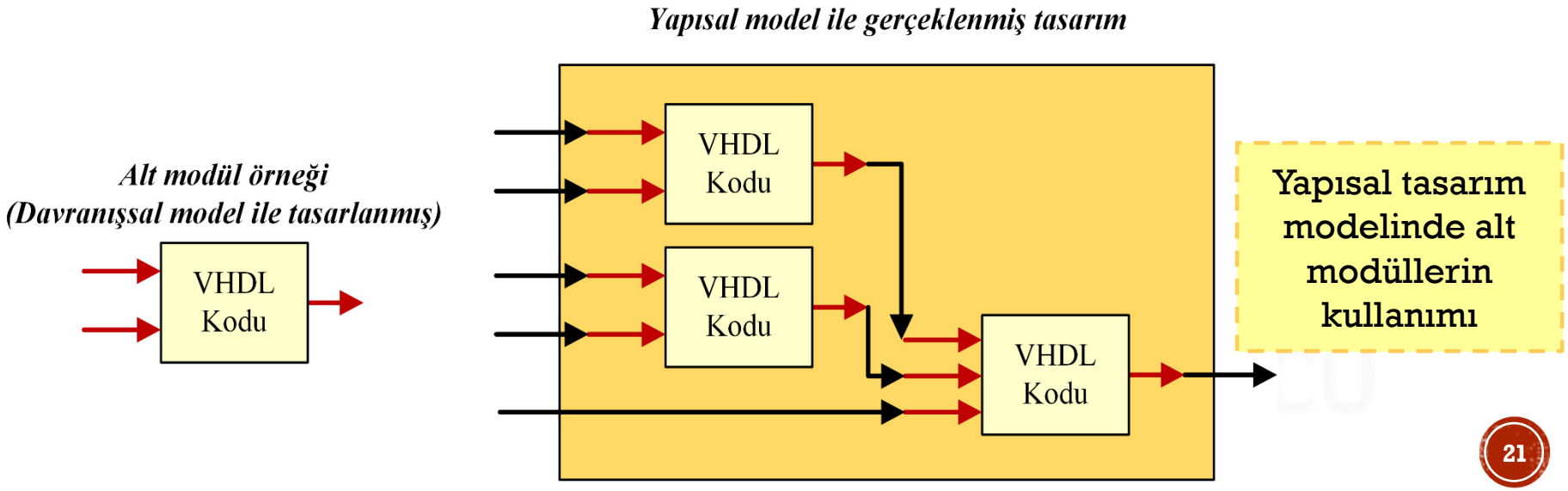
- Diğer bir deyişle, **sistem çıkışlarının girişlere verdiği tepki modellenir** ve **modele dair iç yapı (kapı vb. eleman bağlantıları) önemsenmez.**

- **Veri-akış modeline göre daha üst tasarım teşkil eder. Bu nedenle *for*, *case* ve *if* gibi komutlar bu modelde işletilir.**

3.1.4 Modelleme Teknikleri

- VHLD tasarım kapsamında temel olarak üç tip modelleme tekniği mevcuttur:
  - **1-) Veri akış (Dataflow),**
  - **2-) Davranışsal (Behavioral),**
  - **3-) Yapısal (Structural).**

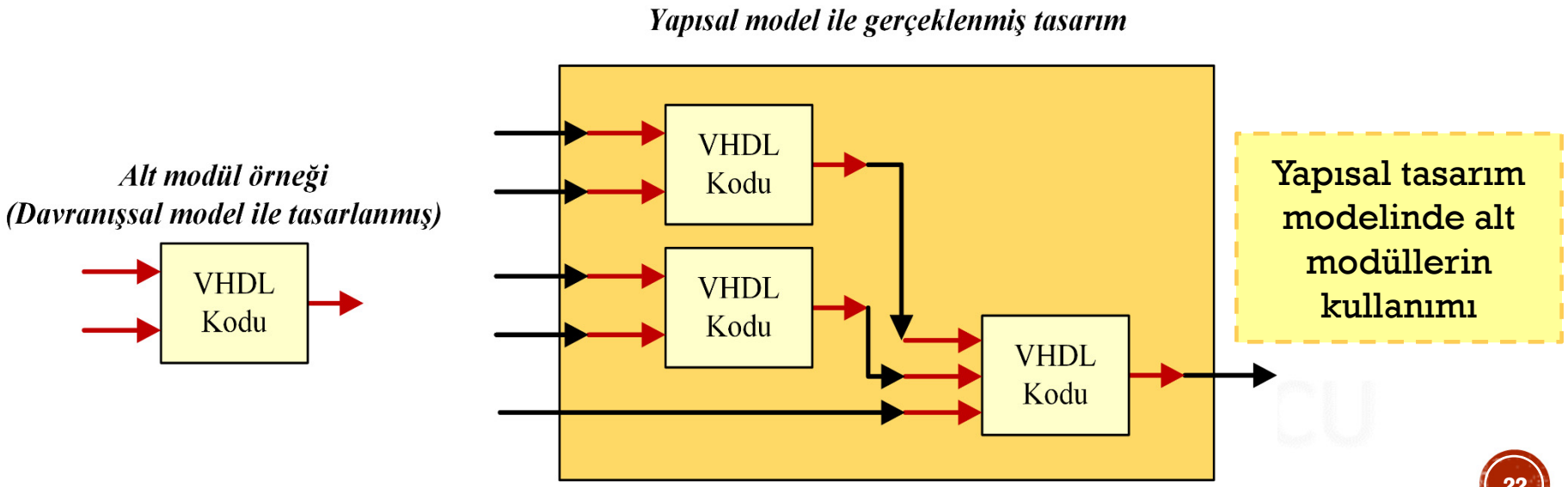
- **Yapısal modellemede** ise bütün bir tasarımın, farklı görevleri içeren alt modüllerden meydana geldiği düşünülür.



3.1.4 Modelleme Teknikleri

- VHLD tasarım kapsamında temel olarak üç tip modelleme tekniği mevcuttur:
  - **1-) Veri akış (Dataflow),**
  - **2-) Davranışsal (Behavioral),**
  - **3-) Yapısal (Structural).**

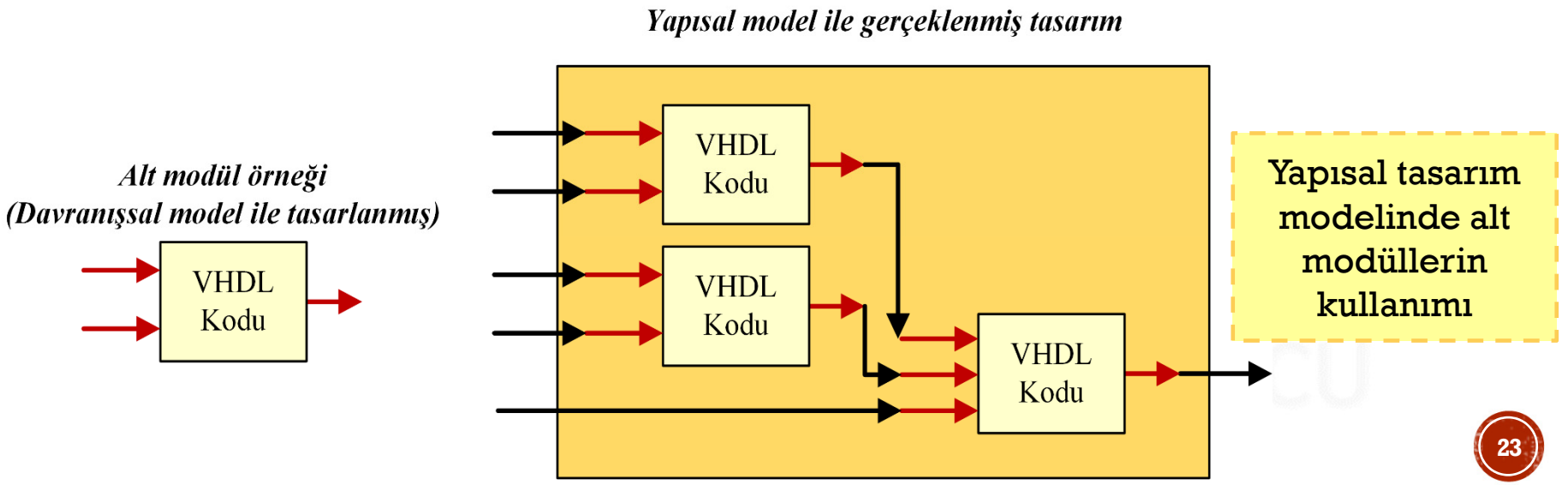
- Bu modüller bir araya getirilerek büyük ve karmaşık sistem tasarımı sağlanır.



3.1.4 Modelleme Teknikleri

- VHLD tasarım kapsamında temel olarak üç tip modelleme tekniği mevcuttur:
  - **1-) Veri akış (Dataflow),**
  - **2-) Davranışsal (Behavioral),**
  - **3-) Yapısal (Structural).**

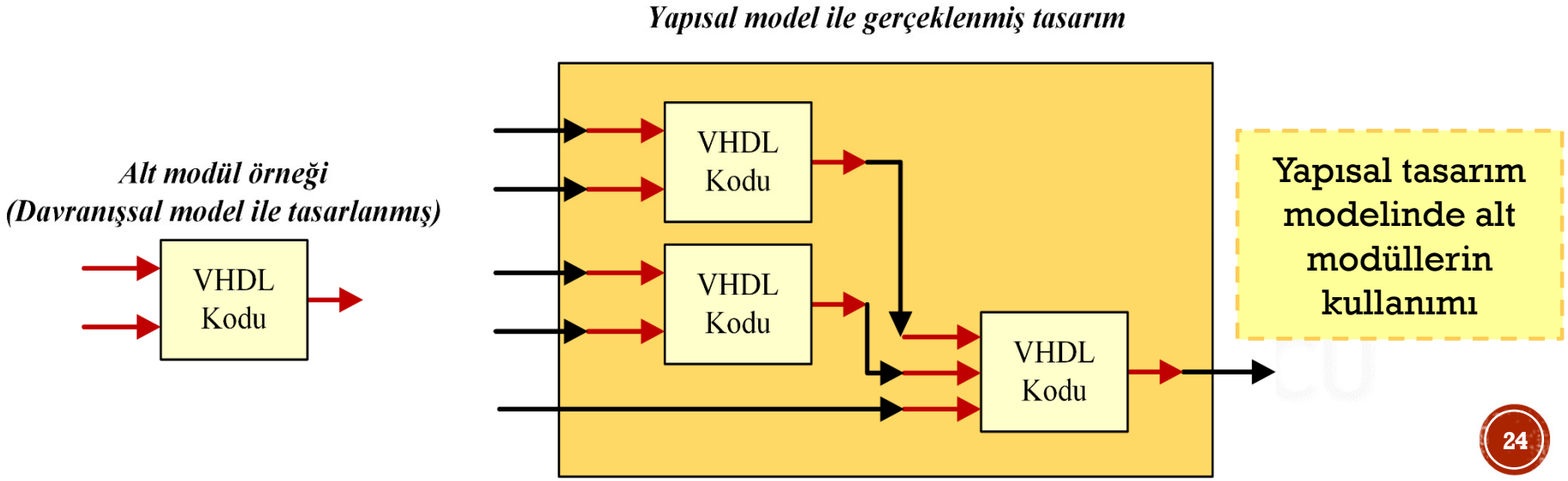
- ***Davranışsal*** ve / veya ***veri-akış modeli*** kullanılarak üretilen alt birimlerin ***yapısal birleştirilmesi***, **yapısal modelleme** kapsamında ele alınır.



3.1.4 Modelleme Teknikleri

- VHLD tasarım kapsamında temel olarak üç tip modelleme tekniği mevcuttur:
  - **1-) Veri akış (Dataflow),**
  - **2-) Davranışsal (Behavioral),**
  - **3-) Yapısal (Structural).**

- VHDL tasarımda genel olarak her bir alt birim (modül) **davranışsal modelleme** ele alınarak tasarlanır ve bütün modüller **yapısal modelleme** kullanılarak birbirine bağlanır.



3.1.4 Modelleme Teknikleri

ÖZETLE

- VHLD tasarım kapsamında temel olarak üç tip modelleme tekniği mevcuttur:

1-) Veri akış (Dataflow),

2-) Davranışsal (Behavioral),

3-) Yapısal (Structural).

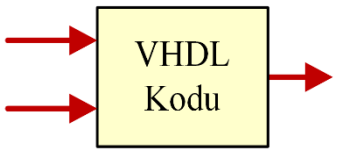
Eş zamanlı (paralel) işletilecek kod ile tasarımdır

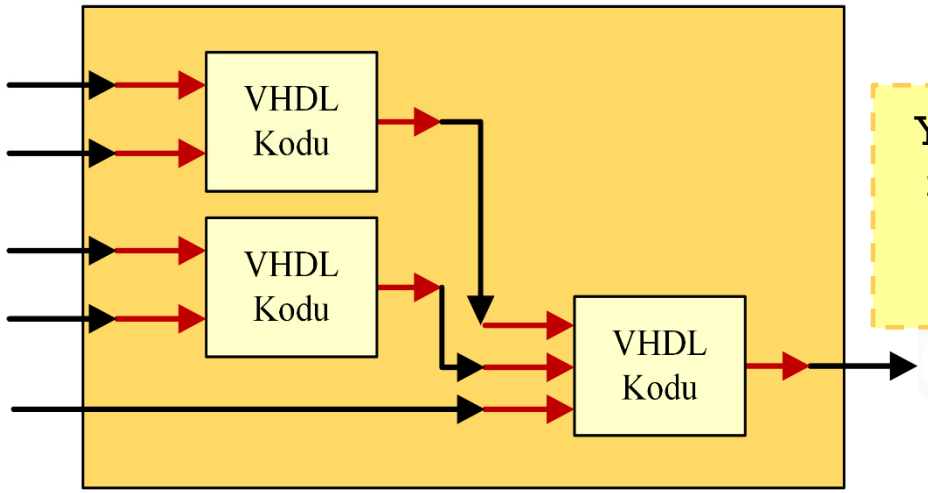
Ardışıl (sıralı) + Eş zamanlı (paralel) işletilecek kod temelli tasarımdır

Birden fazla veri akış ve/veya davranışsal tasarımın birleştirilmesidir.

Yapısal model ile gerçeklenmiş tasarım

Alt modül örneği
(Davranışsal model ile tasarlanmış)





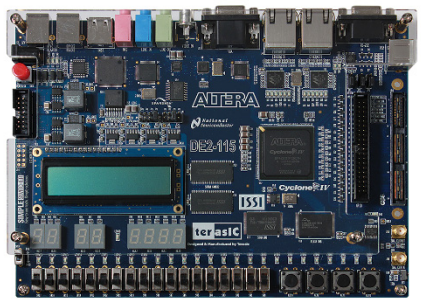
Yapısal tasarım modelinde alt modüllerin kullanımı

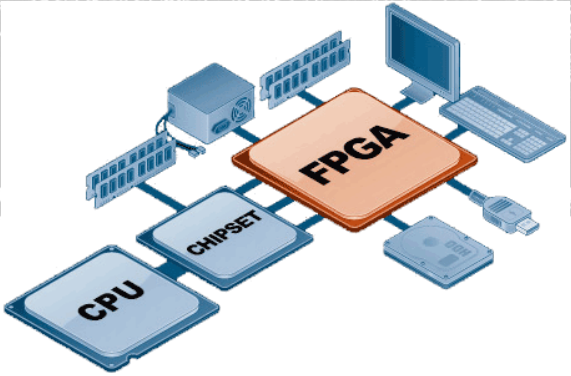
SONRAKİ DERS KONUSU

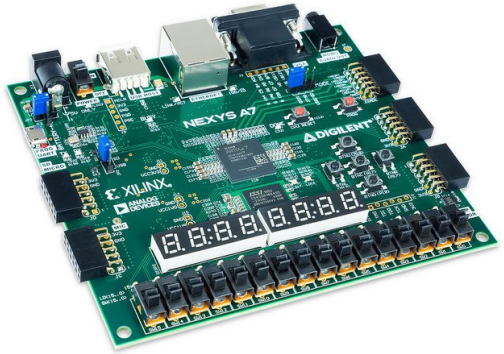
4- VHDL TASARIM BÖLÜMLERİ

İLERİ SAYISAL SİSTEMLER

4- VHDL TASARIM BÖLÜMLERİ







Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

4. VHDL TASARIM BÖLÜMLERİ

- VHDL dilinde 3 ayrı ana tasarım yapısı mevcuttur. Bunlar; **varlık** (**entity**), **mimari** (**architecture**) ve **kütüphane** (**library**) şeklindedir.

- Bu bölümlere ek olarak, bazı kaynaklarda **konfigürasyon** bölümü ayrı bir başlıkta ele alınır.

- **Library:** **Halihazırda mevcut olan** veya **kullanıcı tarafından hazırlanan** tanımlamaların gruplandığı yapıdır. Bu tanımlama grupları (**kütüphaneler** veya **paketler**) ihtiyaca göre tanımlanır ve kullanılır.

- **Entity:** Tasarımın dış çevresi ile bağlantı arayüzünü oluşturur. Bu kısımda giriş ve çıkış portları belirtilir.

- **Architecture:** Modele dair davranışın tanımlandığı bölümdür.

- **Konfigürasyon:** Alt modüllerin bir araya gelerek bütün bir sistem tasarımının nasıl oluşturulduğunu gösteren bölümdür.

4.1 Varlık (Entity)

- **Varlık** (**Entity**) bildirimi, tasarlanacak sisteme dair giriş ve çıkış birimlerinin belirtildiği bölümdür.

- VHDL tasarımlarda **yalnızca bir adet entity** tanımlanır ve bu tanımlama **kütüphane** (paket, library) tanımlaması sonrasında yapılır.

- Aynı **entity** tanımlaması üzerinden **birden fazla architecture** ve **konfigürasyon** tanımlaması yapılabilir.

- **Port** olarak isimlendirilen ve sistemin dış çevresi ile bağlantısını oluşturan girdi ve çıktı birimlerinin tanımlamaları **entity** bölümünde verilir.

- **Portlar**; giriş, çıkış, hem giriş hem çıkış veya tampon olacak şekilde dört farklı formatta tanımlanır.

4.1 Varlık (Entity)

- **in:** Sisteme dışarıdan gelen sinyalleri tanımlarken kullanılır ve yalnızca ***okuma yapılabilen port*** tanımlamasıdır.

- **out:** Sistemden çıkış yapacak olan sinyalleri tanımlarken kullanılır ve yalnızca ***yazma yapılabilen port*** tanımlamasıdır.

- **inout:** Sisteme ***hem giriş hem de çıkış*** yapacak olan sinyaller için kullanılan ***port*** tanımlamasıdır.

- **buffer:** ***out portu*** gibi davranır ve bu porttan farklı olarak ***sistem içinde okuma yapılabilir.***

4.1 Varlık (Entity)

- **Jenerik** (**Generic**) ise **entity**' ye ait bazı parametreler için sistem bileşenleri üzerine kod bölümünde kolaylık sağlayan tanımlamadır.

- **Statik bilgi** (**sabit değer**) sağlayan tanımlamalardır.

- **Generic** tanımlaması **entity içinde port bilgisinden önce verilir** ve yazılan tanımlama **entity ve entity' nin ilişkili olduğu architecture bölümlerinde** kullanılabilir.

- **Generic**' lere dair kullanımlardan bazıları şunlardır:
  - Port boyutlarının ifadesi
  - Alt bileşen sayısının nitelenmesi
  - Zamanlama özellikleri
  - Tasarıma dair fiziksel özellik tanımlamaları
  - Architecture içinde belirtilecek vektörlerin uzunluk tanımlamaları
  - Döngü sayısı atanması

4.1 Varlık (Entity)

- **Entity** yapısının gösterimi şu şekildedir:

```
Entity entity_ismi is
    Generic (jenerik_arayüz_listesi);
    Port (port_ismi: mod tür;
          diğer portlar…);
End entity_ismi;
```

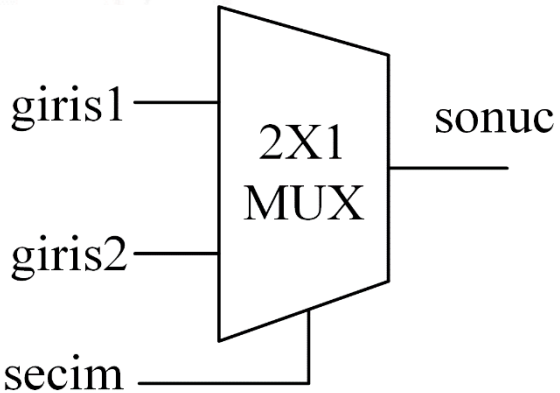
Örnek 4.1: Yapısında clock ve reset sinyal girişleri, 16 bitlik sayı girişi ve 8' er bitlik iki sayı çıkışı için port tanımlamasının yapıldığı **entity** kodunu yazınız.

```
Entity Uyg1 is
    Port (clk: in  std_logic;
          rst: in  std_logic;
          sayi_16b: in  std_logic_vector(15 downto 0);
          sayi1_8b: out  std_logic_vector(7 downto 0);
          sayi2_8b: out  std_logic_vector(7 downto 0));
End Uyg1;
```

4.1 Varlık (Entity)

Örnek 4.2: *giris1*, *giris2* ve *secim* isimli üç giriş portu ile *sonuc* isimli bir çıkış portuna sahip 2x1 MUX için **entity** tanımlaması yapınız.

```
Entity mux_birimi is
    Port (giris1:  in   std_logic;
          giris2:  in   std_logic;
          secim:   in   std_logic;
          sonuc:   out  std_logic);
End mux_birimi;
```



Örnek 4.3: 16 bitlik adres giriş bilgisi gerektiren RAM için, **generic** tanımlamasını adres bit sayısı üzerinden belirterek gerekli **entity** kodunu yazınız.

```
Entity Ram_birimi is
    Generic (adres_boyut: integer := 16);
    Port (adres:  in  std_logic_vector(adres_boyut-1 downto 0);
End Ram_birimi;
```

4.1 Varlık (Entity)

Entity kodu yazılırken bazı kurallar mevcuttur:

1. Entity ismi bütün bir tasarımın ismidir ve bir harfle başlamak koşuluyla ***harf***, ***alt çizgi*** ve ***rakam*** içerebilir.

2. Bir ***generic***, ***port*** veya ***pasif ifade*** kullanmadan yalın halde **entity** tanımlaması yapılabilir.

3. Aynı tür ve port tanımlaması (modu) aynı olan tanımlamalar, birbirlerinin arasına virgül (,) konularak yapılır. Herbir port tanımlaması noktalı virgül (;) ile ayrılır.

```
Entity elevator_control is
    Port (yukari, asagi, dur, acil:  in std_logic;
          yon_kontrol, diger_islem:  out  std_logic);
End elevator_control;
```

4.1 Varlık (Entity)

Entity kodu yazılırken bazı kurallar mevcuttur:

4. Sistem tasarımına bağlı olarak **entity** kodunda tür, alt tür ve sabitler tanımlanabilir. Tanımlı ifadeler ise **entity**' ye bağlı olan bütün **architecture** bölümlerinde kullanılabilir.

```
Entity deneme is
    Port (A, B, C, D:  in std_logic;
          E, F:  out  std_logic);
          signal deg1: std_logic;
          constant deg2: integer :=20;
End deneme;
```

```
Library ieee;
Use ieee_std_logic_1164.all;

Entity demux_1 is
    Port (A, sel:  in std_logic;
          B, C:  out  std_logic);
End demux_1;
```

5. Entity tanımlaması **kütüphane** tanımlaması sonrası yapılır.

4.1 Varlık (Entity)

Örnek 4.4: 16 bitlik adres giriş bilgisi gerektiren RAM için, **generic** tanımlamasını adres bit sayısı üzerinden belirterek gerekli **entity** kodunu yazınız.

```
Entity RAM_birimi is
    Generic (adres_boyut: integer :=16);
    Port (adres_bilgisi : in  std_logic_vector(adres_boyut-1 downto 0);
End RAM_birimi;
```

4.2 Mimari (Architecture)

- **Mimari** (**Architecture**) bildirimi, tasarıma dair gerçekleştirilecek işin tanımlandığı bölümdür.

- Diğer bir deyişle, ***tasarım davranışının ve iç yapısının tasarlandığı bölümdür.***

4.2 Mimari (Architecture)

- Ayrıca modellemede sıralı veya paralel işlem tiplerinden hangisinin kullanacağı da bu bölümde belirtilir.

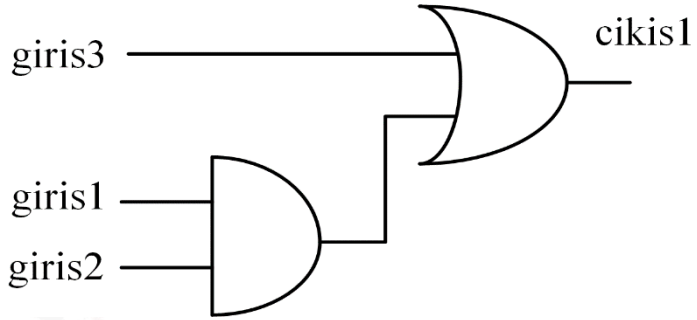
- **Architecture** kodu genelde dört farklı şekilde yazılabilir: 1-Veri akış, 2-Yapısal, 3-Davranışsal, 4-Karma.

- **Architecture** yapısının gösterimi şu şekildedir:

```
Architecture architecture_ismi of entity_ismi is
    bildirimler (sinyal, değişken vb. tanımlamalar);
Begin
    architecture gövdesi
End architecture_ismi;
```

4.2 Mimari (Architecture)

Örnek 4.5: Yan tarafta görülen devre için gerekli VHDL kodunu yazınız.



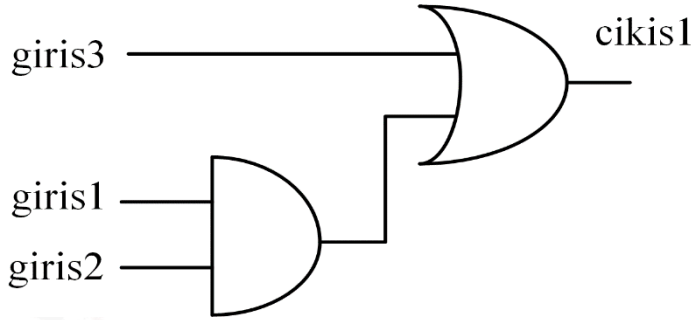

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity tumlesik_devre is
Port (giris1, giris2, giris3: in std_logic;
        cikis1: out std_logic);
End tumlesik_devre;

Architecture icyapi of tumlesik_devre is
    signal ara1: std_logic;
Begin
    ara1 <= giris1 and giris2;
    cikis1 <= ara1 or giris3;
End icyapi;
```

Architecture kodu aşağıdaki gibi ara bağlantı kullanmadan da yazılabilir.

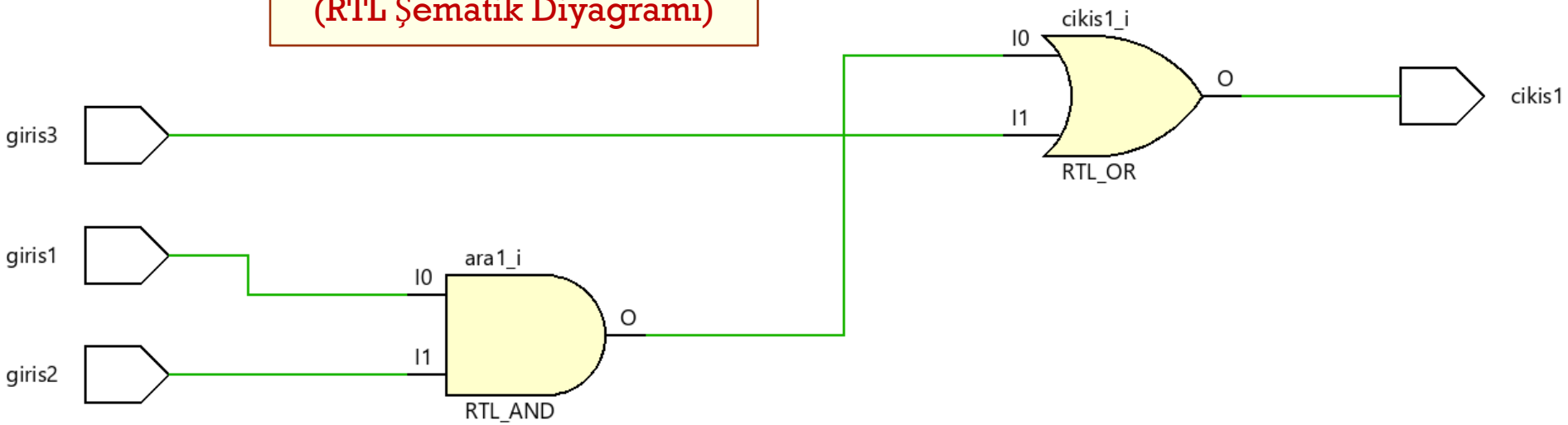
```
cikis1 <= (giris1 and giris2) or giris3;
```

4.2 Mimari (Architecture)

Örnek 4.5: Yan tarafta görülen devre için gerekli VHDL kodunu yazınız.



Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



4.2 Mimari (Architecture)

Architecture kodu yazılırken bazı kurallar mevcuttur:

1. Bütün **architecture** yapısının tek ismi olur ve bu isim **architecture** ifadesi sonrası yazılır.

- **Architecture** ismi sonrasında sırasıyla '**of**' ifadesi ve **entity** ismi belirtilmelidir.

- Bütün bir **architecture** tanımının yapıldığı satır '**is**' ifadesi ile sonlandıktan sonra kod tanımlamalarına giriş yapılır.

Architecture icyapi **of** tumlesik_devre **is**

4.2 Mimari (Architecture)

Architecture kodu yazılırken bazı kurallar mevcuttur:

2. İlk satır (**architecture** tanımının yapıldığı satır) sonrasında **begin** (başlangıç) komutuna kadar çeşitli tanımlamalar (sinyal, değişken, sabit, alt program, bileşen ve veri türü gibi tanımlamalar) yapılır.

- Ara değer atamasında kullanılacak olan ara sinyal tanımlamaları, **architecture** ilk satırı sonrasında **begin** komutuna kadar olan bölümde yapılmalıdır.

```
signal ara1: std_logic;
```

3. Begin ve **end** komutları arası **tasarım tanımlama bölümüdür**. Bu bölümde ***sinyal atamaları***, ***process ve bileşen çağrılması*** gibi işlemler gerçekleştirilir.

```
Begin
    ara1 <= giris1 and giris2;
    cikis1 <= ara1 or giris3;
End icyapi;
```

4.2 Mimari (Architecture)

Architecture kodu yazılırken bazı kurallar mevcuttur:

4. Architecture içinde tanımlanan bütün işlem ve ifadeler (sıralı komutlar hariç) eş zamanlı işletilir.

5. Architecture bittiğinde (**begin** komutunun karşılığı olarak) **end** komutu kullanılır ve bu komut sonrası başta tanımlanan **architecture ismi** yazılır.

End icyapi;

6. Her **architecture** yapısının bir **entity**' ye bağlı olması gerekir.

- Bir **entity** birden fazla **architecture** içerebilir, ancak tersi durum söz konusu değildir.

4.2 Mimari (Architecture)

Architecture kodu yazılırken bazı kurallar mevcuttur:

6. Her **architecture** yapısının bir **entity**' ye bağlı olması gerekir.

- Bir **entity** birden fazla **architecture** içerebilir, ancak tersi durum söz konusu değildir.

- Bu durumda ise herbir **architecture** isminin farklı olması gerekir.

- Kısaca bir **architecture** yalnızca tek bir **entity** ile ilişkili olabilir.

Architecture icyapi **of** tumlesik_devre **is**

→ burada **architecture** ismi **icyapi**, bağlı olduğu **entity** ise **tumlesik_devre** olarak görülür.

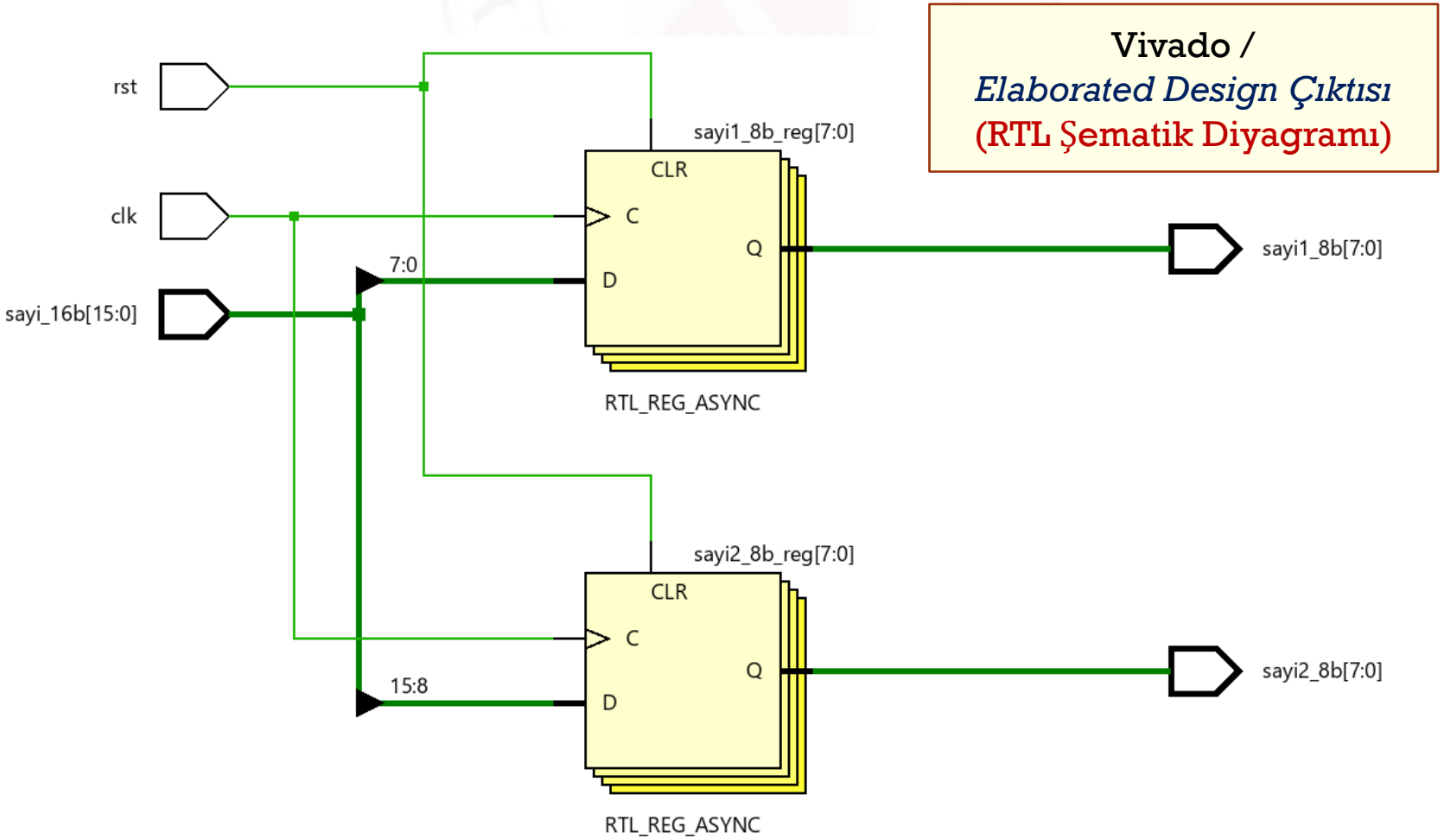
4.2 Mimari (Architecture)

Örnek 4.6: Girişe gelen 16 bitlik sayıyı iki parçaya bölen, ***clk*** ve ***rst*** sinyallerine göre işlem yapan tasarım için gerekli VHDL kodunu yazınız.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity Uyg_bolum is
Port (clk, rst: in std_logic;
      sayi_16b: in std_logic_vector(15 downto 0);
      sayi1_8b: out std_logic_vector(7 downto 0);
      sayi2_8b: out std_logic_vector(7 downto 0));
End Uyg_bolum;

Architecture davranis of Uyg_bolum is
Begin
   Process (clk, rst)
   Begin
      If (rst= '1') then
           sayi1_8b <= "00000000"; -- veya  sayi1_8b <= (others => '0');
           sayi2_8b <= "00000000"; -- veya  sayi2_8b <= (others => '0');
      Elsif rising_edge(clk) then
           sayi1_8b <= sayi_16b(7 downto 0);
           sayi2_8b <= sayi_16b(15 downto 8);
      End if;
  End process;
End davranis;
```

4.2 Mimari (Architecture)

Örnek 4.6: Girişe gelen 16 bitlik sayıyı iki parçaya bölen, ***clk*** ve ***rst*** sinyallerine göre işlem yapan tasarım için gerekli VHDL kodunu yazınız.



4.3 Kütüphane (Library)

- Bir tasarım içinde VHDL kaynak kodlarının derlenmesi için **Kütüphane** (**Library**) yapılarına ihtiyaç duyulur.

- Kullanıcı kendisinin tanımlayacağı **paket** (**package**) yapıları üzerinden kendi oluşturacağı **kütüphaneyi** kullanabileceği gibi, derleyicide mevcut olan **kütüphaneleri** de kullanabilmektedir.

- **Kütüphanenin** kullanıma açılması için **library** komutu kullanılır.

- Komut sonrası kullanılacak **kütüphane** ismi belirtilir (**Library** *ieee*, vb gibi).

4.3 Kütüphane (Library)

Çalışma Kütüphanesi: Bir tasarım kapsamında oluşturulmuş VHDL kodlarını kapsayan kütüphanedir.

- Mevcut tasarım kodları her zaman ***çalışma kütüphanesinde*** derlenmektedir.

- Bu nedenle çalışma kütüphanesi halihazırda mevcut olduğundan ayrıca bir tanımlama ile program içinde belirtilmez.

- Derleyici programda varsayılan kütüphane "**work**" olarak bilinir.

- Kullanıcı tarafından kullanılabilecek olan **paket** (**package**) tanımlamaları bu kütüphane kapsamında incelenebilir.

4.3 Kütüphane (Library)

- Veri türleri, alt programlar, sabitler, sinyal ve bileşenler gibi pek çok tanımlamayı içinde bulunduran genel yapılara **paket** denir.

- Kullanıcı işletilmesi gereken birçok tanımlamayı **paket** içerisine yerleştirerek, bu tanımlamaların bütün modül ve alt modüllerde kullanılmasını sağlayabilir.

- **Paket** kullanımı, gerekli tanımlamaların birden fazla kez yapılmasını önler ve tasarım kolaylığı sağlar.

- **Paket** yapısı ***bildirim*** ve ***gövde*** olmak üzere iki alt başlıkta tanımlanır.

4.3 Kütüphane (Library)

- **Paket bildirim** bölümünde veri türü tanımlamaları ve çeşitli bildirimler (sabit, fonksiyon ve prosedür) yapılır.

```
Package paket_ismi is
    paket bildirimleri
End package paket_ismi;
```

- **Paket gövdesi** ise alt program tanımlamalarını içermektedir. Bu bölüm **bildirim** kısmını basitleştirerek derleme işlemini kolay hale getirir. Yine fonksiyon ve prosedürler bu bölümde yer alır.

```
Package body paket_ismi is
    alt program tanımlamaları
End package body paket_ismi;
```

4.3 Kütüphane (Library)

Paket Oluşturma Örneği

Paket Bildirimi

```
Package deneme is
    constant katsayi_deg: integer;
    type hava_durum is (bulutlu,karli,gunesli);
    Function sayma(k: integer) return integer;
End package deneme;
```

Paket Gövdesi

```
Package body deneme is
    constant katsayi_deg: integer :=10;
    Function sayma(k: integer) return integer is
    Begin
    ….
    End sayma;
End package body deneme;
```

4.3 Kütüphane (Library)

Standard Kütüphanesi: **Standard** (**Std**) kütüphanesi hâlihazırda VHDL içinde tanımlı bir kütüphanedir ve kullanıcı tarafından tanımlanmasına gerek yoktur. Ön tanımlı VHDL bileşenlerini içerir.

- Kütüphane kapsamında çeşitli bileşenler mevcuttur. Bu kütüphane iki paket içerir: ***1-Standard***, ***2-Textio***

- ***Standard*** başlığı altında bit, boolean, integer gibi basit veri tanımlamaları ve tanımlamalara dair fonksiyonlar yer alır.

- ***Textio*** paketi ise metin dosyası işlemleri kapsamında prosedür, tür ve fonksiyonları barındırır.

4.3 Kütüphane (Library)

IEEE Kütüphanesi: Bu kütüphane kapsamında en son ve güncel paket ***IEEE Standart 1164*** paketidir. Kullanılan paketler ve ilgili özellikler aşağıdaki gibidir:

Use *ieee.std_logic_1164.all* : std_logic, std_ulogic, std_logic_vector, std_ulogic_vector ile ilgili fonksiyonları içerir.

Use *ieee.std_logic_arith.all* : signed, unsigned, integer, std_ulogic, std_logic ve std_logic_vector türleri için aritmetik, dönüşüm ve karşılaştırma fonksiyonlarını kapsar.

Use *ieee.std_logic_unsigned.all* : işaretsiz aritmetik fonksiyonları kapsar.

Use *ieee.std_logic_signed.all* : işaretli aritmetik fonksiyonları kapsar.

4.3 Kütüphane (Library)

Use *ieee*.math_complex.*all* : karmaşık sayılarla ilintili fonksiyonları kapsar.

Use *ieee*.*math_real*.*all* : gerçek sayılarla ilintili fonksiyonları kapsar.

Use *ieee*.*numeric_bit*.*all* : bit türünde aritmetik işlem fonksiyonlarını kapsar.

Use *ieee*.*numeric_std*.*all* : std_logic türündeki verilerin aritmetik işlem fonksiyonlarını içerir. std_logic_arith paketinin alternatifidir.

Use *ieee*.*std_logic_misc*.*all* : std_logic_1164 paketini destekler nitelikte veri türleri, fonksiyonları ve sabitleri kapsar.

Use *ieee*.*std_logic_textio*.*all* : dosyalara veri yazma ve dosyalardan veri okuma kapsamında prosedürleri içerir.

4.3 Kütüphane (Library)

Özetle

- Kullanıcı **paket** (**package**) yapılarını kullanarak kendi kütüphanesini oluşturabilir.

- **Çalışma** kütüphanesi derlemenin yapıldığı temel kütüphanedir ve tanımlanma gereksinimi yoktur. **Paket** tanımlamaları bu kütüphane kapsamında yapılır.

- **Standard** kütüphanesi ön tanımlı bileşenlerin bulunduğu kütüphanedir ve (Textio harici) tanımlanma gereksinimi yoktur.

- Devre tasarımlarında **IEEE** kütüphanesinin kendisine ve kullanılacak alt türe dair tanımlama her zaman yapılmalıdır.

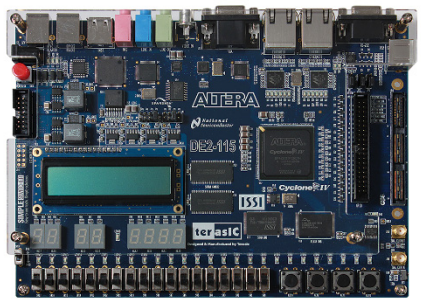
- Bahsi geçen kütüphaneler dışında üreticiye (Altera, Xilinx, vb. gibi) özel kütüphaneler de mevcuttur.

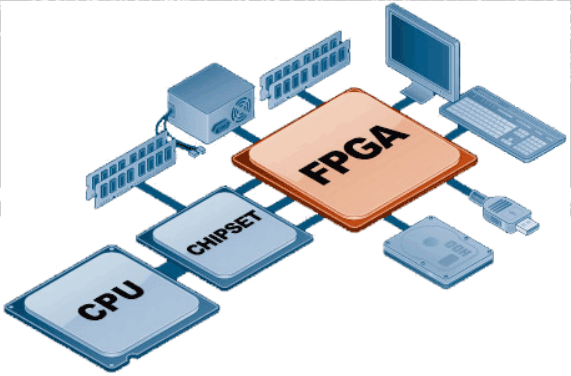
SONRAKİ DERS KONUSU

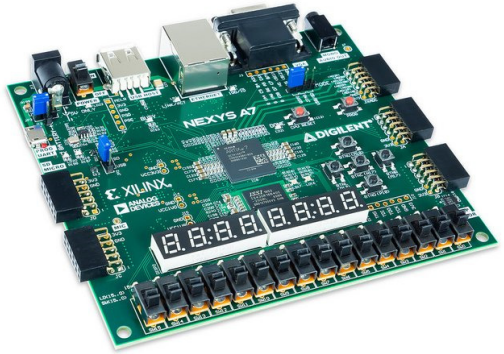
5- VHDL'DE NESNELER

İLERİ SAYISAL SİSTEMLER

5- VHDL'DE NESNELER







Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

5. VHDL'DE NESNELER

- Nesneler, VHDL yapısı içinde genel olarak üç başlıkta tanımlanır. Bunlar; **sinyaller**, **değişkenler** ve **sabitlerdir**.

- **Sinyal** ve **değişken** verilerine dair değerler tasarım kapsamında sürekli farklılık gösterebilirken, **sabit** olarak tanımlanan verilerde değer hiçbir zaman değişmez.

5.1 Sinyaller (Signals)

- **Sinyaller**, bir tasarımda devre için **ara bağlantıları sağlayan nesnelerdir.**

- Diğer bir deyişle, sistemdeki veri akışını açıklayan nesnelerdir.

5.1 Sinyaller (Signals)

- Sinyaller, **architecture** içinde eşzamanlı yapılar arasındaki fiziksel iletişimi sağlar.

- **Sinyaller**; **package**, **architecture** ve **entity** içinde **tanımlanır**, ancak **process**, **procedure** ve **function** içinde **tanımlanamazlar**.

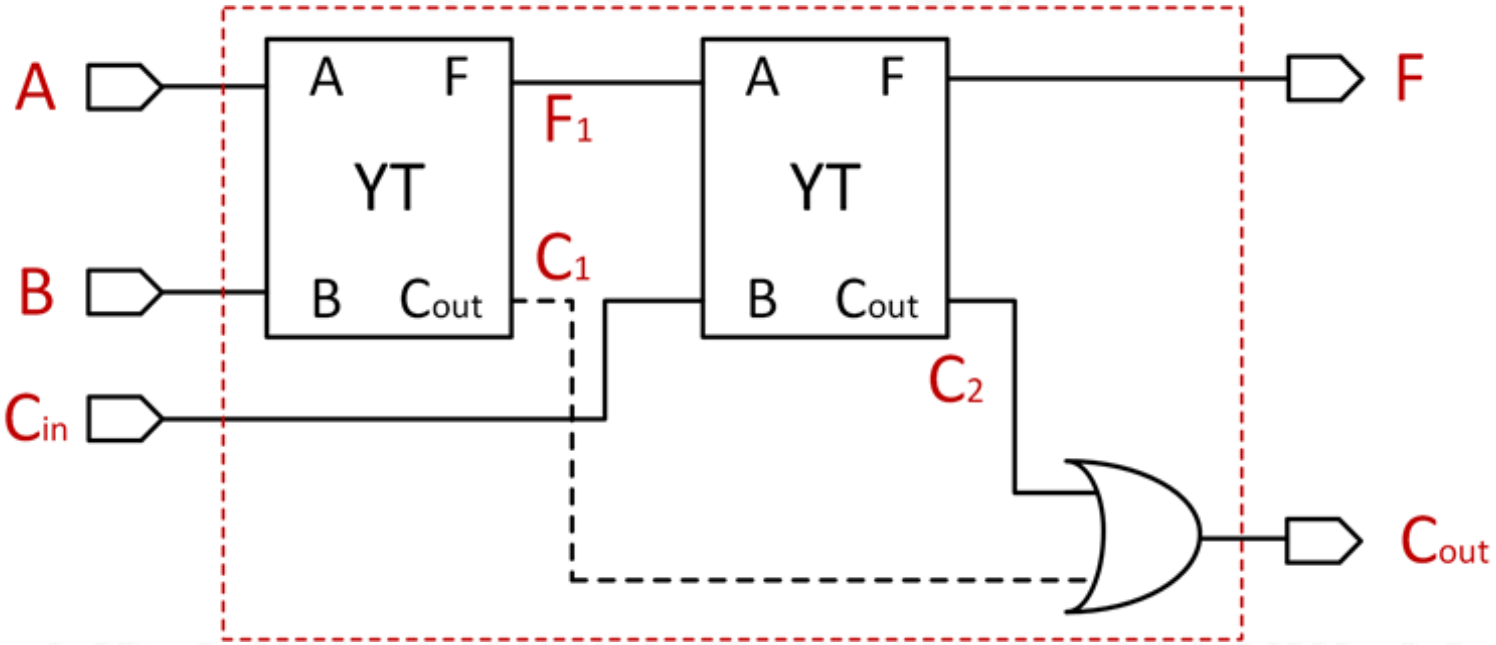
- **Package** ve **entity** içinde tanımlandıklarında alt birimlerde (farklı **architecture** yapılarında) tekrar tanımlanmalarına gerek yoktur.

- Tekil bir **architecture** içinde tanımlanan sinyal ise yalnızca o mimari içinde işletilebilir.

5.1 Sinyaller (Signals)

- F_1, C_1 ve C_2 verileri sinyal nesnesi olarak ifade edilir.

- Sistem içindeki ***Yarı Toplayıcı*** iki bileşenin sinyal(ler) ile bağlanması gereklidir.

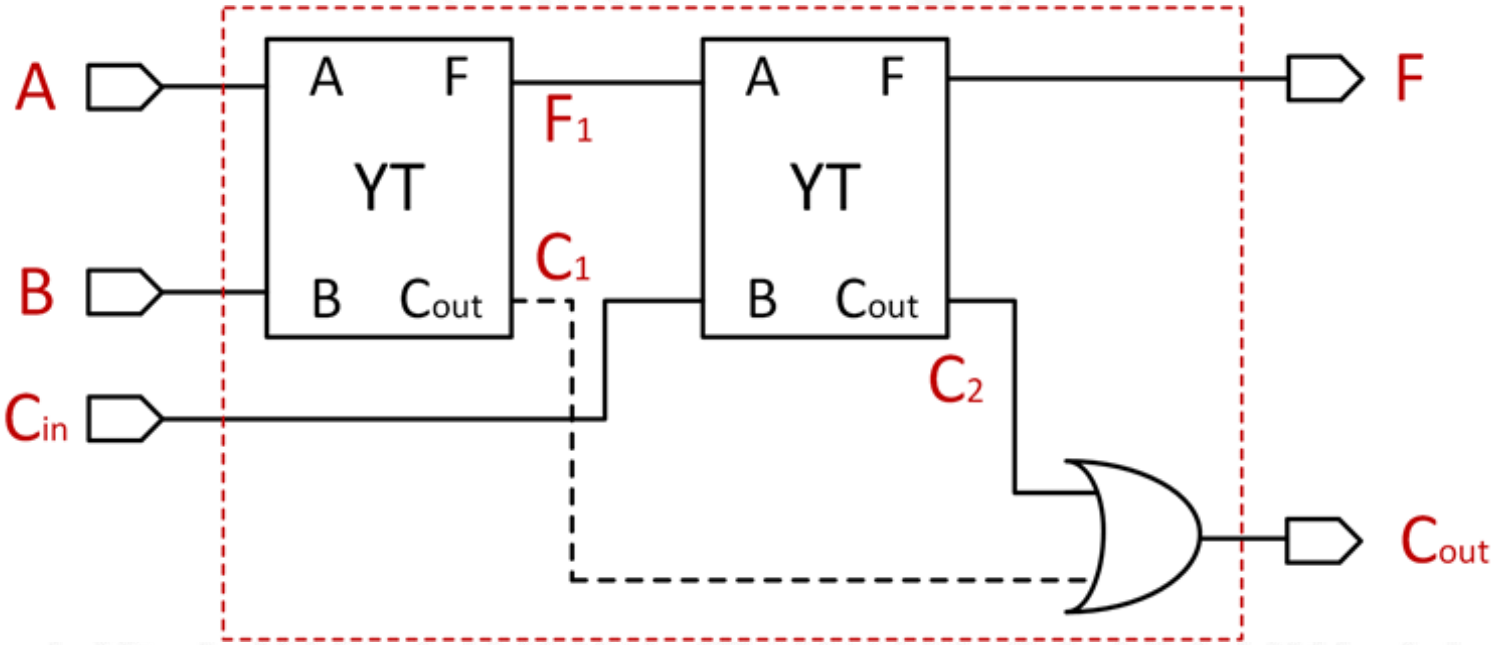


Bir bitlik tam toplayıcı üzerinden sinyal tanımlama örneği

5.1 Sinyaller (Signals)

- Benzer şekilde C_1 ve C_2 sinyalleri bileşenlerden mantık kapısına bilgi taşımaktadır.

- Bu noktada sinyaller, devre yollarının üstlendiği görevleri gerçekleştirir.



Bir bitlik tam toplayıcı üzerinden sinyal tanımlama örneği

5.1 Sinyaller (Signals)

- Sinyaller atanırken kod bloğu içinde '<=' sembolü kullanılır. Bu sembol C dilindeki '=' ifadesine karşılık gelir.

- İlk değer atamaları için ise '**:=**' ifadesi işletilir ve bu amaçla aşağıdaki kod satırı kullanılır.

signal isim: ***tür*** := ilkdeğer;

İlk değer ataması için

Örneğin; **signal** vektor: *std_logic_vector*(7 *downto* 0) := "00001111";

signal dortlu: *std_logic_vector*(3 *downto* 0) := "1111";

signal seviye: *bit* := '0';

signal yil: *integer* := 1980;

5.1 Sinyaller (Signals)

- Sinyallere ilk değer ataması "**:=**" işareti ile sağlanırken, normal değer ataması "**<=**" işareti ile sağlanır. İlintili örnekler aşağıdaki gibidir;

Normal atama için

- ✓ yil <= 2011;
- ✓ seviye <= '1';
- ✓ vektor(2) <= '0';
- ✓ vektor <= "00001111";
- ✓ dortlu <= "1111";
- ✓ dortlu <= b"1111";
- ✓ dortlu <= x"F";
- ✓ dortlu <= (others=>'1');

5.1 Sinyaller (Signals)

conv_std_logic_vector fonksiyonunu barındırır.

Tasarım-1

Örnek 5.1: 4-bitlik ileri sayıcı tasarımı için gerekli VHDL kodunu yazınız.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Use ieee.std_logic_unsigned.all;

Entity sayici_4b is
Port (clk: in std_logic;
      rst: in std_logic;
      cikis: out std_logic_vector(3 downto 0));
End sayici_4b;

Architecture davranis of sayici_4b is
  signal sayma: integer := 0;
Begin
  Process (clk, rst)
  Begin
    If (rst = '1') then
       sayma <= 0;
    Elsif rising_edge(clk) then
       sayma <= sayma +1;
    End if;
  End process;
  cikis <= conv_std_logic_vector(sayma, cikis'length);
End davranis;
```

std_logic_vector türünde nesne için (**sayma** türü **integer** yerine **std_logic_vector** seçilirse) doğrudan aritmetik işlem yapılabilmesini sağlar.

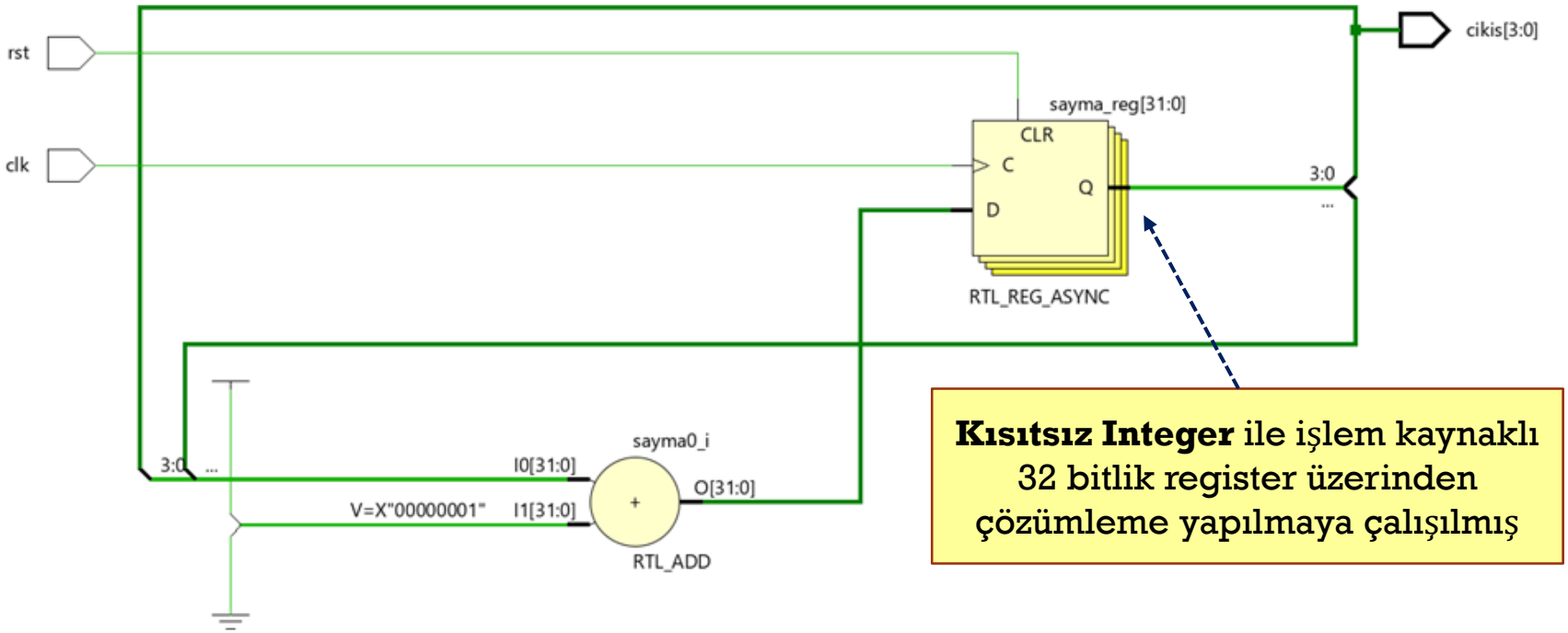
5.1 Sinyaller (Signals)

Örnek 5.1: 4-bitlik sayıcı tasarımı için gerekli VHDL kodunu yazınız.

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

Tasarım-1

```
Architecture davranis of sayici_4b is
   signal sayma: integer := 0;
Begin
   Process (clk, rst)
   Begin
      If (rst = '1') then
            sayma <= 0;
      Elsif rising_edge(clk) then
            sayma <= sayma +1;
      End if;
   End process;
   cikis <= conv_std_logic_vector(sayma, cikis'length);
End davranis;
```



Kısıtsız Integer ile işlem kaynaklı 32 bitlik register üzerinden çözümleme yapılmaya çalışılmış

5.1 Sinyaller (Signals)

conv_std_logic_vector fonksiyonunu barındırır.

Tasarım-2

Örnek 5.1: 4-bitlik sayıcı tasarımı için gerekli VHDL kodunu yazınız.

std_logic_vector türünde nesne için (**sayma** türü **integer** yerine **std_logic_vector** seçilirse) doğrudan aritmetik işlem yapılabilmesini sağlar.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Use ieee.std_logic_unsigned.all;

Entity sayici_4b is
Port (clk: in std_logic;
      rst: in std_logic;
      cikis: out std_logic_vector(3 downto 0));
End sayici_4b;

Architecture davranis of sayici_4b is
  signal sayma: integer range 0 to 15 := 0;
Begin
  Process (clk, rst)
  Begin
    If (rst = '1') then
       sayma <= 0;
    Elsif rising_edge(clk) then
       If sayma = 15 then
          sayma <= 0;
       Else
          sayma <= sayma +1;
       End if;
    End if;
  End process;
  cikis <= conv_std_logic_vector(sayma, cikis'length);
End davranis;
```

5.1 Sinyaller (Signals)

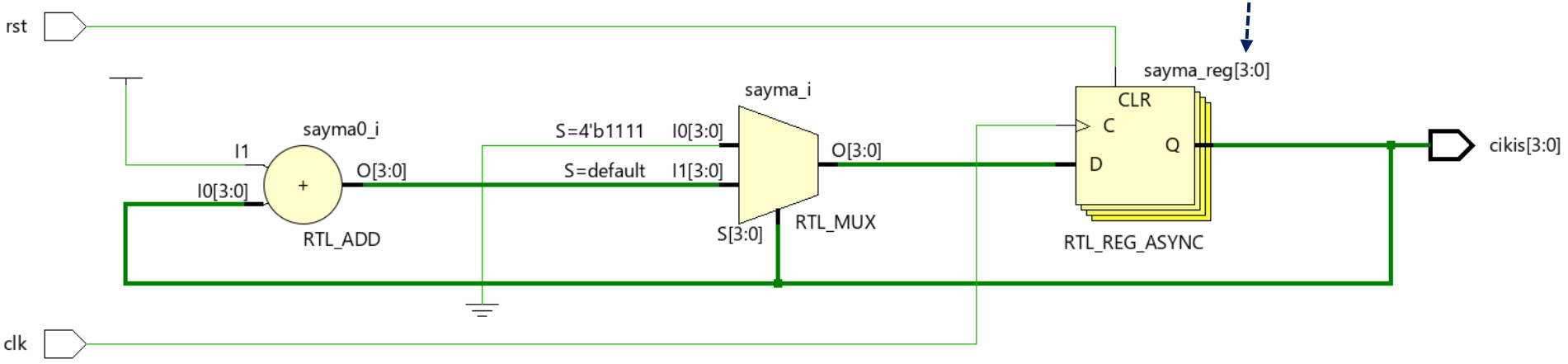
Tasarım-2

Örnek 5.1: 4-bitlik sayıcı tasarımı için gerekli VHDL kodunu yazınız.

```
Architecture davranis of sayici_4b is
   signal sayma: integer range 0 to 15 := 0;
Begin
   Process (clk, rst)
   Begin
      If (rst = '1') then
           sayma <= 0;
      Elsif rising_edge(clk) then
           If sayma = 15 then
                sayma <= 0;
           Else
                sayma <= sayma +1;
           End if;
      End if;
   End process;
   cikis <= conv_std_logic_vector(sayma, cikis'length);
End davranis;
```

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

Integer tanımlama **sınırlandırıldığı** ve **sıfırlama sağlandığı** için çok daha küçük register dizisi kullanılmış



5.1 Sinyaller (Signals)

Tasarım-3

Örnek 5.1: 4-bitlik sayıcı tasarımı için gerekli VHDL kodunu yazınız.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity sayici_4b is
Port (clk: in std_logic;
      rst: in std_logic;
      cikis: out std_logic_vector(3 downto 0));
End sayici_4b;

Architecture davranis of sayici_4b is
  signal sayma: unsigned (3 downto 0):= (others =>'0');
Begin
  Process (clk, rst)
  Begin
    If (rst = '1') then
        sayma <=  (others =>'0');
    Elsif rising_edge(clk) then
        If sayma = "1111" then
            sayma <= (others =>'0');
        Else
            sayma <= sayma +1;
        End if;
    End if;
  End process;
  cikis <= std_logic_vector(sayma);
End davranis;
```

5.1 Sinyaller (Signals)

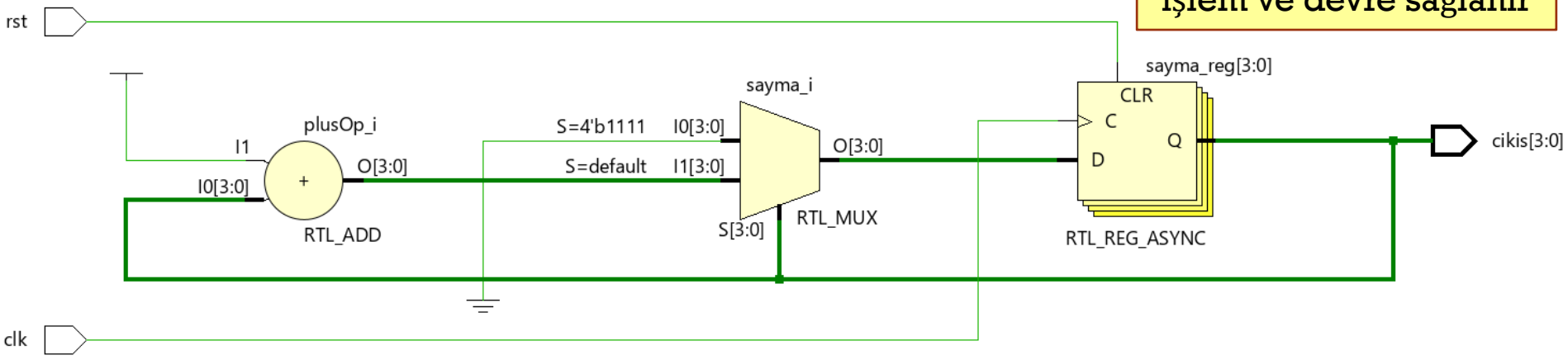
Tasarım-3

Örnek 5.1: 4-bitlik sayıcı tasarımı için gerekli VHDL kodunu yazınız.

```
Architecture davranis of sayici_4b is
   signal sayma: unsigned (3 downto 0):= (others =>'0');
Begin
   Process (clk, rst)
   Begin
      If (rst = '1') then
           sayma <=  (others =>'0');
      Elsif rising_edge(clk) then
           If sayma = "1111" then
              sayma <= (others =>'0');
           Else
              sayma <= sayma +1;
           End if;
      End if;
   End process;
   cikis <= std_logic_vector(sayma);
End davranis;
```

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

Tasarım-2 ile aynı işlem ve devre sağlanır



5.2 Değişkenler (Variables)

- **Değişken**; **process** ve **alt programlar** (**fonksiyon** ve **prosedür**) içinde tanımlanan, değeri değiştirilebilen ve en son değerini muhafaza eden nesnelerdir.

- Herbir **değişken** belirtilen türde bir değer tutar ve sadece tanımlandığı **process** veya **alt program** kapsamında kullanılabilir.

- Aşağıdaki kod satırı, hem ilk değer atamalarında hem de normal (program içindeki) atamalarda kullanılır. Burada işletilen işaret "**:=**" olmalıdır.

variable isim: ***tür*** := ilkdeğer;

İlk değer ataması ve Normal atama için

Örneğin; **variable** deg1: *integer* := 16;

variable deg2: *integer*;

5.2 Değişkenler (Variables)

- İlintili örnekler aşağıdaki gibidir;
  - ✓ yil := 2011;
  - ✓ seviye := '1';
  - ✓ vektor(2) := '0';
  - ✓ vektor := "00001111";
  - ✓ dortlu := "1111";
  - ✓ dortlu := x"F";
  - ✓ dortlu := b"1111";
  - ✓ dortlu := (others=>'1');

İlk değer ataması ve Normal atama için

- ***Veri türleri aynı ise*** değişkenden sinyale veya sinyalden değişkene atama yapılabilmektedir.

5.2 Değişkenler (Variables)

- **Değişken** değeri ise yalnızca **process** ve **alt programlar** içinde işletilebildiğinden, **process** dışına değer atamasında değişkenin bir sinyale aktarılması gerekir.

- Karmaşık hesaplama ve algoritmalar için **process** içinde **değişken** kullanılması tavsiye edilir.

- Sinyaller ile **process** içine aktarılan değerin bir **değişkene** atanması, **process** içindeki işlemin **değişkene** göre sonlandırılması ve **değişkene** dair değerin **process** çıkışında tekrar sinyale aktarılması önerilir.

- ***Simülasyon sırasında değişkenler, genellikle sinyallere göre daha az bellek tüketir ve daha hızlı işlem yapar*, çünkü sadece ilgili process içinde geçici olarak tutulurlar.**

- Ancak ***donanım sentezi açısından, kalıcı veri saklama (depolama elemanı) için sinyal kullanımı gereklidir.***

5.2 Değişkenler (Variables)

- Bir **değişken** birden fazla **process** veya **alt program** içinde işletilecekse ***ortak değişken*** olarak tanımlanmalıdır.

- Bu tanımlama **architecture tanımlama satırı** ve **begin** komutları arasına yapılabilir.

- Aşağıdaki örnekte tek bir architecture yapısı mevcuttur ve içindeki herbir **process**'te *sayac* değişkeni ortak kullanılabilir.

Architecture cikis **of** sayma **is**

shared variable sayac: *std_logic_vector*(3 *downto* 0) := "0011";

Begin …

5.2 Değişkenler (Variables)

Örnek 5.1: 4-bitlik ileri-geri sayıcı tasarımı gerçekleştirilecektir. Bir anahtar kullanılacak olup, anahtar değeri **lojik-0** olduğunda **geri**, **lojik-1** olduğunda ise **ileri** sayma gerçekleştirilecektir. Gerekli VHDL kodunu yazınız.

- **variable** nesneleri, önceden belirtildiği gibi ***process***, ***procedure*** ve ***function*** içinde tanımlanmalıdır.

- Yine **sinyal** tanımlaması ise **mimari** (**architecture**) içinde yapılmıştır, ancak ***process***, ***procedure*** ve ***function*** içinde tanımlanmaları mümkün değildir. **Sinyal nesneleri bu üç yapının içerisinde işletilebilirler.**

Tasarım-1

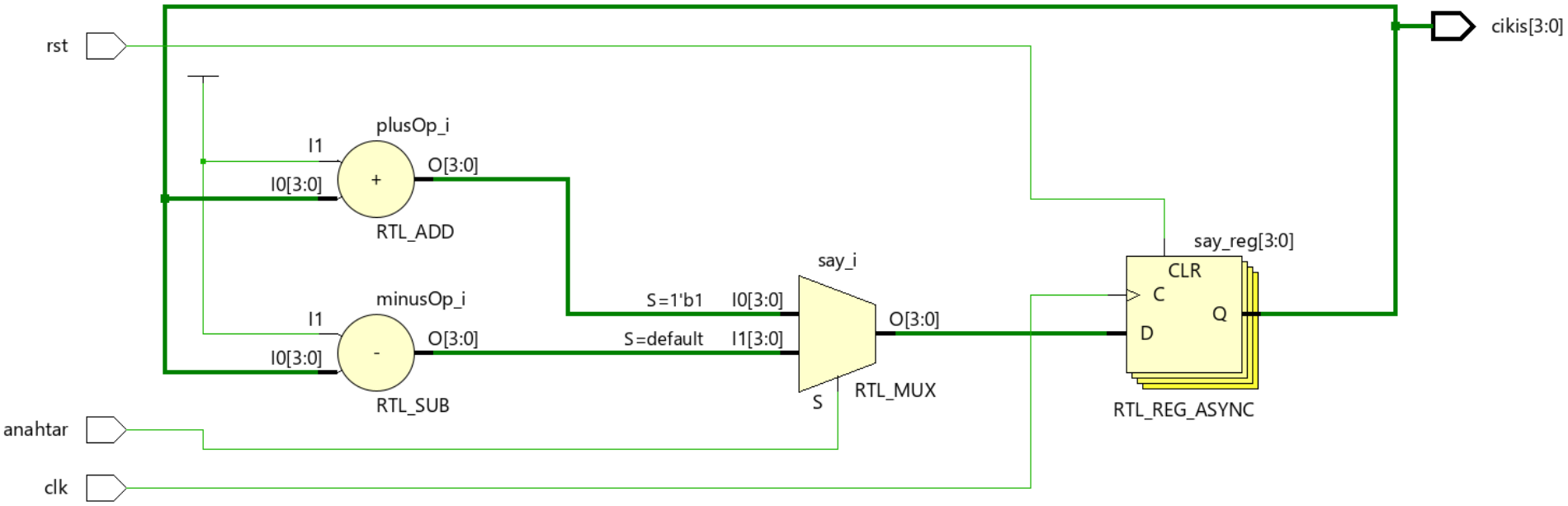
```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity ilerigerisay is
Port (clk, rst, anahtar: in std_logic;
      cikis: out std_logic_vector(3 downto 0));
End ilerigerisay;

Architecture davranis of ilerigerisay is
   signal say: unsigned(3 downto 0) := (others => '0');
Begin
  Process (clk, rst)
  Begin
    If (rst= '1') then
        say <= (others => '0');
    Elsif rising_edge(clk) then
        If (anahtar = '1') then
            say <= say+1;
        Else
            say <= say-1;
        End if;
    End if;
  End process;
cikis <= std_logic_vector(say);
End davranis;
```

5.2 Değişkenler (Variables)

Tasarım-1

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity ilerigerisay is
Port (clk, rst, anahtar: in std_logic;
        cikis: out std_logic_vector(3 downto 0));
End ilerigerisay;

Architecture davranis of ilerigerisay is
    signal say: unsigned(3 downto 0) := (others => '0');
Begin
   Process (clk, rst)
   Begin
      If (rst= '1') then
          say <= (others => '0');
      Elsif rising_edge(clk) then
          If (anahtar = '1') then
              say <= say+1;
          Else
              say <= say-1;
          End if;
      End if;
  End process;
cikis <= std_logic_vector(say);
End davranis;
```

Vivado / *Elaborated Design Çıktısı* (RTL Şematik Diyagramı)



5.2 Değişkenler (Variables)

Örnek 5.1: Aynı örneğin VHDL kodunu **say** nesnesi **variable** olduğu durum için yazalım.

- ***Nesnelerin tanımlama yerlerine ve nesnelere değer atama için gerekli komutlara dikkat edilmelidir.***

Tasarım-2

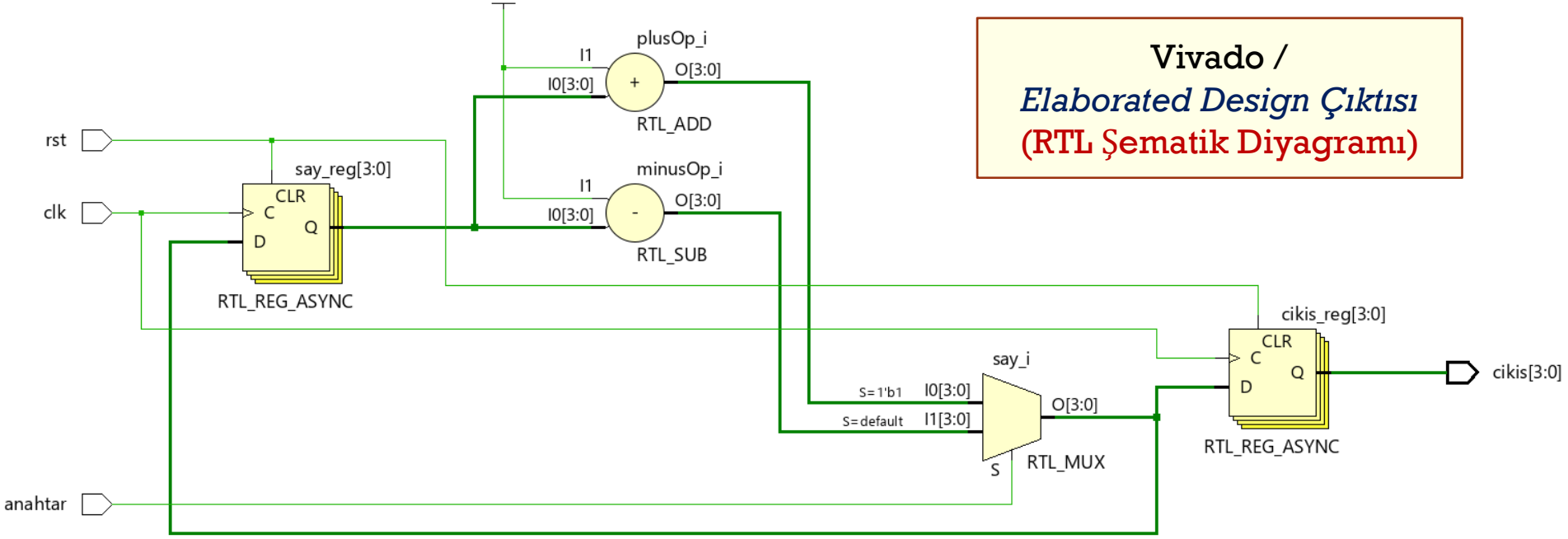
```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity ilerigerisay is
Port (clk, rst, anahtar: in std_logic;
        cikis: out std_logic_vector(3 downto 0));
End ilerigerisay;

Architecture davranis of ilerigerisay is
Begin
   Process (clk, rst)
   variable say: unsigned(3 downto 0) := (others => '0');
   Begin
      If (rst= '1') then
           say := (others => '0');
           cikis <= std_logic_vector(say);
      Elsif rising_edge(clk) then
           If (anahtar = '1') then
                say := say+1;
           Else
                say := say-1;
           End if;
           cikis <= std_logic_vector(say);
      End if;
  End process;
End davranis;
```

5.2 Değişkenler (Variables)

Tasarım-2

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity ilerigerisay is
Port (clk, rst, anahtar: in std_logic;
      cikis: out std_logic_vector(3 downto 0));
End ilerigerisay;

Architecture davranis of ilerigerisay is
Begin
   Process (clk, rst)
   variable say: unsigned(3 downto 0) := (others => '0');
   Begin
      If (rst= '1') then
          say := (others => '0');
          cikis <= std_logic_vector(say);
      Elsif rising_edge(clk) then
          If (anahtar = '1') then
              say := say+1;
          Else
              say := say-1;
          End if;
          cikis <= std_logic_vector(say);
      End if;
  End process;
End davranis;
```

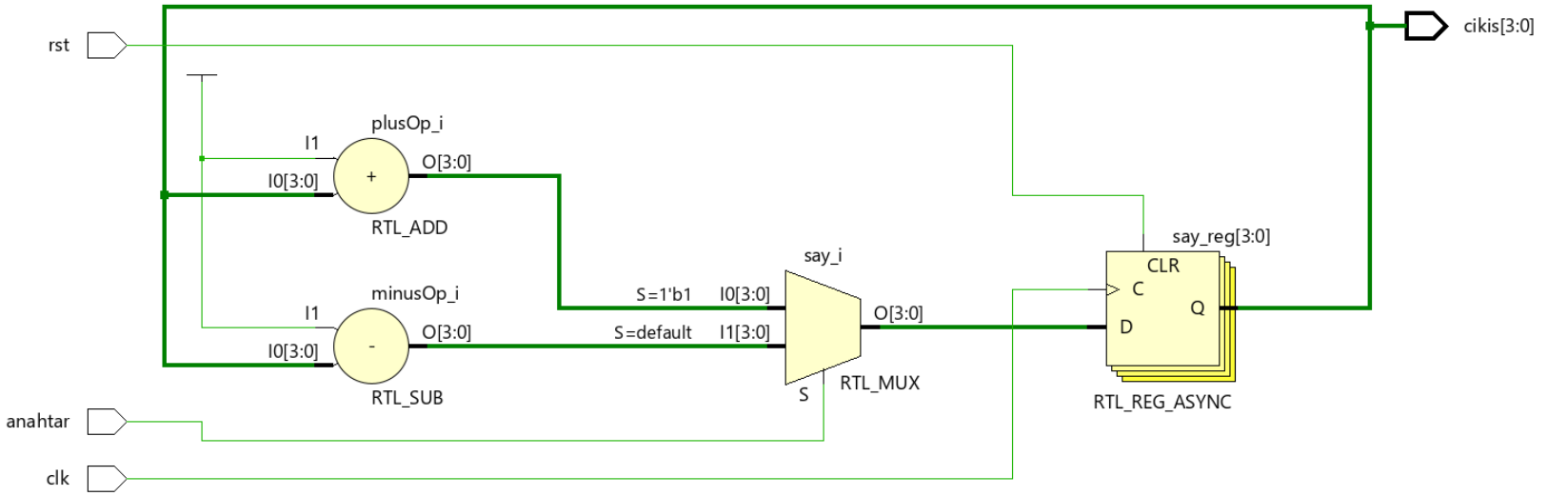
Vivado / *Elaborated Design Çıktısı* (RTL Şematik Diyagramı)



5.2 Değişkenler (Variables)

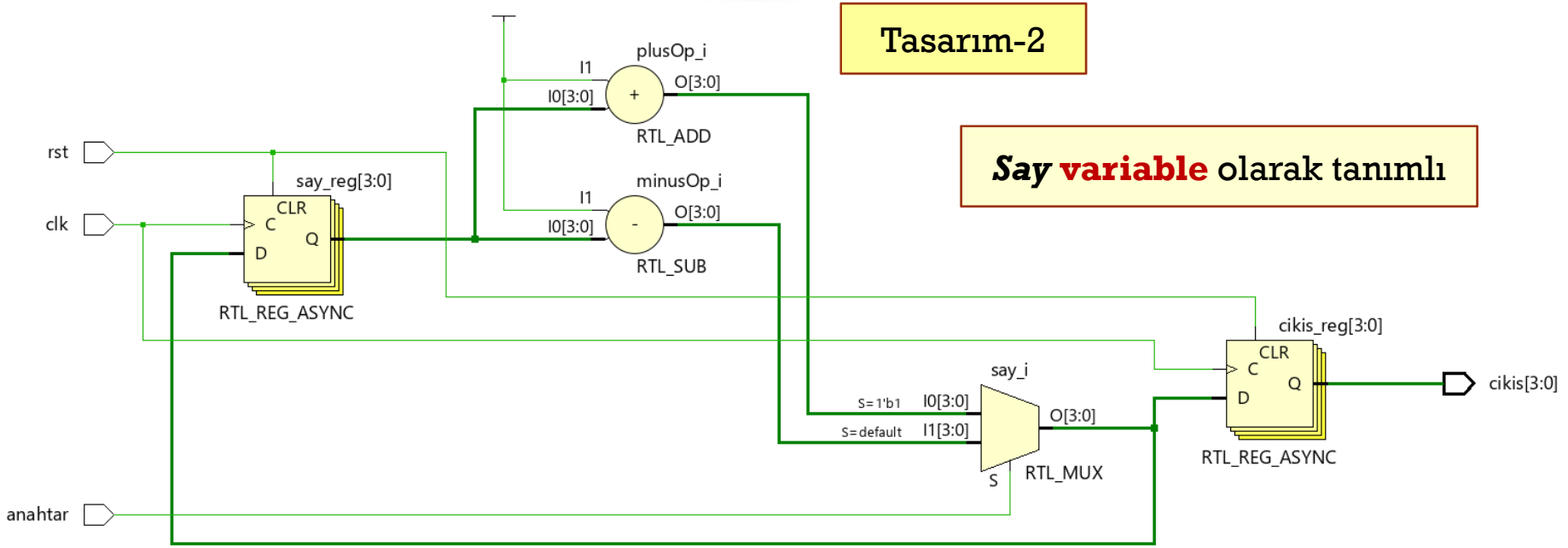
Tasarım-1

***Say* signal** olarak tanımlı



Tasarım-2

***Say* variable** olarak tanımlı



5.2 Değişkenler (Variables)

- ***Tasarım-1*** ve ***Tasarım-2*** sırasıyla **signal** ve **variable** tanımlamalarına örnek vermek amacıyla sunulmuştur.

- ***Her iki tasarımda da*** görüleceği üzere port bilgileri ***process*** içinde doğrudan işletilebilir. (***anahtar*** portunun durumuna göre işlem sağlanması, ***çıkış*** ataması, vb gibi)

Önemli Bilgi

- ***Process*** içi **variable** kullanarak çözüm önermek ***bazı durumlarda*** daha basit devre yapısı sağlayabilir.

- Ancak **sentezlenebilirlik** üzerine de **sinyal** kullanımı tavsiye edilir.

5.2 Değişkenler (Variables)

- **Çıkışa değer ataması,** ilk örnekte (**Tasarım-2**'de) **rst='1'** ve **clk** sinyalinin **yükselen kenarında** sağlanır.

- Diğer örnekte ise **If bloğu dışına** ve **process içine** yazımı sağlanmıştır. **Process** içinde işlem yapılması doğrudur (**variable process** dışında kullanılmaz, **process** dışına çıkarmak için **sinyale** aktarmak gerekir).

Tasarım-2

```
Architecture davranis of ilerigerisay is
Begin
   Process (clk, rst)
   variable say: unsigned(3 downto 0) := (others => '0');
   Begin
      If (rst= '1') then
           say := (others => '0');
           cikis <= std_logic_vector(say);
      Elsif rising_edge(clk) then
           If (anahtar = '1') then
               say := say+1;
           Else
               say := say-1;
           End if;
           cikis <= std_logic_vector(say);
      End if;
  End process;
End davranis;
```

Dikkatli kullanım gerekir

```
Architecture davranis of ilerigerisay is
Begin
   Process (clk, rst)
   variable say: unsigned(3 downto 0) := (others => '0');
   Begin
      If (rst= '1') then
           say := (others => '0');
      Elsif rising_edge(clk) then
           If (anahtar = '1') then
               say := say+1;
           Else
               say := say-1;
           End if;
      End if;
      cikis <= std_logic_vector(say);
  End process;
End davranis;
```

5.2 Değişkenler (Variables)

- Ancak **If bloğu** dışına yazım olduğu için sistemi gereksiz yükleme altında bırakır.

- **Process** bloğu duyarlık listesinde **rst** ve **clk** sinyalleri tanımlı olduğundan bu blok **rst** ve **clk** sinyallerinin her biri için **0→1** ve **1→0** değişimleri için tetiklenir.

Tasarım-2

```
Architecture davranis of ilerigerisay is
Begin
   Process (clk, rst)
   variable say: unsigned(3 downto 0) := (others => '0');
   Begin
      If (rst= '1') then
           say := (others => '0');
           cikis <= std_logic_vector(say);
      Elsif rising_edge(clk) then
           If (anahtar = '1') then
               say := say+1;
           Else
               say := say-1;
           End if;
           cikis <= std_logic_vector(say);
      End if;
  End process;
End davranis;
```

Dikkatli kullanım gerekir

```
Architecture davranis of ilerigerisay is
Begin
   Process (clk, rst)
   variable say: unsigned(3 downto 0) := (others => '0');
   Begin
      If (rst= '1') then
           say := (others => '0');
      Elsif rising_edge(clk) then
           If (anahtar = '1') then
               say := say+1;
           Else
               say := say-1;
           End if;
      End if;
      cikis <= std_logic_vector(say);
  End process;
End davranis;
```

5.2 Değişkenler (Variables)

- Eğer ***çıkış atama komutu*** **If bloğu** dışına yazılırsa, çıkışa atama işlemi **clk** sinyalinin **düşen kenarında dahi** sağlanır.

- **Bu nedenle hangi kodun nereye yazıldığı çok önemlidir.**

Tasarım-2

```
Architecture davranis of ilerigerisay is
Begin
   Process (clk, rst)
   variable say: unsigned(3 downto 0) := (others => '0');
   Begin
      If (rst= '1') then
           say := (others => '0');
           cikis <= std_logic_vector(say);
      Elsif rising_edge(clk) then
           If (anahtar = '1') then
               say := say+1;
           Else
               say := say-1;
           End if;
           cikis <= std_logic_vector(say);
      End if;
  End process;
End davranis;
```

Dikkatli kullanım gerekir

```
Architecture davranis of ilerigerisay is
Begin
   Process (clk, rst)
   variable say: unsigned(3 downto 0) := (others => '0');
   Begin
      If (rst= '1') then
           say := (others => '0');
      Elsif rising_edge(clk) then
           If (anahtar = '1') then
               say := say+1;
           Else
               say := say-1;
           End if;
      End if;
      cikis <= std_logic_vector(say);
  End process;
End davranis;
```

5.3 Sabitler (Constants)

Sabit (**Constant**), sinyal ve değişken nesnelerinin aksine değeri değiştirilemeyen nesnelerdir.

- **Sabit** nesneleri, programların okunabilirliğini artırmaktadır. **Sabit** tanımlamaları aşağıdaki kod satırı ile gerçekleştirilir.

constant isim: ***tür***:= ilkdeğer;

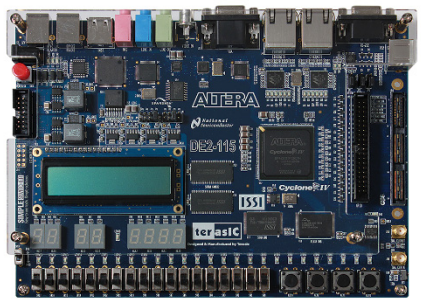
Örneğin; **constant** katsayi: *integer*:= 4;

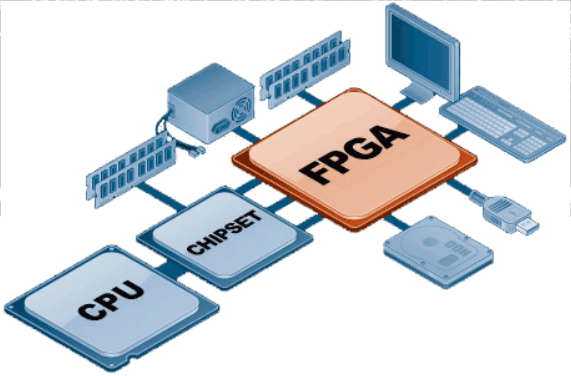
SONRAKİ DERS KONUSU

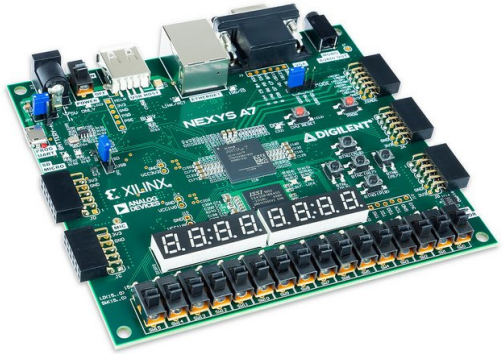
6- VHDL VERİ TÜRLERİ

İLERİ SAYISAL SİSTEMLER

6- VHDL VERİ TÜRLERİ







Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

6. VHDL VERİ TÜRLERİ

- VHDL üzerinden tasarlanan devrelerde **veriyi hafıza elemanlarında muhafaza ederek temsil eden türlerdir.**

- **Veri türleri**; **sinyal** (**signal**), **değişken** (**variable**) ve **sabit** (**constant**) olarak belirtilen **nesnelerin alabileceği değerleri tanımlamada** kullanılırlar.

- VHDL, nesnelerin belirgin bir şekilde tanımlanmalarını zorunlu kılan bir dildir ve veri **tür tanımlamaları** büyük bir önem arz eder.

- Veri türleri, bildirimleri sonrasında **program içinde değiştirilemez**.

- Tür bildirimleri, **type** ve **subtype** komutları ile gerçekleştirilir. **Type** komutu ile **yeni bir tür**, **subtype** komutu ile de **mevcut bir türe dair alt tür** tanımlanır.

6.1 Skaler Türler

- Ele alınan nesnenin **yalnızca bir veri değeri tutabildiği türlerdir.**

- **Sıralı**, **fiziksel**, **kayan nokta** ve **integer** türlerinden meydana gelir.

- Bu tür kapsamındaki tanımlamalarda **aralık** veya **sıralama** kullanılır.

Örneğin; **type** gri_seviye **is range** 0 *to* 255;

type gri_seviye **is range** 255 *downto* 0;

type sys_in **is** (gerilim, akim, sicaklik, nem);

- Değer aralığı tanımlamasında küçükten büyüğe "***to***", büyükten küçüğe ise "***downto***" komutları işletilir.

6.1.1 Sıralı Tür

- Nesnelere dair türlerin **sıralı** listeleme yöntemi kullanılarak tanımlandığı, **kullanıcıya özel** (kullanıcının tanımlaması gereken) veya **ön tanımlı** (önceden sistemde alacağı kategoriler bilinen ve yeniden tanımlanması gerekmeyen) veri türüdür.

- Bu tür, **kullanıcıya özel inceleme kapsamında** özellikle **durum makinesi modellemek için** kullanılır ve durumlara dair okunabilirliğin artırılmasını sağlar.

Tanımlama komutu: **type** nesne_ismi **is** (kategori1, kategori2, kategori3,…);

Örneğin; **type** lojik_seviye **is** (L,H);

type durum **is** (basla, dur, ileri, geri);

- Tanımlama komutundaki kategori isimleri **harf**, **alt çizgi** ve **numaralardan** oluşur.

6.1.1 Sıralı Tür

- Kategori isminin **ilk elemanı harf ile başlatılmalıdır**.

- Değişken ve sinyallere atama yapılırken, **atanacak değer** ve **değerin atanacağı nesnenin aynı türde olması gereklidir**.

Örneğin; **type** lojik_seviye **is** (L,H);

type durum **is** (basla, dur, ileri, geri);

Sıralı tür atama örneği

```
type doluluk is (bos, ceyrek, yarim, yariceyrek, tam);
type lojik_seviye is ('0', '1');
variable doluluk_kontrol: doluluk;
signal seviye_kontrol: lojik_seviye;
…
doluluk_kontrol := ceyrek;
seviye_kontrol <= '1';
```

6.1.1 Sıralı Tür

Boolean Türü: **Ön tanımlı** türlerden birisi olup, ***true*** veya ***false*** değerlerinden birini alır.

Ön tanım komutu: **type** boolean **is** (true, false);

Bit Türü: '0' ve '1' değerlerini alabilen **ön tanımlı** türdür.

- **Integer (tam sayı)** türü ile karıştırılmaması için **tırnak işareti içinde tanımlanır.**

Ön tanım komutu: **type** bit **is** ('0', '1');

Örneğin; **signal** deger: *bit* := '1'; // **bildirim ve ilk değer ataması**

…

deger <= '0'; // **kod bloğu içinde değer ataması**

6.1.1 Sıralı Tür

Character Türü: ISO-8859-1 karakter setindeki 256 farklı karakteri içerir.

Sıralı Tür İçin Önemli Bilgi: Sıralı tür içerisinde bir diğer önemli konu ise **aşırı yüklemedir**.

- Bir kategori birden fazla sıralı tür içerisinde kullanıldığında aşırı yükleme durumu söz konusu olabilir.

- Herhangi bir anlam kargaşası oluşmaması için program içinde kullanılan sıralı türün ***hangisi olduğu açık bir şekilde belirtilmelidir***.

Örneğin; **type** girdiler **is** (**gerilim**, **akim**, ruzgar, nem, sicaklik);

type ayarlanacak **is** (**gerilim**, **akim**, katsayi);

…

deg <= girdiler'(**akim**);

6.1.2 Tamsayı Türü

- Alabileceği değerler kullanıcı tarafından belirlenen aralıktaki tamsayıları kapsayan **skaler** türdür.

Tanımlama komutu: **type** isim **is range** alt_limit *to* ust_limit;

type isim **is range** ust_limit *downto* alt_limit;

Örneğin; **type** tam_sayi **is range** -1000 *to* 3000;

subtype pozitifler **is** *tam_sayi* **range** 1 *to* 3000;

Integer Türü: Standard paketinde bulunan **ön tanımlı tamsayı türüdür**. Tanımlı olduğu aralık ise -2.147.483.647 ile 2.147.483.647 arasıdır.

Ön tanım komutu: **type** integer **is range** -2147483647 *to* 2147483647;

Örneğin; **signal** sayma: *integer*;

…

sayma <= 657;

6.1.3 Fiziksel Tür

- Uzunluk, ağırlık, sıcaklık gibi ölçülebilen fiziksel özellikler **nicelik** olarak bilinir.

- Fiziksel tür, **nesnelerin nicel tanımlamalarını birim baz üzerinden yapabilmek için** kullanılan **skaler** türdür.

- Nicel olarak sayısal oranlarla kendi içinde dönüştürülebilen ve ölçülebilen birimleri ifade etmek için kullanılır.

- Fiziksel türe dair en küçük birim **temel birim** olarak adlandırılır.

- Türün bildirimi sırasında **range** komutu işletilir ve nesneye dair sınır değerler (en yüksek ve en düşük) tanımlanır.

6.1.3 Fiziksel Tür

Tanımlama komutu: **type** tur_ismi **is range** alt_limit *to* ust_limit

units

temel_birim_ismi;

ikincil_birim_ismi = **deger** temel_birim_ismi;

ucuncul_birim_ismi = **deger** ikincil_birim_ismi;

…

end units tur_ismi;

- Fiziksel türler yalnızca **simülasyonda** kullanılır.

- Fiziksel türde tanımlı bir nesne üzerinde **aritmetiksel ve mantıksal işlemler yapılabilir**, ancak bu türler **sentezde kullanılamaz**.

6.1.3 Fiziksel Tür

Örneğin; **type** uz_deg **is range** 0 *to* 1E9

units

mm;

cm = 10 mm;

dm = 10 cm;

m = 10 dm;

km = 1000 m;

end units uz_deg;

variable sayi1: *uz_deg*;

variable sayi2: *integer*;

…

sayi1 := 2 mm + 5 dm + 3 m;

sayi2 := 1250;

6.1.3 Fiziksel Tür

Time Türü: Standard paketinde bulunan **ön tanımlı** fiziksel türdür.

- Tanımlı olduğu aralık ise -2.147.483.647 ile 2.147.483.647 arası olup, temel birim **femtosaniye** (saniyenin 10^{-15}'i)'dir.

Ön Tanım komutu:

```
type time is range -2147483647 to 2147483647
  units
    fs;                 --femtosecond
    ps   = 1000 fs;     --picosecond
    ns   = 1000 ps;     --nanosecond
    us   = 1000 ns;     --microsecond
    ms   = 1000 us;     --milisecond
    sec  = 1000 ms;     -- second
    min  = 60 sec;      --minute
    hr   = 60 min;      --hour
  end units;
```

Örneğin; **constant** periyot_deg: *time* := 5ns;

6.1.4 Kayan Nokta Türü

- Özel olarak **belirli sınırlarda tanımlanan** ve **gerçek sayılardan oluşan** fiziksel türdür.

- Matematiksel işlemlerde (toplama, çıkarma, üs alma, vb. gibi) kullanılabilirler.

Tanımlama komutu:

type tur_ismi **is range** ust_sinir *downto* alt_sinir;

type tur_ismi **is range** alt_sinir *to* ust_sinir;

Örneğin; **type** akim_degeri **is range** -3.7 *to* 6.5;

type gerilim_degeri **is range** -360.87 *to* 520.35;

6.1.4 Kayan Nokta Türü

Real Türü: Standard paketinde bulunan **ön tanımlı** kayan nokta türü olup, gerçek sayıların tanımlanacağı nesneler için kullanılır.

Ön tanım komutu: **type** real **is range** -1.7014111e+308 *to* 1.7014111e+308;

Örneğin; **constant** katsayi: *real* := 5.785;

6.2 Kompozit Tür

- Bir değer yığınını gösteren veri türüdür.
- Bu türde tanımlı nesneler **birden fazla unsura sahiptir**.
- **Array** ve **record** olmak üzere **iki kompozit tür** bulunur.

6.2.1 Array (Dizi) Türü

- Aynı türdeki **bir veya daha fazla elemanı gruplayıp bir nesne olarak tanımlayan** veri türüdür.

- Bir dizideki her elemanın, belli aralık içinde bir indeks numarası bulunur.

- Tek boyutlu dizilerde **bir indeks numarası** bulunurken, çok boyutlu dizide **dizi boyutu kadar sayıda indeks numarası** kullanılır.

- Çok boyutlu diziler genelde sentezlenemez.

Tanımlama komutları:

type tur_ismi **is array** (aralık) **of** *dizi_turu*;→ (Kısıtlamalı Dizi)

type tur_ismi **is array** (alt_tur **range** < >) **of** *dizi_turu*; → (Kısıtlamasız Dizi)

6.2.1 Array (Dizi) Türü

Örneğin;

Yeni tür ismi

2^3 × 4 Ram / Rom için tür tanımı

```
type ram_turu is array (0 to 7) of std_logic_vector (3 downto 0);
```

- Diziler aralığa bağlı olarak **kısıtlı** veya **kısıtsız** olabilmektedir.

- Değer aralığı açık bırakılan kısıtsız dizilerdeki eleman sayısı da **belirsiz** olacaktır.

- Kısıtlamasız olarak bildirilen bir diziye dair boyutun, kod bloğu içinde **mutlaka bildirilmesi gerekir**.

- Bu türdeki sinyal ve değişkenler ancak **bu şekilde program kapsamında kullanılabilir**.

6.2.1 Array (Dizi) Türü

Örnek 6.1: 4'er bitten oluşan ve kısıtlanmamış bir bellek tanımlaması için gerekli VHDL kodunu yazınız.

type bellek **is array** (*natural* **range** < >) **of** *bit_vector* (0 *to* 3);

- Sınırsız oluşturulan bellek kapasitesi aşağıdaki kod bloğu ile belirlenebilir.

signal ram_bellegi: *bellek* (0 *to* 5);

	0	1	2	3
0				
1				
2				
⋮	⋮	⋮	⋮	⋮
n				

6.2.1 Array (Dizi) Türü

- Array tipinde tanımlı dizi elemanlarının her birisine bağımsız olarak değer atanabilmektedir.

- Dizilerde sinyal–değişken arası atama yapılırken, bunların **tür ve genişliklerinin aynı olması gereklidir**.

Örneğin; Tam sayı türünde 10 elemanlı dizi için gerekli tür tanımlaması:

type int_vekt **is array** (1 *to* 10) **of** *integer*;

Örneğin; Bit türünde 5 elemanlı dizi için gerekli tür tanımlaması:

type bit_dizisi **is array** (0 *to* 4) **of** *bit*;

6.2.1 Array (Dizi) Türü

Örneğin; Kısıtlamasız bir **bit vektör** dizisinin tanımlanması ve kod içinde bu türde 8 bitlik bir nesne tanımlaması:

type bit_vektoru **is array** (*natural* **range** < >) **of** *bit*;

…

signal nesne1: *bit_vektoru* (7 *downto* 0);

- Burada kod bloğu içinde kısıtlamasız tanımlanan **bit_vektoru** isimli dizinin boyutu, program içinde 8 ile sınırlandırılarak kullanılıyor.

6.2.1 Array (Dizi) Türü

Örnek 6.2: 3x6 boyutlu, **integer** değerleri tutacak olan bir matrisde 1x5 konumuna 91 değerini atayınız.

```
type tams_vek is array (0 to 5) of integer;
type matris_tip is array (0 to 2) of tams_vek;
signal matris_ilk: matris_tip;
…
matris_ilk(1)(5) <= 91;
```

	0	1	2	3	4	5
0						
1						91
2						

Örnek 6.2' de bir diğer kod tanımlaması:

```
type matris_tip is array (0 to 2, 0 to 5) of integer;
signal matris_ilk: matris_tip;
…
matris_ilk(1)(5) <= 91;
```

6.2.1 Array (Dizi) Türü

Bit_vector Türü: **Ön tanımlı** array türü olup, elemanlarının her biri ‘0’ ve ‘1’ lerden oluşur.

- Bildirim aşamasında dizinin genişliği belirtilir. **Bit_vector** türünde tanımlı nesneler için **vektör** değerleri atanacak ise **çift tırnak**, **tekil** bir bit değeri atanacaksa **tek tırnak** kullanılır.

Ön Tanım Komutu: **type** bit_vector **is array** (*natural* **range** < >) **of** *bit*;

Örneğin;

```
signal vektor1: bit_vector(3 downto 0);
signal vektor2: bit_vector(7 downto 0);
signal tekil_deg: bit;
…
vektor1 <= "1011";
vektor1(3) <= '0';
tekil_deg <= '0';
vektor2 <= "10101111";
vektor2 <= x"AF";
```

6.2.1 Array (Dizi) Türü

String Türü: **Ön tanımlı** array türü olup, karakterlerden oluşan dizilerdir.

- Bildirim aşamasında dizinin boyutu ve sıralaması belirtilir.
- Bu türdeki değerler **çift tırnak içinde** tanımlanır.
- Sentezlenemeyen bir türdür ve simülasyon içinde mesaj yayınlamak için kullanılır.

Ön Tanım Komutu: **type** string **is array** (*positive* **range** < >) **of** *character*;

Örneğin;

```
constant eleman: string := "FPGA";
constant dil: string(1 to 4) := "VHDL";
signal mesaj2: string(1 to 8);
…
mesaj2 <= eleman & dil;
```

6.2.2 Record Türü

- Kapsamındaki elemanların **farklı türdeki nesneleri** içerdiği **kompozit** türdür.

- Tanımlama komutları: **type** record_ismi **is record**
  eleman_ismi: *tur*;
  eleman_ismi: *tur*;
  …
  end record;

Örneğin;

type not_kaydi **is record**
  ogr_not: *integer*;
  ogr_isim: *string* (1 to 40);
end record;

6.3 Synopsis Veri Türleri

- IEEE kütüphanesinde tanımlanmış olan veri türleridir.

- En sık kullanılanları **Std_Logic_1164** ve **Std_Logic_Arith** paketlerinde tanımlı türlerdir.

6.3.1 Std_Logic_1164 Paketinde Tanımlı Türler

- Bu kapsamda yer alan türlere dair bildirimler, program başlangıcında aşağıdaki tanımlama yapıldıktan sonra sağlanır:

Library *ieee*;

Use *ieee.std_logic_1164.all*;

6.3.1 Std_Logic_1164 Paketinde Tanımlı Türler

Std_logic: VHDL dilinde **en sık kullanılan veri türüdür**. Bu tür kapsamında 9 farklı durum değer olarak atanabilmektedir. Bu türler:

- 1 → Mantıksal – 1
- 0 → Mantıksal – 0
- H → Zayıf – 1
- L → Zayıf – 0
- X → Bilinmeyen
- U → Tanımlanmamış
- - → Önemsiz
- W → Zayıf bilinmeyen
- Z → Yüksek Empedans şeklindedir.

- Eğer bu kapsamda tanımlanan bir veri için **başlangıç değeri atanmamışsa**, varsayılan değer **"U – Tanımlanmamış"** şeklinde atanır.

- **Std_logic** türündeki nesnelere atamalar **tek tırnak** içinde yapılır.

6.3.1 Std_Logic_1164 Paketinde Tanımlı Türler

Örneğin;

signal ekg1: *std_logic*; // ilk atama yapılmamış, varsayılan değer 'U' olarak bilinir.

signal ekg2: *std_logic* := '1'; // ilk atama yapılmış ve değer mantıksal-1 (yani lojik-1).

signal ekg3: *std_logic* := '-'; // ilk atama yapılmış ve değer önemsiz.

Std_logic_vector: Elemanlarının her birisi **std_logic** türünde olan dizi türüdür. Bağlı bulunan nesne değerleri **çift tırnak** içinde ifade edilir.

Örneğin;

signal parcacik: *std_logic_vector* (3 *downto* 0);

signal durum1: *std_logic_vector* (2 *to* 5) := "1010";

signal durum2: *std_logic_vector* (7 *downto* 0) := x"FF";

6.3.2 Std_Logic_Arith Paketinde Tanımlı Türler

- Bu kapsamda yer alan türlere dair bildirimler, program başlangıcında aşağıdaki tanımlama yapıldıktan sonra sağlanır:

Library *ieee*;

Use *ieee.std_logic_arith.all*;

Unsigned: İşaretsiz sayıları temsil etmek için tanımlanır.

- En küçük değeri "0" olup, **negatif sayıları içermez**.

- Bu türde tanımlanacak nesne bildirimlerinde **değer aralığı bildirilmelidir**.

- Nesneye atama yapılırken **nesne içi bir elemana atamada tek tırnak**, **birden çok eleman atamasında** ise **çift tırnak** kullanılır.

6.3.2 Std_Logic_Arith Paketinde Tanımlı Türler

Örneğin;

```
signal in_deg: unsigned (0 downto 0);
…
in_deg <= '0';
in_deg(0) <= '1';
```

Örneğin;

```
variable degisken: unsigned (0 to 7) := "11111111";
…
degisken(1 to 3) := "000";
degisken(2) := '1';
```

6.3.2 Std_Logic_Arith Paketinde Tanımlı Türler

Signed: **İşaretli sayı** değerlerini temsil etmek için tanımlanır.

- En sol kısımda yer alan bit **işaret biti** olarak nitelenir, **'0'** olması halinde sayı **pozitif** ve **'1'** olması halinde sayı **negatiftir**.

- Negatif olan sayılar ikiye tümleme mantığı ile yazılmalıdır.

- Yine atamalarda değer aralığı belirtilmeli, **çoklu eleman** atamalarında **çift tırnak** ve **tekli eleman** atamalarında **tek tırnak** kullanılmalıdır.

- "n" bitlik **signed** türünde bir nesne için alabileceği değerler, $-(2^{n-1})$ ile $(2^{n-1}-1)$ arası sayısal değer olacaktır.

signed ("0011") // (+3)

signed ("1101") // (-3)

6.3.2 Std_Logic_Arith Paketinde Tanımlı Türler

Örneğin;

variable deg_in: *signed* (1 *to* 5);

…

deg_in := "11100";

deg_in(4) := '1';

deg_in(1 *to* 2) := "00";

6.4 Alt Türler

- VHDL dilinde mevcut olan **bir türün belirli bir kısıtla sunulan hali**, **alt tür** olarak tanımlanır.

- Alt tür, mevcut bir türün alt kümesidir.

6.4 Alt Türler

- Standard paketleri kapsamında en sık kullanılan iki ön tanımlı alt tür, **Natural** ve **Positive** türleridir.

Tanımlama komutu: **subtype** alttur_ismi **is** temel_tur **range** aralık_kisiti;

Örneğin; **type** tam_sayi **is range** -20 *to* 20;

subtype poz_tam_sayi **is** tam_sayi **range** 1 *to* 20;

Natural: Kapsamında **"0"** ve **pozitif sayıların** yer aldığı tür olup, **integer** türünün alt türüdür.

Ön Tanım Komutu: **subtype** natural **is** *integer* **range** 0 *to* integer'high;

Örneğin; **signal** deger: *natural*;

...

deger <= 0;

deger <= 5850;

6.4 Alt Türler

Positive: Kapsamında **pozitif sayıların** yer aldığı tür olup, **integer** türünün alt türüdür.

Ön Tanım Komutu: **subtype** positive **is** *integer* **range** 1 *to* integer'high;

Örneğin; **signal** deger: *positive*;

```
…
deger <= 1;
deger <= 9523;
```

6.5 Dizi İşlemleri

- VHDL nesneleri, kendilerine değer ataması yapılarak veya birbirleri ile işleme tabi tutularak kullanılır.

- Dizilerde de karşılaştırma, atama, ekleme gibi işlemler gerçekleştirilir.

6.5 Dizi İşlemleri

- Birden fazla elemana sahip dizilerde bu işlemlerin sağlanabilmesi için 4 farklı yardımcı işlem bulunur:

1-Birleştirme

2-Kümeleme

3-Dilimleme

4-Takma ad

Birleştirme (&): Aynı türdeki iki dizinin **'&' (ampersand)** işareti kullanılarak birleştirilmesi sağlanır.

- Bu operatör (&), sinyal atama operatörünün (<=) yalnızca **sağ tarafında** işletilir.

- Bu operatörle dizilere **tek bir eleman eklenebileceği gibi**, **aynı türde farklı iki eleman da birbirine bağlanabilir**.

6.5 Dizi İşlemleri

Örnek 6.3: Dizi birleştirme örnekleri

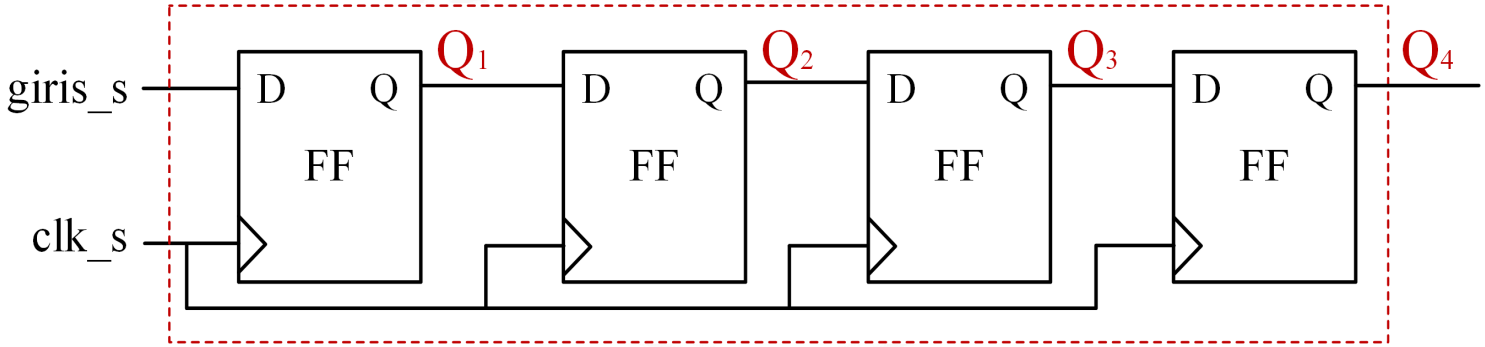
```
elde_1 <= B & C;
elde_2 <= A & D;
elde_3 <= A & B;
elde_4 <= A & B & C;
```

A	B	C	D
01	111	1011	1111

elde_1	elde_2	elde_3	elde_4
111 1011	01 1111	01 111	01 111 1011

6.5 Dizi İşlemleri

Örnek 6.4: Öteleyici register tasarımı için gerekli **architecture** kodunu **birleştirme operatörü kullanarak** sağlayınız.




```
Architecture davranis of oteleyici_reg is
   signal Q: std_logic_vector(3 downto 0) := "0000";
Begin
   Process (clk_s)
   Begin
      If (clk_s'event) and clk_s = '1' then
           Q <= Q(2 downto 0) & giris_s;
      End if;
  End process;
End davranis;
```

6.5 Dizi İşlemleri

Kümeleme: **Record** ve **array** türündeki nesnelere değer atanırken kullanılır.

- Kümeleme yönteminde dizideki **herbir elemana teker teker değer atamak yerine**, **bütün elemanlara aynı anda farklı değerler** atanabilir.

- Nesnelere atama yapılırken kümeleme kullanılırsa, parantez içinde **her bildirim arası virgül kullanılır**.

- Kümeleme işleminde ***dizi elemanları indeks numarasına göre adreslenebilmektedir.***

- Adreslenmeyen elemanlar için **"others"** terimi üzerinden adresleme yapılabilir ve bu terim bildirilirken **parantez içi sonda yer alır**.

6.5 Dizi İşlemleri

Dizi Türü İçin Örnekler;

variable vektor1: *std_logic_vector* (3 *downto* 0) := ('1', '0', '1', '1');

- vektor1 nesnesine dair ilk değer ataması **1011** şeklinde olur.

variable vektor2: *std_logic_vector* (3 *downto* 0) := ('1', '1',others=>'0');

- vektor2 nesnesine dair ilk değer ataması **1100** şeklinde olur.

variable vektor3: *bit_vector* (3 *downto* 0) := (0=>'1', 1=>'1', 2=>'0', 3=>'0');

- vektor3 nesnesine dair ilk değer ataması **0011** şeklinde olur.

signal veri: *bit_vector* (7 *downto* 0);

…

veri <= (7 downto 4 => '0', 3 downto 0 => '1');

- veri nesnesine değer ataması **00001111** şeklinde olur.

6.5 Dizi İşlemleri

Record Türü İçin Örnek;

```
type bilgi is record
    sayi_deg: integer;
    eleman_bil: string (1 to 4);
end record;
variable yaz: bilgi := (sayi_deg=>5, eleman_bil=> "ASIC");
…
yaz := (10, "FPGA");
```

Dilimleme: Bir diziyi parçalara ayırmak amacıyla kullanılır. Ayrılan herbir parça **dizi dilimi** olarak nitelenir.

- Herbir dizi diliminin **adı**, **içindeki elemanlara dair veri türü** ve **yönü** çıktığı **(ana) dizidekiyle aynıdır**.

- Örneğin ana dizideki veri yönü ***downto*** ile belirli ise **dilimlerde de aynı durum olmalıdır.**

6.5 Dizi İşlemleri

Örneğin; **signal** bilgi: *std_logic_vector* (0 *to* 15);

...

bilgi(0 *to* 3) -- ilk dilim: bilgi (ana) dizisinin **ilk 4 elemanını** içerir

bilgi(4 *to* 7) -- 2. dilim: bilgi dizisindeki **ikinci 4'lüyü** içerir

bilgi(8 *to* 8) -- 3. dilim: bilgi dizisinde **9. elemanı** içerir

bilgi(9 *to* 15) -- 4. dilim: bilgi dizisindeki **son 7 elemanı** içerir

---

Takma Ad: Varolan bir nesnedeki bilgileri, ***herbiri farklı isimle belirtilmiş farklı gruplara ayırmak için*** işletilir.

- Büyük verilerin parçalanarak daha kolay işlenmesini sağlar.

Tanımlama komutu:

alias takma_isim: *takma_isim_veri_turu* **is** nesne_ismi;

6.5 Dizi İşlemleri

Örneğin kod bloğu;

signal bilgi: *std_logic_vector*(7 *downto* 0);

alias alt_bilgi1: *std_logic_vector*(2 *downto* 0) **is** bilgi(7 *downto* 5);

alias alt_bilgi2: *std_logic* **is** bilgi(4);

alias alt_bilgi3: *std_logic_vector*(3 *downto* 0) **is** bilgi(3 *downto* 0);

alias takma_isim: *takma_isim_veri_turu* **is** nesne_ismi;

bilgi								
alt_bilgi1			alt_bilgi2			alt_bilgi3		
7.Bit	6.Bit	5.Bit		4.Bit		3.Bit	2.Bit	1.Bit	0.Bit

6.6 Nitelik (Attribute)

- **Attribute**, sinyal veya değişken nesnelere dair ***özellikleri döndürmek için*** kullanılır.

- Ön tanımlı olan nitelik türleri, VHDL standardı olarak bütün sentezleme araçları tarafından desteklenir.

- **Left**: Vektör olarak ifade edilen nesnenin **solundaki** indisi gösterir.

- **Right**: Vektör olarak ifade edilen nesnenin **sağındaki** indisi gösterir.

- **High**: Vektör olarak ifade edilen nesnenin **en büyük** indisi gösterir.

- **Low**: Vektör olarak ifade edilen nesnenin **en küçük** indisi gösterir.

- **Length**: Vektör olarak ifade edilen nesnenin **uzunluğunu** gösterir.

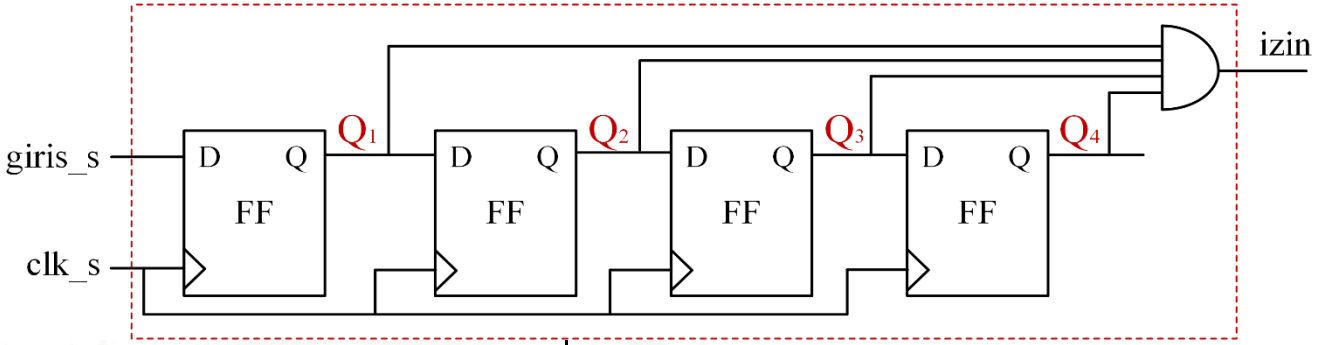
- **Range**: Vektör olarak ifade edilen nesnenin **indis aralığını** gösterir.

6.6 Nitelik (Attribute)

Örneğin; **signal** vektor: *std_logic_vector* (2 *to* 8);

Nitelik İfadesi	Elde Edilecek Değer
vektor'left	2
vektor'right	8
vektor'high	8
vektor'low	2
vektor'length	7
vektor'range	2 *to* 8

6.6 Nitelik (Attribute)



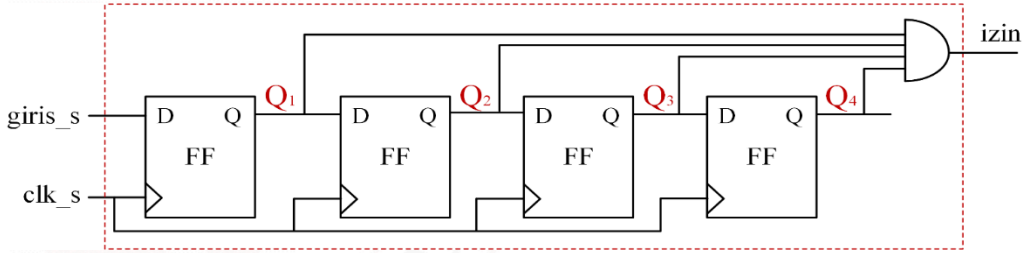

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity dizi_yakala is
Port (clk_s, giris_s: in std_logic;
        izin: out std_logic);
End dizi_yakala;

Architecture davranis of dizi_yakala is
   signal Q: std_logic_vector (3 downto 0) := "0000";
Begin
   Process (clk_s)
   Begin
      If rising_edge(clk_s) then
           Q <= Q(2 downto 0) & giris_s;
      End if;
  End process;
  izin <= Q(3) and Q(2) and Q(1) and (not Q(0));
End davranis;
```

Örnek 6.5: Öteleyici register üzerinden 1110 dizisini yakalayan devrenin tasarımı için gerekli VHDL kodunu yazınız.

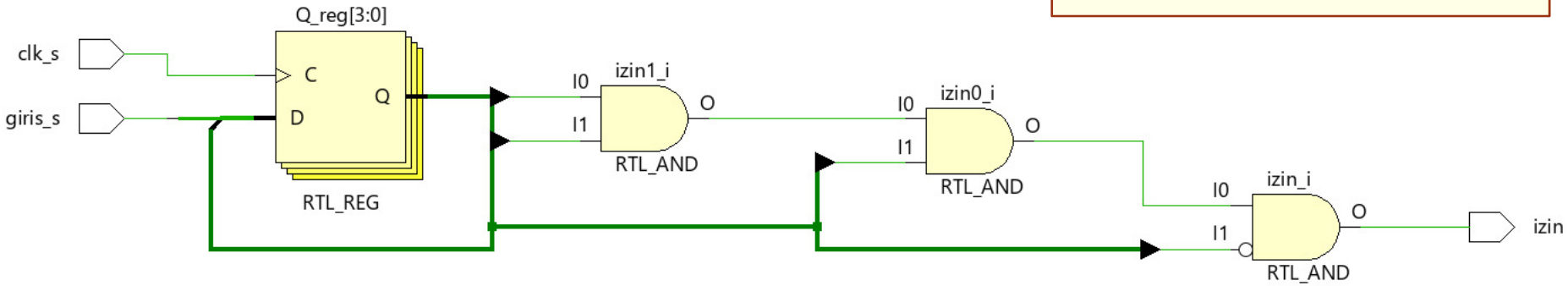
6.6 Nitelik (Attribute)

Örnek 6.5: Öteleyici register üzerinden 1110 dizisini yakalayan devrenin tasarımı için gerekli VHDL kodunu yazınız.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity dizi_yakala is
Port (clk_s, giris_s: in std_logic;
        izin: out std_logic);
End dizi_yakala;

Architecture davranis of dizi_yakala is
   signal Q: std_logic_vector (3 downto 0) := "0000";
Begin
    Process (clk_s)
    Begin
       If rising_edge(clk_s) then
            Q <= Q(2 downto 0) & giris_s;
       End if;
   End process;
   izin <= Q(3) and Q(2) and Q(1) and (not Q(0));
End davranis;
```



Vivado / *Elaborated Design Çıktısı* (RTL Şematik Diyagramı)

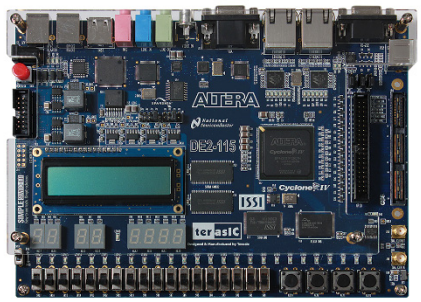


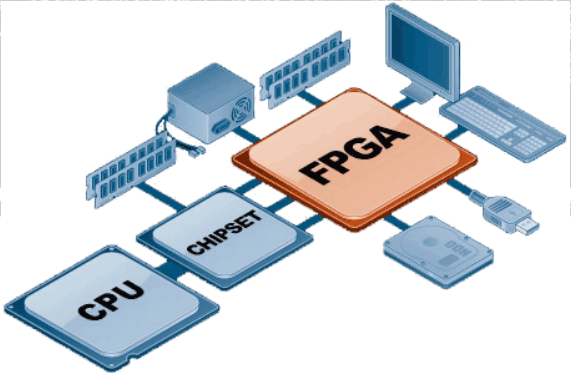
SONRAKİ DERS KONUSU

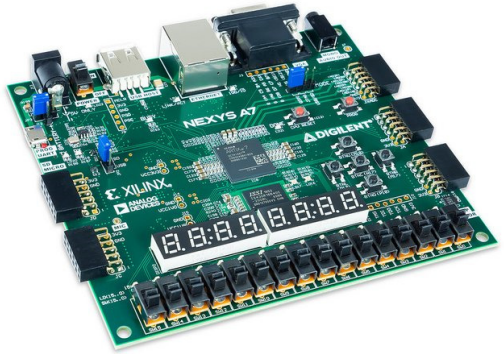
7- VHDL OPERATÖRLERİ

İLERİ SAYISAL SİSTEMLER

7- VHDL OPERATÖRLERİ







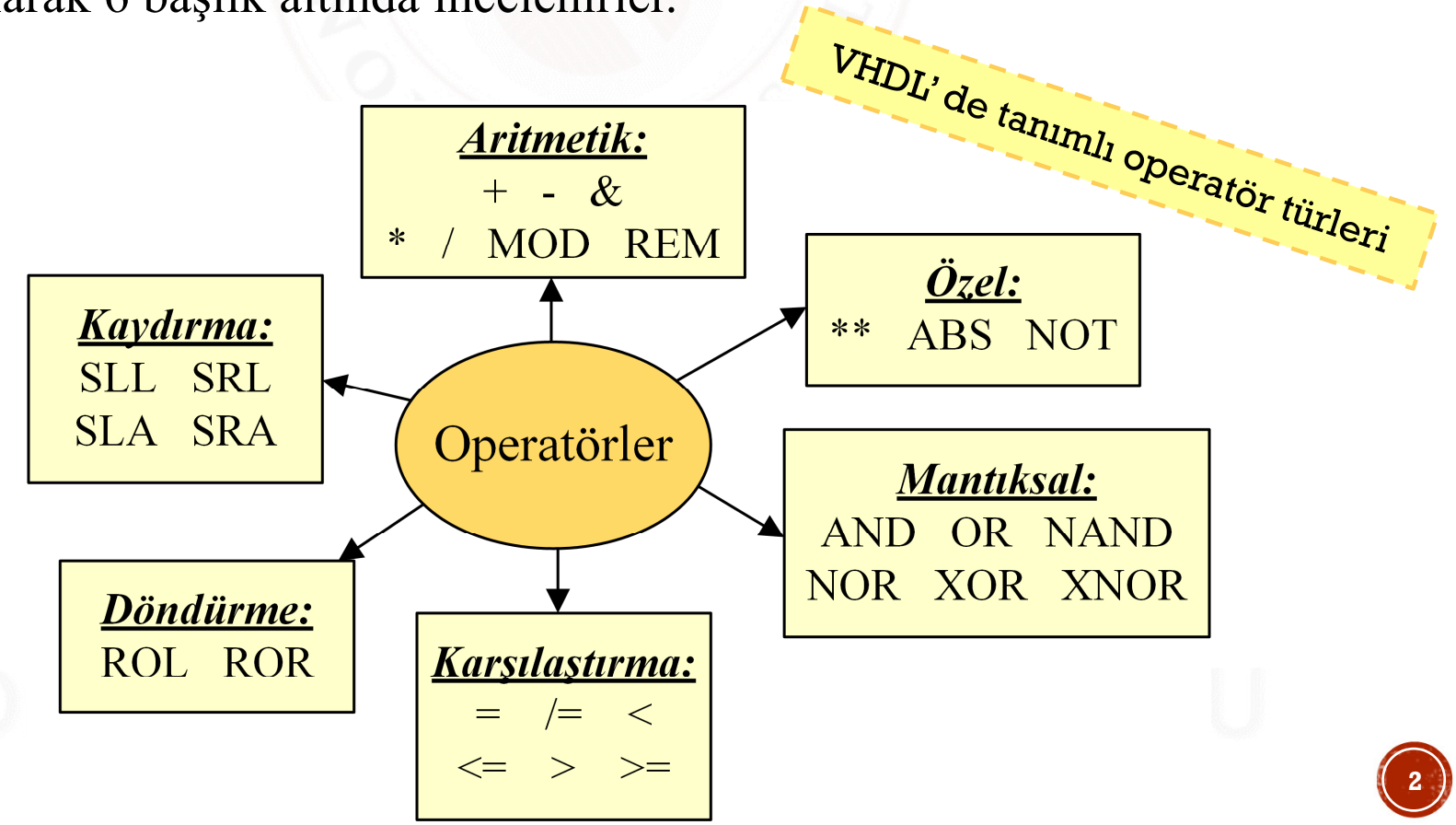
Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

7. VHDL OPERATÖRLERİ

- VHDL ifadeleri arasında kullanımı gerekli olan aritmetik, karşılaştırma ve mantıksal işlemlerin sağlanabilmesi için **operatörler** kullanılır.

- Genel olarak 6 başlık altında incelenirler.



7. VHDL OPERATÖRLERİ

- VHDL operatörlerinin kullanım önceliği tabloda **yukarıdan aşağıya doğru en önemliden başlayarak** sunulmuştur.

- **Aynı sınıfta yer alan operatörler aynı önceliğe sahiptirler** ve **kod içinde soldan sağa doğru öncelikte işlem yapılır**.

- Yine kod içinde parantez kullanılarak öncelik sağlanabilir.

VHDL operatörlerinin özellikleri

Operatör Türü		Operatör İsmi / Sembolü	Öncelik Sırası
	Özel Operatörler	**, NOT, ABS	
Aritmetik	Ulama / Toplama / Çıkarma	&, +, -	
	Çarpma / Bölme	*, /, MOD, REM	
	Kaydırma	SLL, SRL, SLA, SRA	
	Döndürme	ROL, ROR	
	Karşılaştırma	=, /=, <, <=, >, >=	
	Mantık	AND, OR, NAND, NOR, XOR, XNOR	

7.1 Aritmetik Operatörler

- Matematikte kullanılan bütün aritmetik operatör sembolleri ve bu işlemlerin öncelik sırası VHDL dilinde de aynı şekilde kullanılır.

- **Aritmetik işlemler** yalnızca **aynı türdeki elemanlar arası** gerçekleştirilebilir.

- Bu operatörler **bit_vector** dışında **bütün vektör ve sayı türleri için kullanılabilirler.**

Aritmetik operatörlere dair işaret ve işlem ilişkisi

İşaret	İşlem	Örnek
+	Toplama	4+2=6
-	Çıkarma	8-3=5
*	Çarpma	3*4=12
/	Bölme	7/2=3
MOD	Kalan	5 MOD 2 = 1 8 MOD 5 = 3
REM	Kalan	5 REM 2 = 1 (-5) REM 3 = -2

7.1 Aritmetik Operatörler

- Dizilerde aritmetik işlem yapılırken **işleme girecek dizilere dair boyutların eşit olması gereklidir.** Böylece dizilerin bütün elemanları karşılıklı olarak işleme girer.

- **MOD** operatöründe sonucun işareti "**bölen**" sayının işaretiyle aynı olacaktır. **REM** işleminde ise sonuç işareti "**bölünenin**" işareti ile aynı olur.

- REM ve MOD operatörleri, **bir tamsayının başka bir tamsayıya bölümünden kalanı** tespit etmek için kullanılır.

REM ve MOD bölme örnekleri

REM		MOD	
İşlem	**Açıklama**	**İşlem**	**Açıklama**
5 rem 3 = 2	1*3 + 2	5 mod 3 = 2	1*3 + 2
(-5) rem 3 = -2	(-1)*3 + (-2)	(-5) mod 3 = 1	(-2)*3 + 1
(-5) rem (-3) = -2	1*(-3) + (-2)	(-5) mod (-3) = -2	1*(-3) + (-2)
5 rem (-3) = 2	(-1)*(-3) + 2	5 mod (-3) = -1	(-2)*(-3) + (-1)

7.1 Aritmetik Operatörler

Örnek 7.1: Bir dijital saat tasarımı için gerekli VHDL kod bloğunun **architecture** bölümünü yazınız.

```
Architecture davranis of dijital_saat is
   signal saat, dakika, saniye: integer := 0;
   signal sayac: integer := 0;
Begin
   Process (clk)
   Begin
      If rising_edge(clk) then
         If sayac = (3600*24 - 1) then
           sayac <= 0;
         Else
           sayac <= sayac+1;
         End if;
      End if;
  End process;
  saat <= sayac / 3600;
  dakika <= (sayac mod 3600) / 60;
  saniye <= (sayac mod 60);
End davranis;
```

7.1 Aritmetik Operatörler

Örnek 7.2: Giriş portlarından gelen 2 adet 6-bitlik sayılar üzerine toplama, çıkarma, çarpma ve bölme işlemlerinin yapıldığı tasarımın VHDL kodunu yazınız.

- *ieee.std_logic_arith.all* paketi + **conv_std_logic_vector()** + **conv_integer()**

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;

Entity uygulama is
Port (clk : in std_logic;
      rst :  in std_logic;
      sayi1 :  in std_logic_vector (5 downto 0);
      sayi2 :  in std_logic_vector (5 downto 0);
      top : out std_logic_vector (7 downto 0);
      fark: out std_logic_vector (7 downto 0);
      carp: out std_logic_vector (11 downto 0);
      bol: out std_logic_vector (7 downto 0));
End uygulama;
```

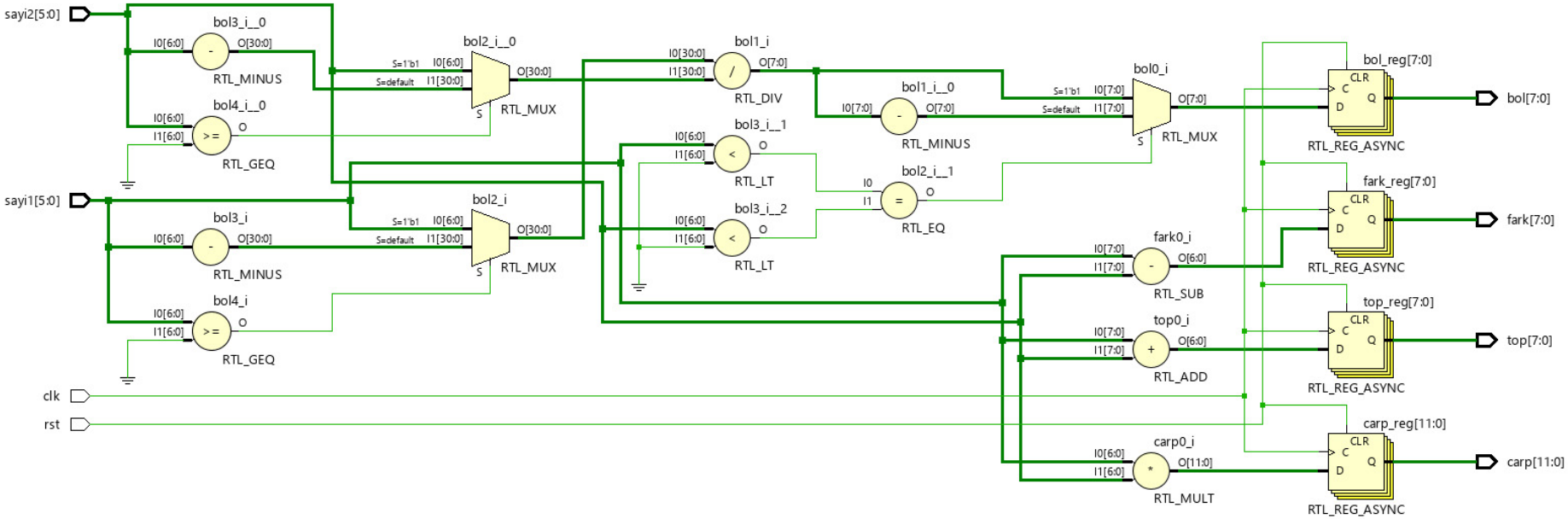
7.1 Aritmetik Operatörler

```
Architecture davranis of uygulama is
Begin
   Process (clk, rst)
      variable s1, s2: integer;
   Begin
      If (rst='1') then
         top <= (others=>'0');
         fark <= (others=>'0');
         carp <= (others=>'0');
         bol <= (others=>'0');
      Elsif rising_edge(clk) then
         s1 := conv_integer(signed(sayi1));
         s2 := conv_integer(signed(sayi2));
         top <= conv_std_logic_vector((s1+s2), top'length);
         fark <= conv_std_logic_vector((s1-s2), fark'length);
         carp <= conv_std_logic_vector((s1*s2), carp'length);
         bol <= conv_std_logic_vector((s1/s2), bol'length);
      End if;
   End process;
End davranis;
```

7.1 Aritmetik Operatörler

Örnek 7.2: Giriş portlarından gelen 2 adet 6-bitlik sayılar üzerine toplama, çıkarma, çarpma ve bölme işlemlerinin yapıldığı tasarımın VHDL kodunu yazınız.

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



7.1 Aritmetik Operatörler

Örnek 7.2: Aynı çözüm (RTL şeması da aynı kalacak şekilde) aşağıdaki VHDL kodu ile de sağlanır.

- *ieee.numeric_std.all* + **std_logic_vector()** + **to_integer()** + **to_signed()**

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity uygulama is
Port (clk, rst : in std_logic;
        sayi1, sayi2 :  in std_logic_vector (5 downto 0);
        top, fark, bol  : out std_logic_vector (7 downto 0);
        carp: out std_logic_vector (11 downto 0));
End uygulama;
```

7.1 Aritmetik Operatörler

Örnek 7.2: Aynı çözüm (RTL şeması da aynı kalacak şekilde) aşağıdaki VHDL kodu ile de sağlanabilir.

```
Architecture davranis of uygulama is
Begin
   Process (clk, rst)
      variable s1, s2: integer;
   Begin
      If (rst='1') then
         top <= (others=>'0');
         fark <= (others=>'0');
         carp <= (others=>'0');
         bol <= (others=>'0');
      Elsif rising_edge(clk) then
         s1 := to_integer(signed(sayi1));
         s2 := to_integer(signed(sayi2));
         top <= std_logic_vector(to_signed((s1+s2), top'length));
         fark <= std_logic_vector(to_signed((s1-s2), fark'length));
         carp <= std_logic_vector(to_signed((s1*s2), carp'length));
         bol <= std_logic_vector(to_signed((s1/s2), bol'length));
      End if;
   End process;
End davranis;
```

7.2 Öncelikli (Özel) Operatörler

- Öncelikli (özel) operatörler, en yüksek önceliğe sahip operatör tipidir. Bunlar; **üs alma** (**), **mutlak değer** (abs) ve **değili / tersi** (not) şeklinde bilinir.

Öncelikli operatörlere dair işaret ve işlem ilişkisi

İşaret	İşlem	Örnek
**	Üs Alma	3**2=9 5**4=625
ABS	Mutlak Değer	abs(-3)=3 abs(5)=5
NOT	Değili / Tersi	not 0 = 1 not 1 = 0

Değili / Tersi İçin Tanımlama Örnekleri;

- **not** a; // a' nın **tersi**,
- **not** a **and** b; // a' nın **tersi ve** b,
- **not** (a **and** b); // (a **ve** b)' nin **tersi**

7.2 Öncelikli (Özel) Operatörler

Örnek 7.2: Bir değer girişinin ‘3’ üssünü alan tasarım için gerekli VHDL kod bloğunun **entity** ve **architecture** bölümlerini yazınız.

```
Entity us_alma is
Port (giris: in integer;
      cikis: out integer);
End us_alma;

Architecture davranis of us_alma is
Begin
  cikis <= giris**3;
End davranis;
```

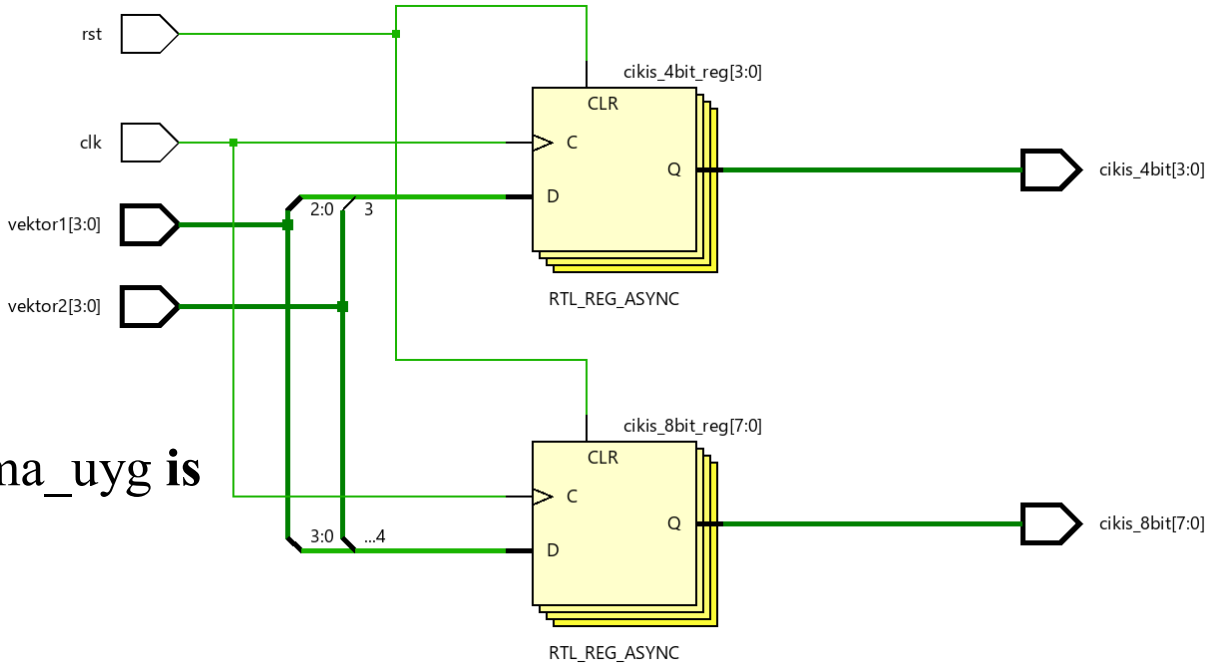
7.2 Öncelikli (Özel) Operatörler

- Ulama operatörü ise **'&'** (ampersand) işareti ile gösterilir. Bu operatör vektörleri genişletmek veya birbirine ulamak için işletilir.

Örnek 7.4: 4-bitlik iki adet giriş sinyalini, 8-bitlik ve 4-bitlik (ikinci vektörün son terimi ve ilk vektörün ilk 3 terimi olacak şekilde) iki ayrı çıkışta birleştirmek için gerekli VHDL kodunu yazınız.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity ulama_uyg is
Port (clk, rst : in std_logic;
      vektor1, vektor2:  in std_logic_vector (3 downto 0);
      cikis_8bit : out std_logic_vector (7 downto 0);
      cikis_4bit: out std_logic_vector (3 downto 0));
End ulama_uyg;
```

7.2 Öncelikli (Özel) Operatörler




```
Architecture davranis of ulama_uyg is
Begin
   Process (clk, rst)
   Begin
      If (rst='1') then
         cikis_8bit  <= (others=>'0');
         cikis_4bit <= (others=>'0');
      Elsif rising_edge(clk) then
         cikis_8bit <= vektor2 & vektor1;
         cikis_4bit <= vektor2(3) & vektor1(2 downto 0);
      End if;
   End process;
End davranis;
```

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

7.3 Mantıksal Operatörler

- En düşük önceliğe sahip olan operatör grubudur.

- Mantık operatörleri; **bit**, **bit_vector**, **boolean**, **std_logic** ve **std_logic_vector** türlerindeki veriler tarafından sık kullanılan operatörlerdir.

- **Boolean** türünde **true** ifadesi **lojik-1**' e karşılık gelirken, **false** ifadesi **lojik-0**' a denk düşer.

- Özünde **NOT** operatörü de mantıksal bir operatördür, ancak önceliği farklı olduğundan **Öncelikli (Özel) Operatörler** kategorisinde belirtilir.

- Mantıksal operatörlerden yalnızca **NOT** operatörü **tek parametrelidir**.

7.3 Mantıksal Operatörler

- ***Std_logic***, ***Bit*** veya ***Boolean*** türünde tanımlı **a** ve **b** sinyalleri üzerine mantıksal bir operatörün işletilmesi gerekiyorsa;

Tanımlama komutu: **a** *mantıksal_operatör* **b** şeklinde olmalıdır.

- Mantıksal operatörde işlenecek nesnelerin aynı tür ve boyutta olması gereklidir.

İki parametreli mantıksal operatörlerin doğruluk tablosu

Girişler		Çıkışlar					
a	**b**	**AND**	**NAND**	**OR**	**NOR**	**XOR**	**XNOR**
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

7.3 Mantıksal Operatörler

- Bazı sentezleyicilerde ***nand*** ve ***nor*** kapılarına dair komutlar tanımlı değildir.

- Farklı sentezleyiciler arası (Quartus, Vivado, Lattice, …) kod aktarımı yapılacaksa, sıkıntı olmaması adına;

- **a *nand* b = not (a *and* b)** veya
- **a *nor* b = not (a *or* b)** şeklinde yazım da tercih edilebilir.

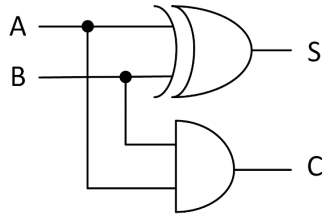
İki parametreli mantıksal operatörlerin doğruluk tablosu

Girişler		Çıkışlar					
a	**b**	**AND**	**NAND**	**OR**	**NOR**	**XOR**	**XNOR**
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

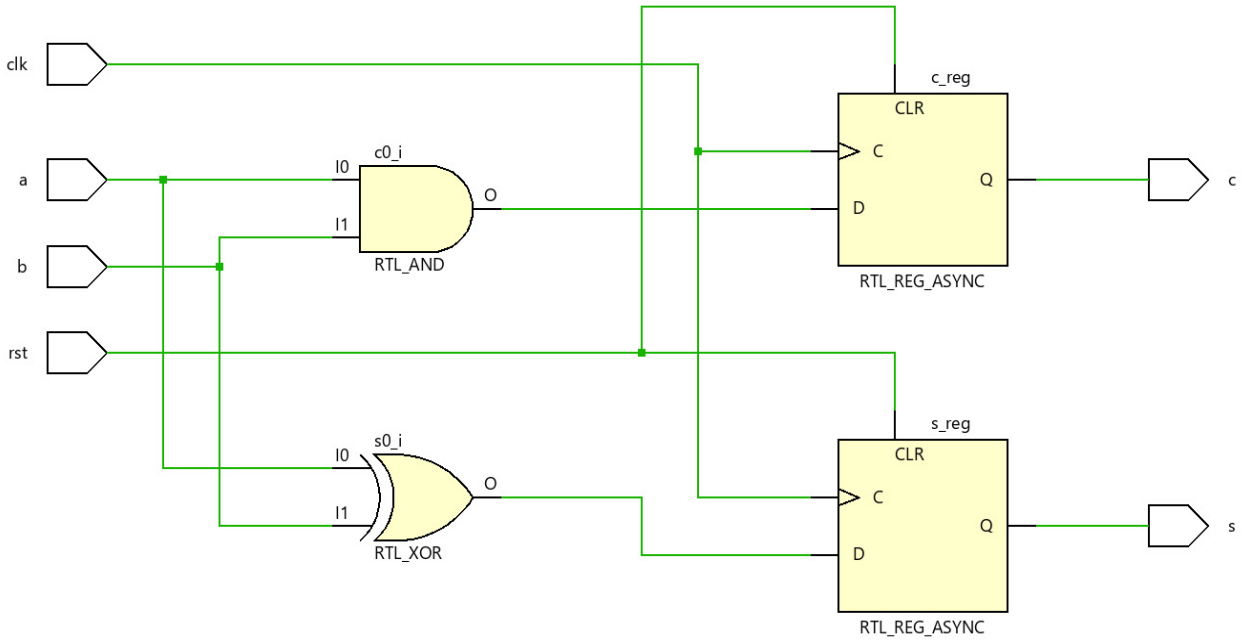
7.3 Mantıksal Operatörler

Örnek 7.5: Mantıksal yarı toplayıcı devresinin VHDL yazılımını, ***clk*** ve ***rst*** sinyallerine bağlı olarak (ardışıl tasarım temelinde) gerçekleştiriniz.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity yari_toplayici is
Port (clk, rst, a, b: in std_logic;
        s,c: out std_logic);
End yari_toplayici;

Architecture davranis of yari_toplayici is
Begin
   Process (clk, rst)
   Begin
      If (rst='1') then
         s  <= '0';
         c  <= '0';
      Elsif rising_edge(clk) then
         s  <= a xor b;
         c  <= a and b;
      End if;
   End process;
End davranis;
```



Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



7.4 Kaydırma Operatörleri

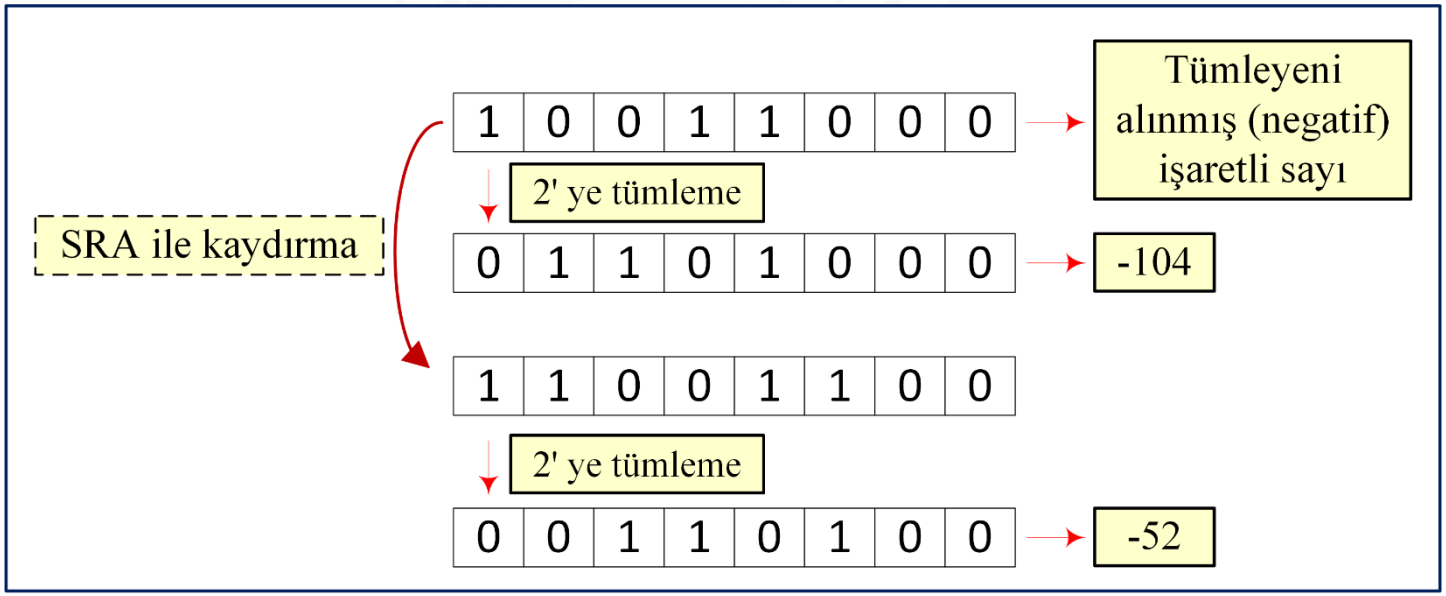
- Kaydırma operatörleri, sıklıkla **bit_vector** ve **std_logic_vector (unsigned** veya **signed'a çevirilen)** türündeki tek boyutlu vektörlerde **kaydırma işlemi** için kullanılır.

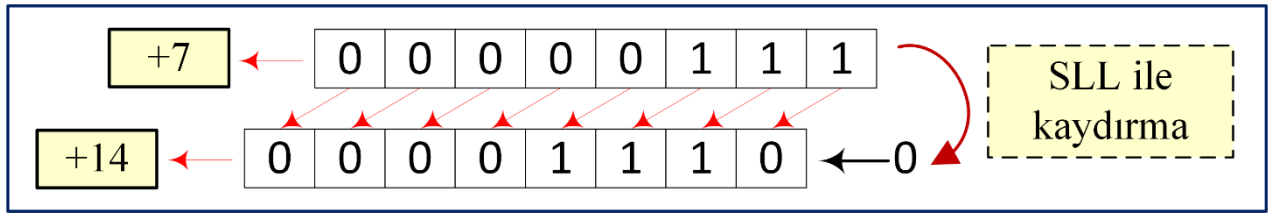
- İşlem neticesinde **aynı boyutta ve türde vektör oluşur**. Kaydırma işlemleri **mantıksal** ve **aritmetik** olmak üzere ikiye ayrılır.

Kaydırma Operatörleri		İşlev	
Mantıksal	Shift Left Logic (SLL)	Vektör içindeki elemanları ***operatörün sağında tanımlı olan sayı kadar sola*** kaydırır. Herbir kaymada en soldaki eleman atılırken, en sağdan '0' değeri eklenir.	V <= "00001111"; Y <= V ***SLL*** 1; Sonuç; Y <= "00011110";
	Shift Right Logic (SRL)	Vektör içindeki elemanları ***operatörün sağında tanımlı olan sayı kadar sağa*** kaydırır. Herbir kaymada en sağdaki eleman atılırken, en soldan '0' değeri eklenir.	V <= "01100111"; Y <= V ***SRL*** 2; Sonuç; Y <= "00011001";
Aritmetik	Shift Left Arithmetic (SLA)	Vektör içindeki elemanları ***operatörün sağında tanımlı olan sayı kadar sola*** kaydırır. Herbir kaymada LSB biti tekrarlanarak sağdan eklenir.	V <= "00100011"; Y <= V ***SLA*** 2; Sonuç; Y <= "10001111";
	Shift Right Arithmetic (SRA)	Vektör içindeki elemanları ***operatörün sağında tanımlı olan sayı kadar sağa*** kaydırır. Herbir kaymada MSB biti tekrarlanarak soldan eklenir.	V <= "01100010"; Y <= V ***SRA*** 1; Sonuç; Y <= "00110001";

7.4 Kaydırma Operatörleri

SRA ve SLL kaydırma örnekleri



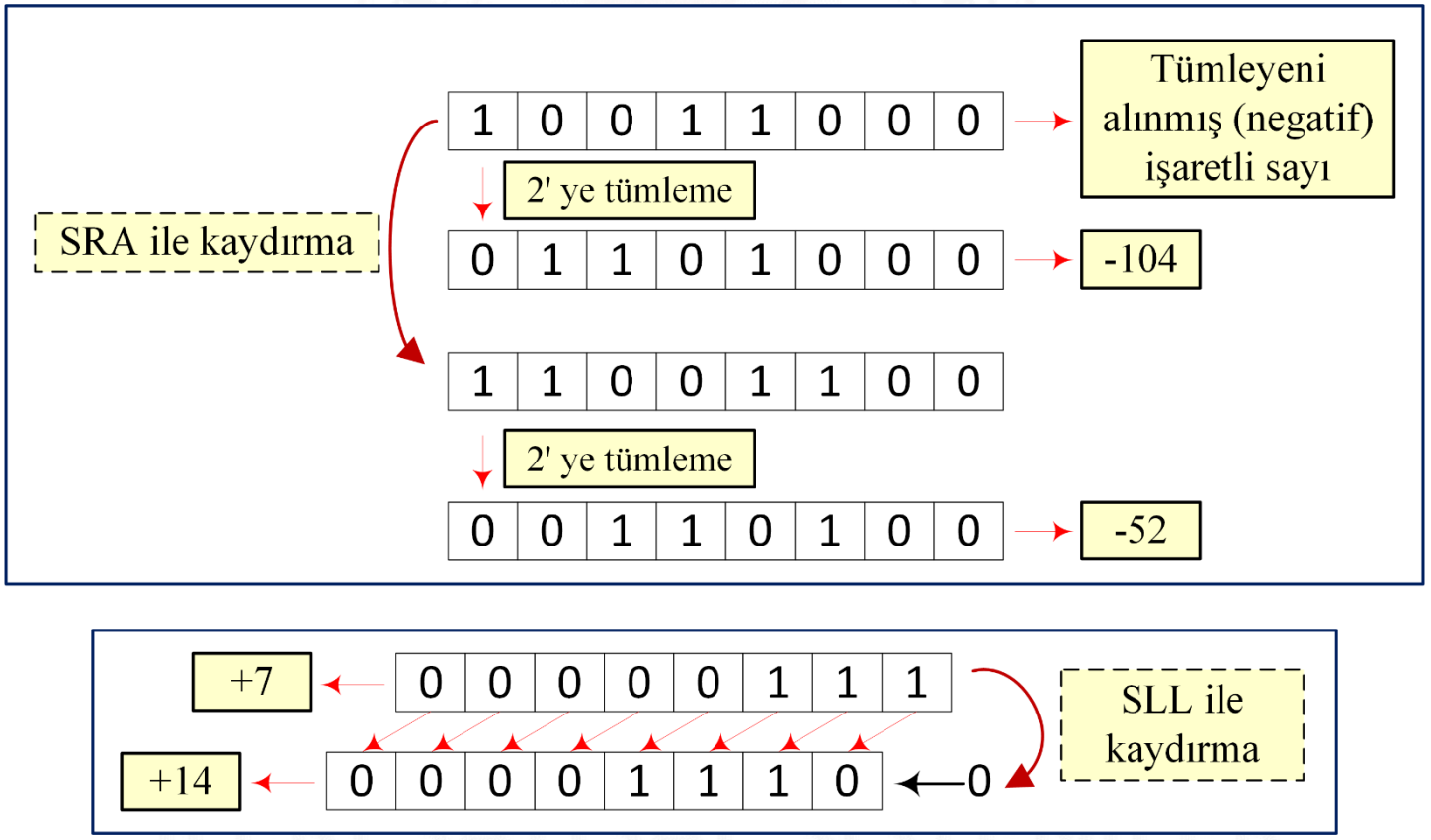


- Görülen örnekte **decimal 7 sayısı** **SLL** operatörü ile ötelenmiş (‘0’ biti eklenerek sola kaydırılmış) ve **decimal 14 sayısı** elde edilmiştir.

- İkili sayılarda ‘0’ biti ile kaydırma mantığı gereği; **sola kaydırmada ‘*2’** işlemi yapılırken, **sağa kaydırmada ‘/2’** işlemi yapılır.

7.4 Kaydırma Operatörleri

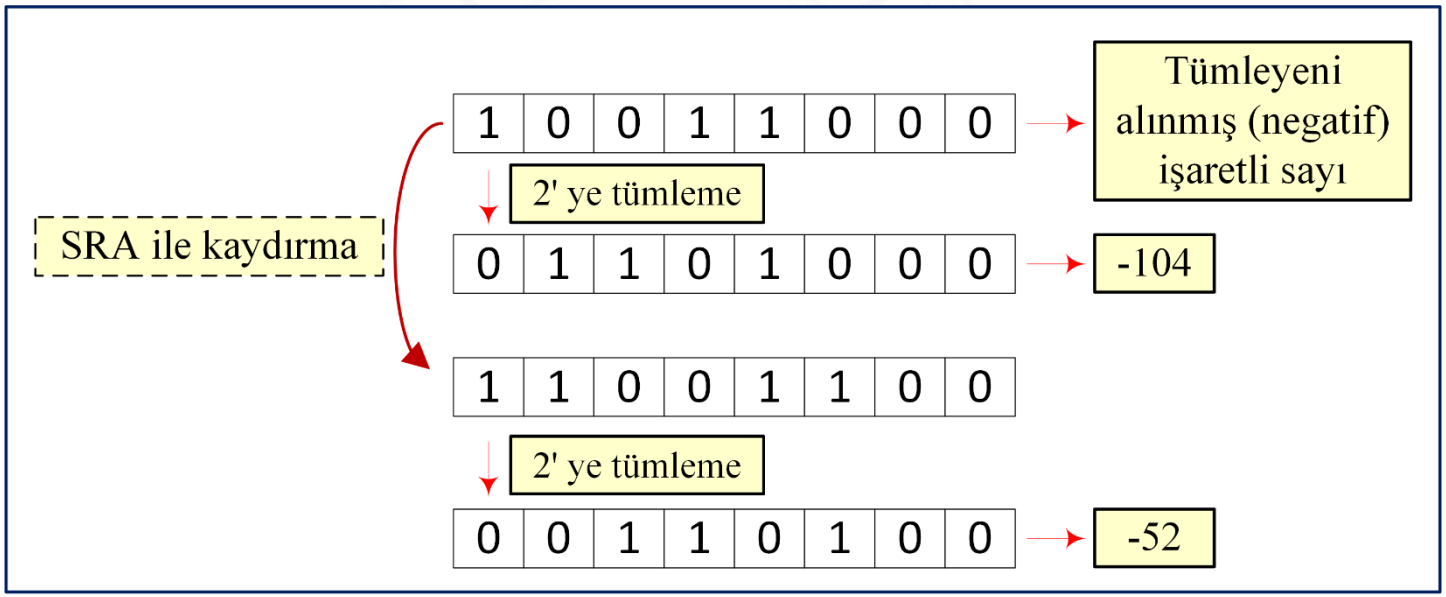
SRA ve SLL kaydırma örnekleri

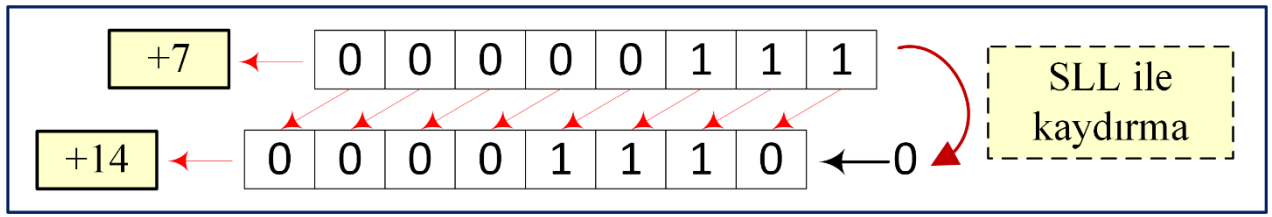


- **Aritmetik kaydırma** işlemi, genelde **negatif sayıları işlemede (tümleyeni alınmış) değerler için işaret bitinin kaybolmamasını sağlar.**

7.4 Kaydırma Operatörleri

SRA ve SLL kaydırma örnekleri





- Şekilde görüldüğü gibi, **tümleyeni alınmış (negatif) işaretli sayının SRA kullanılarak ötelenmesi gereklidir.**

- Aksi takdirde **negatif sayı üzerine '/2' işlemi gerçekleştirilemez.**

7.4 Kaydırma Operatörleri

Örnek 7.6: Aşağıda belirtilen VHDL koduna göre ***sonuc1***, ***sonuc2***, ***sonuc3*** ve ***sonuc4*** nesnelerinin ikili karşılıklarını yazınız.

```
Entity kaydirma_uyg is
Port (sonuc1, sonuc2, sonuc3, sonuc4: out bit_vector(7 downto 0));
End kaydirma_uyg;

Architecture davranis of kaydirma_uyg is
   signal vektor: bit_vector (7 downto 0) := "01101001";
Begin
   sonuc1 <= vektor sll 2;
   sonuc2 <= vektor srl 2;
   sonuc3 <= vektor sla 2;
   sonuc4 <= vektor sra 2;
End davranis;
```

sonuc1 = 10100100
sonuc2 = 00011010
sonuc3 = 10100111
sonuc4 = 00011010

- Kaydırma operatörlerinin kullanılması için aşağıdaki komutun (**numeric_std** paketinin) işletilmesi gereklidir:

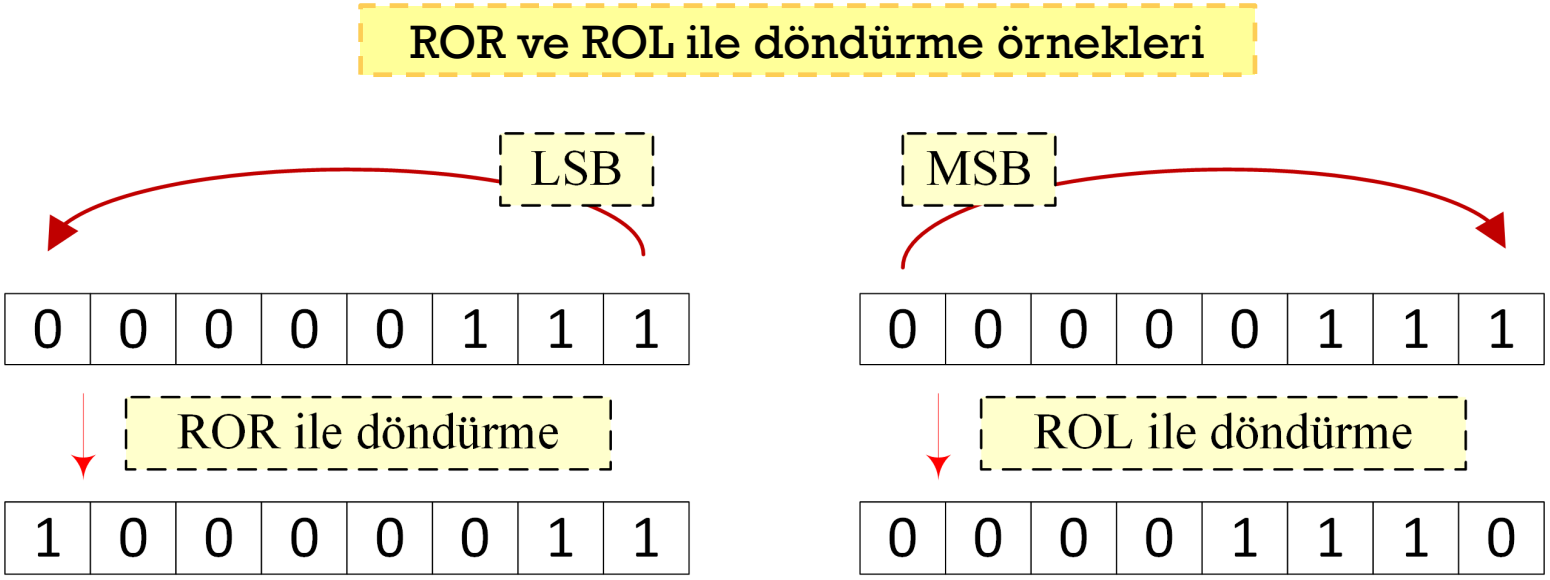
Paket tanımlama komutu: **Use** *ieee.numeric_std.all*;

7.5 Çevirme / Döndürme Operatörleri

- Çevirme / Döndürme operatörleri, sıklıkla **bit_vector** ve **std_logic_vector (unsigned** veya **signed'a çevirilen)** türündeki tek boyutlu vektörlerde **döndürme işlemi** için kullanılır. Sonuçta **aynı tür ve boyutta yeni bir vektör elde edilir**.

- Sağa döndürme (Rotate Right → **ROR**) işleminde vektör içindeki değerler sağa kaydırılırken, **LSB biti MSB' nin yerini alır** (**dizi soluna LSB gelir**).

- Sola döndürme (Rotate Left → **ROL**) işleminde vektör içindeki değerler sola kaydırılırken, **MSB biti LSB' nin yerini alır** (**dizi sağına MSB gelir**).



7.5 Çevirme / Döndürme Operatörleri

Örnek 7.7: Aşağıda belirtilen VHDL koduna göre ***sonuc1***, ***sonuc2***, ***sonuc3*** ve ***sonuc4*** nesnelerinin ikili karşılıklarını yazınız.

sonuc1 = 11010010
sonuc2 = 10100101
sonuc3 = 10110100
sonuc4 = 01011010

```
Entity dondurme_uyg is
Port (sonuc1, sonuc2, sonuc3, sonuc4: out bit_vector(7 downto 0));
End dondurme_uyg;

Architecture davranis of dondurme_uyg is
   signal vektor: bit_vector (7 downto 0) := "01101001";
Begin
   sonuc1 <= vektor rol 1;
   sonuc2 <= vektor rol 2;
   sonuc3 <= vektor ror 1;
   sonuc4 <= vektor ror 2;
End davranis;
```

- Döndürme operatörlerinin kullanılması için aşağıdaki komutun (**numeric_std** paketinin) işletilmesi gereklidir:

Paket tanımlama komutu: **Use** *ieee.numeric_std.all*;

7.6 Karşılaştırma Operatörleri

- İki adet aynı türden veriyi karşılaştıran ve **boolean** türünde (**true** veya **false**) veri üreten operatörlerdir.

- Bu operatörler **IF** ve **WHEN** yapıları ile sıklıkla kullanılır. Vektör şeklinde tanımlı nesneler karşılaştırılacağı zaman, **dizi türü ve boyutları aynı olmak zorundadır**.

Karşılaştırma Operatörleri

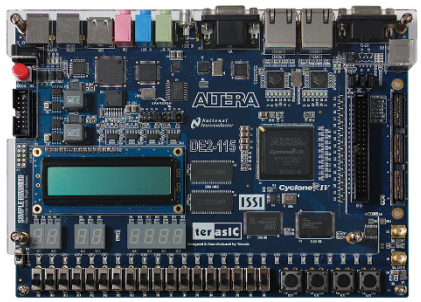
Operatör Sembolü	İşlev	
a = b	Eşittir	a ***eşit*** b
a /= b	Eşit değil	a ***eşit değil*** b
a < b	Küçüktür	a ***küçük*** b
a <= b	Küçük eşit	a ***küçük veya eşit*** b
a > b	Büyüktür	a ***büyük*** b
a >= b	Büyük eşit	a ***büyük veya eşit*** b

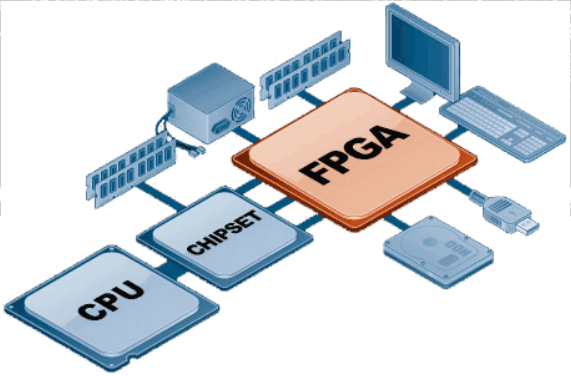
SONRAKİ DERS KONUSU

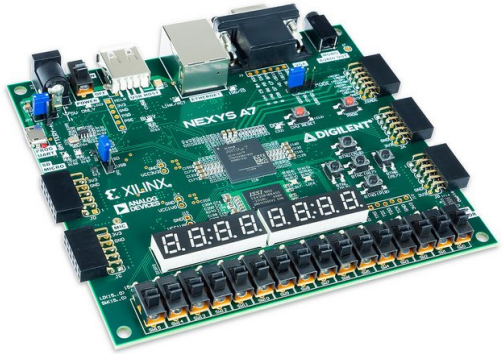
8- KOMBİNASYONEL LOJİK DEVRELER

İLERİ SAYISAL SİSTEMLER

8- KOMBİNASYONEL LOJİK DEVRELER







Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

8. KOMBİNASYONEL LOJİK DEVRELER

- VHDL tasarım dili ile **sonradan değiştirilebilir sayısal devre tasarımları** gerçekleştirilebilmektedir.

- Bu devre tiplerinden biri de **kombinasyonel devrelerdir**.

- Bu devreler, herhangi bir **hafıza birimi içermezler** ve **girişindeki veriyi tasarım fonksiyonuna göre işleyerek doğrudan çıkış üretirler**.

- Mantık kapıları ile gerçeklenen devreler olmak üzere; **toplayıcı**, **çoklayıcı** ve **kodlayıcı** gibi devreler **kombinasyonel devre tasarımı ile gerçeklenir.**

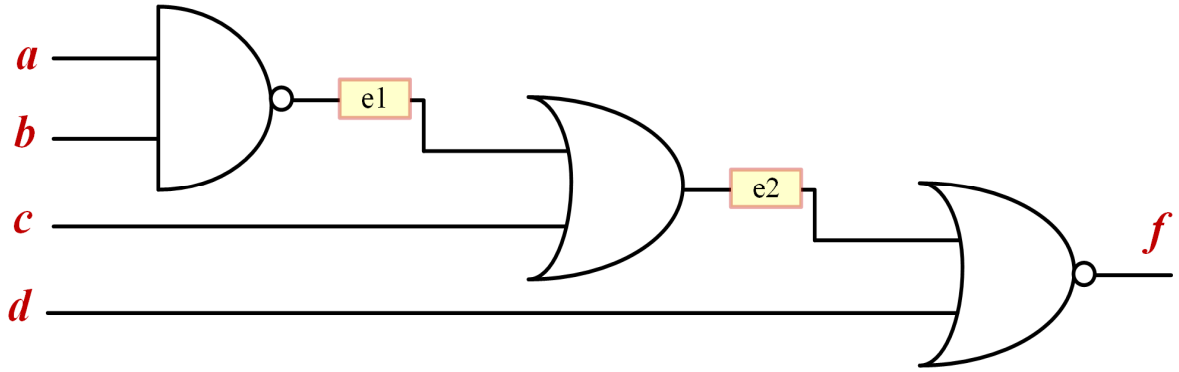
8.1 Katmanlı Devre Tasarımı

- Katmanlı devre tasarımlarında, blok çıkışları için **ara kablo bağlantıları** belirtilerek **komplike tasarımlarda esneklik sağlanır**.

- Bu kablo bağlantıları genelde **sinyal** türünde tanımlanır.

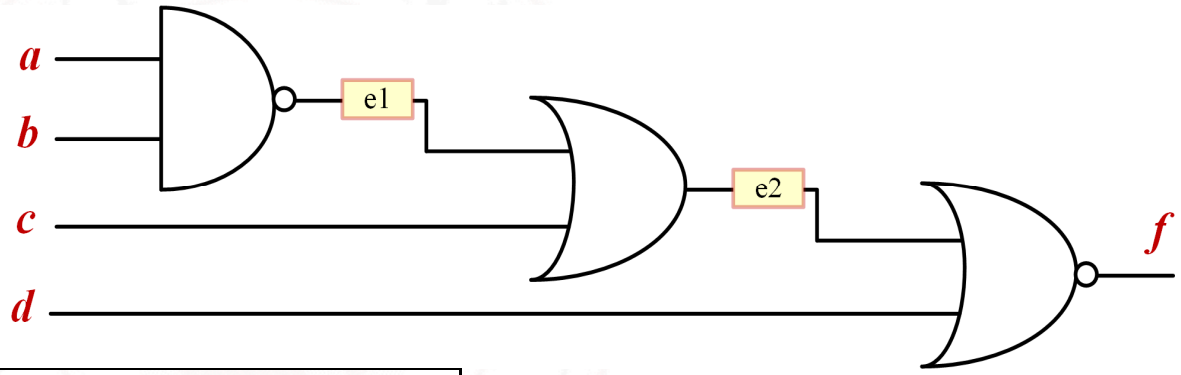
- **Ara kablo bağlantısı kullanmadan da yalın kodlar ile tasarım sağlanabilmektedir**.

Örnek 8.1: Aşağıda belirtilen devreyi VHDL dilinde tasarlayınız.



8.1 Katmanlı Devre Tasarımı

Örnek 8.1: Belirtilen devreyi VHDL dilinde tasarlayınız.




```
Library ieee;
Use ieee.std_logic_1164.all;

Entity eszamanli_islem is
Port (a, b, c, d : in std_logic;
        f : out std_logic);
End eszamanli_islem;

Architecture davranis of eszamanli_islem is
   signal e1, e2: std_logic;  -- ara kablo tanımlaması
Begin
   e1 <= a nand b; -- kablo ara sonucu
   e2 <= e1 or c; -- kablo ara sonucu
   f <= e2 nor d;
End davranis;
```

Ara kablo kullanmadan;

f <= ((a **nand** b) **or** c) **nor** d;

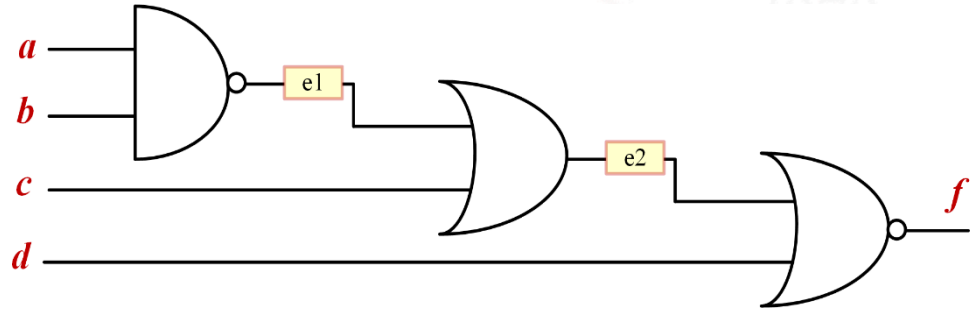
8.1 Katmanlı Devre Tasarımı

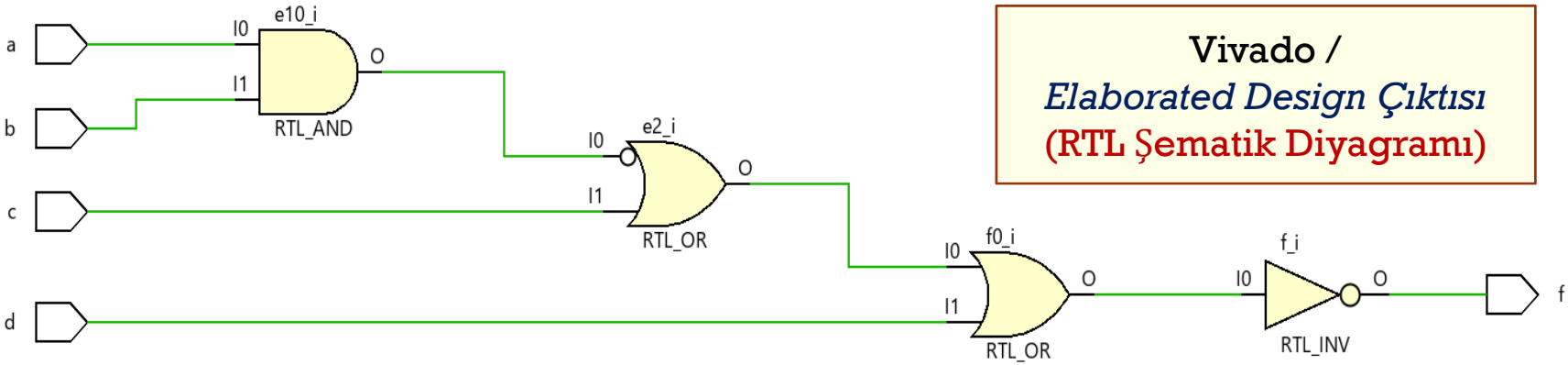
- Bazı sentezleyicilerde ***nand*** ve ***nor*** kapılarına dair komutlar tanımlı değildir.

- Farklı sentezleyiciler arası (Quartus, Vivado, Lattice, …) kod aktarımı yapılacaksa, sıkıntı olmaması adına;

- **a *nand* b = not (a *and* b)** veya

- **a *nor* b = not (a *or* b)** şeklinde yazım da tercih edilebilir.

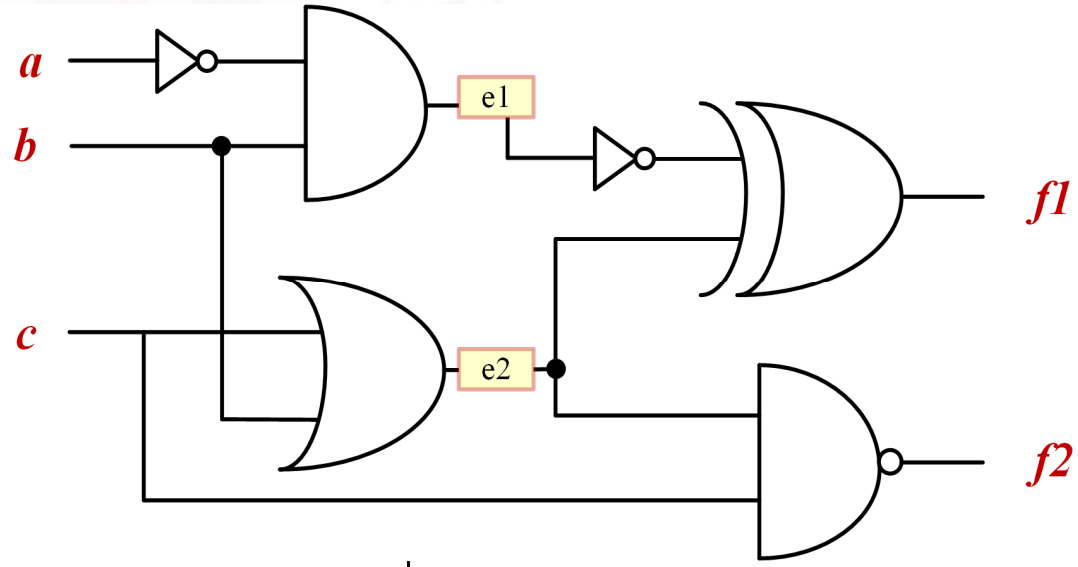




Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

8.1 Katmanlı Devre Tasarımı

Örnek 8.2: Aşağıda belirtilen devreyi VHDL dilinde tasarlayınız.



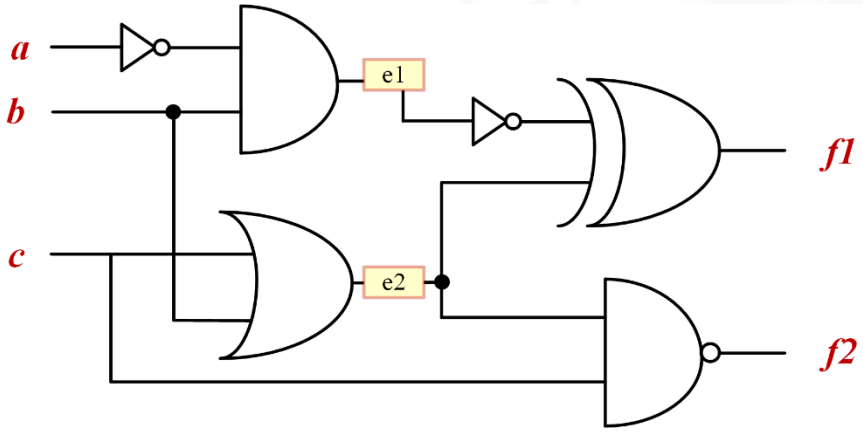

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity karma_islem is
Port (a, b, c : in std_logic;
        f1, f2 : out std_logic);
End karma_islem;

Architecture davranis of karma_islem is
   signal e1, e2: std_logic;  -- ara kablo tanımlaması
Begin
   e1 <= not a and b; -- kablo ara sonucu
   e2 <= b or c; -- kablo ara sonucu
   f1 <= not e1 xor e2;
   f2 <= e2 nand c;
End davranis;
```

Ara kablo kullanmadan;

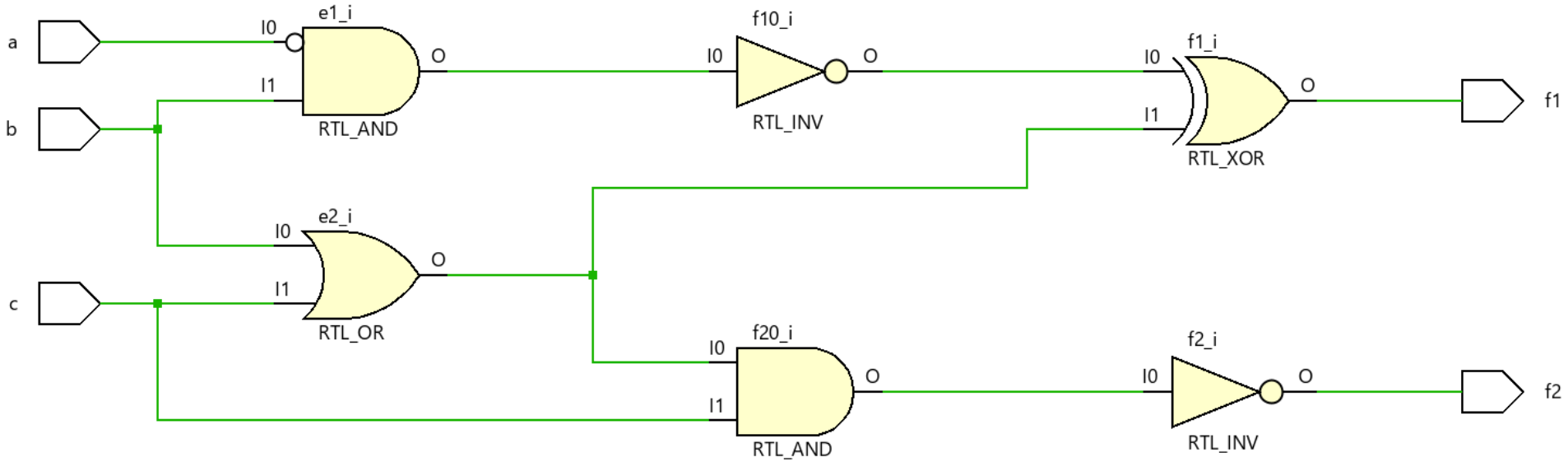
```
f1 <= (not (not a and b)) xor (b or c);

f2 <= (b or c) nand c;
```

8.1 Katmanlı Devre Tasarımı

Örnek 8.2: Aşağıda belirtilen devreyi VHDL dilinde tasarlayınız.



Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



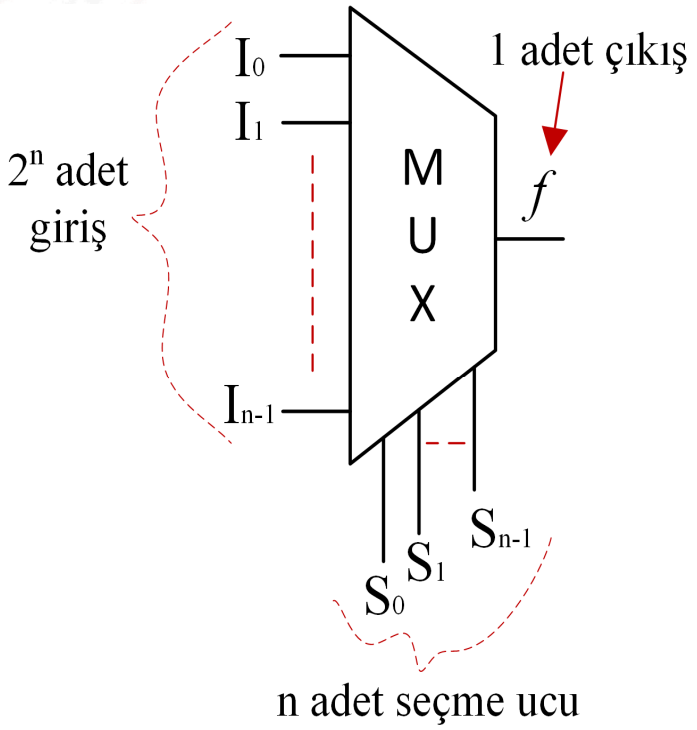
8.2 Çoklayıcılar

- Lojikte **çoklayıcı** veya **multiplexer** olarak bilinen ve bir grup girişten yalnızca bir tanesini seçerek çıkışa aktaran kombinasyonel elemandır.

- ***İşlemci birimleri gibi komplike yapıların en temel bileşenlerinden birisidir.***

- Bir çoklayıcının **normal giriş portlarının** yanısıra **seçme girişleri** de bulunur ve **seçme girişlerinin alacağı duruma göre gerekli girişin çıkışa aktarılması sağlanır.**

- Çoklayıcıların VHDL içinde tanımlamaları, **birçok farklı kombinasyonel elemanın tanımlanmasını sağlayan koşullu atama** veya **seçmeli atama** ile sağlanabilir.



Çoklayıcı (Multiplexer) yapısı

8.3 Koşullu Atamalar (When-Else)

- Bir sinyal nesnesine; **belirli bir grup değerden yalnızca birinin aktarılması için**, **gerekli şart yerine geldiği durumda atama sağlayan işlemdir.**

- **Koşullu atama** işlemi **When-Else** ifadesi işletilerek sağlanır ve tümleşik devre tasarımlarında sıklıkla tercih edilir.

- Gerekli koşul **When** ifadesinden sonra yazılır.

Tanımlama komutu: f <= ifade_1 **When** kosul_1 **Else**

ifade_2 **When** kosul_2 **Else**

…

ifade_n;

8.3 Koşullu Atamalar (When-Else)

- 2:1 MUX yapısına dair doğruluk tablosu yan tarafta sunulmuştur.

- İki bit giriş olduğundan ve seçme ucunun da analize dâhil edilmesi gerektiğinden **3 bitlik durumun incelenmesi gerekir.**

- Buna göre **'s'** seçme ucu **'0'** olduğu durumda **'a' girişi f çıkışına** aktarılır ve **s=1** olduğu durumda **'b' girişi f çıkışına** aktarılacaktır.

s	a	b	f
0	0	0	0
	0	1	0
	1	0	1
	1	1	1
1	0	0	0
	0	1	1
	1	0	0
	1	1	1

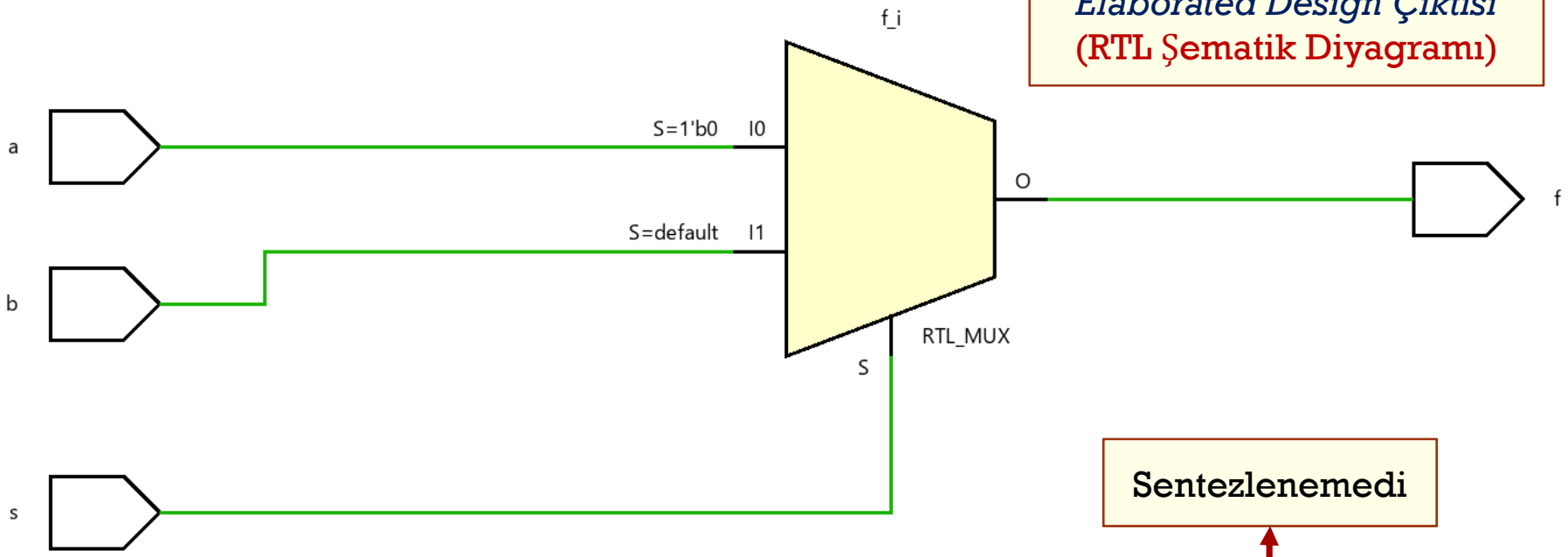
- **When-Else** ifadesi ile VHDL kodu aşağıdaki gibi yazılabilir.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity multiplexer_kodu is
Port (a, b, s : in std_logic;
      f : out std_logic);
End multiplexer_kodu;
```

```
Architecture davranis of multiplexer_kodu is
Begin
   f <= a When s='0' Else -- s=0 ise a değerini ata
        b When s='1' Else -- s=1 ise b değerini ata
        'Z'; --eğer seçme sinyali lojik-0 veya lojik-1
değilse Z ata
End davranis;
```

8.3 Koşullu Atamalar (When-Else)

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



Sentezlenemedi

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity multiplexer_kodu is
Port (a, b, s : in std_logic;
      f : out std_logic);
End multiplexer_kodu;
```

```
Architecture davranis of multiplexer_kodu is
Begin
   f <= a When s='0' Else -- s=0 ise a değerini ata
        b When s='1' Else -- s=1 ise b değerini ata
        'Z'; --eğer seçme sinyali lojik-0 veya lojik-1
değilse Z ata
End davranis;
```

8.3 Koşullu Atamalar (When-Else)

Örnek 8.3: VHDL dilinde yazılmış bir yapay zekâ sınıflandırma algoritması ile akciğer verilerine dair 4 farklı sınıfın ayırt edilmesi istenmiştir.

Bu sınıflar; ***sağlıklı*** (**00**), ***bronşit*** (**01**), ***astım*** (**10**) ve ***nodül*** (**11**) şeklindedir.

Yapay zekâ çıkış birimi için gerekli VHDL sınıflama kodunu yazınız.

(**Not:** yapay zekâ çıkışı MUX seçme ucuna göre atanacaktır ve herbir sınıf MUX girişidir)

```
Library ieee;
Use ieee.std_logic_1164.all;
Use std.textio.all;

Entity cikis_birimi is
Port (secim : in std_logic_vector (1 downto 0);
      cikis : out string (1 to 3));
End cikis_birimi;

Architecture davranis of cikis_birimi is
Begin
  cikis <=  "SAG" When secim = "00" Else
            "BRO" When secim = "01" Else
            "AST" When secim = "10" Else
            "NOD" When secim = "11" Else
            "XXX";
End davranis;
```

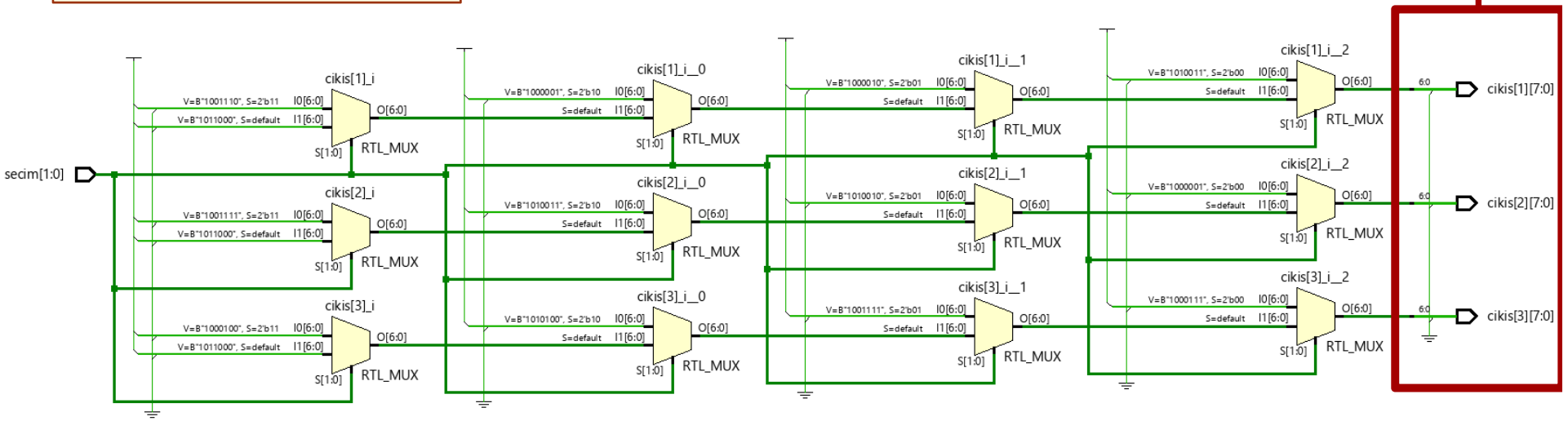
8.3 Koşullu Atamalar (When-Else)

Karakter türü sentezlenemez

Bu sınıflar; ***sağlıklı*** (**00**), ***bronşit*** (**01**), ***astım*** (**10**) ve ***nodül*** (**11**) şeklindedir.

```
Architecture davranis of cikis_birimi is
Begin
   cikis <=  "SAG" When secim = "00" Else
             "BRO" When secim = "01" Else
             "AST" When secim = "10" Else
             "NOD" When secim = "11" Else
             "XXX";
End davranis;
```

Vivado / *Elaborated Design Çıktısı* (RTL Şematik Diyagramı)



8.4 Seçimli Atamalar (With-Select)

- Tümleşik devrelerde sıklıkla tercih edilen bir diğer atama yöntemi de **With-Select** ifadesidir.

- **Seçimli atama** olarak bilinen bu atama yöntemi, koşullu yöntem olan **When-Else** ifadesine benzer, ancak **yazım biçimi bakımından farklılık gösterir.**

- Genel programlama dillerinde **Switch-Case** ifadesine benzer işlev görür.

- Gerekli seçme deyimi **With** ve **Select** ifadeleri arasına, seçme deyiminin alabileceği ifadeler **When** sonrasına, seçime göre gerekli koşul ise **When** öncesine yazılır.

- Tek çıkışın olduğu kombinasyonel tümleşik tasarımlarda sıklıkla tercih edilir.

8.4 Seçimli Atamalar (With-Select)

Tanımlama komutu: **With** secme_deyimi **Select**

f <= ifade_1 **When** secenek_1,
ifade_2 **When** secenek_2,
…
ifade_n **When** Others;

▪ Çoklayıcı (Multiplexer) yapısının **When-Else** yerine **With-Select** ifadesi üzerinden eldesini inceleyelim.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity multiplexer_kodu is
Port (a, b, s : in std_logic;
      f : out std_logic);
End multiplexer_kodu;
```

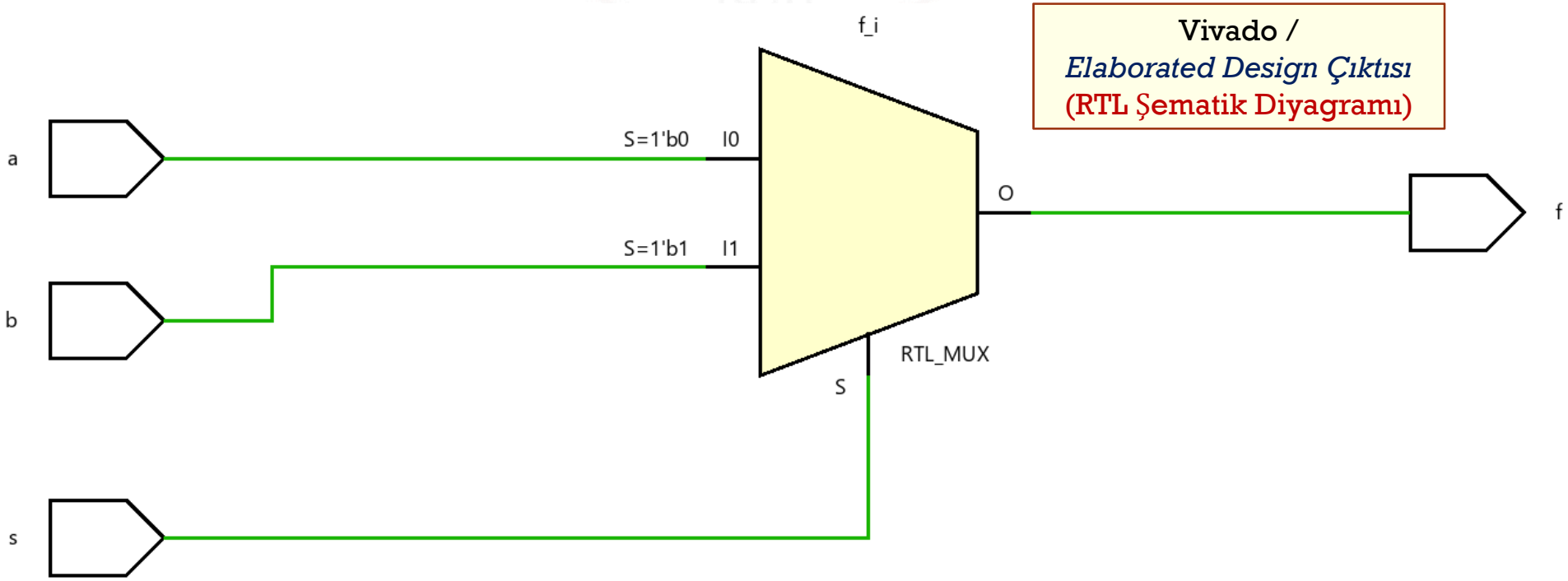
```
Architecture davranis of multiplexer_kodu is
Begin
   With s Select
      f <= a When '0',
           b When '1',
           'Z' When Others;
End davranis;
```

8.4 Seçimli Atamalar (With-Select)

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity multiplexer_kodu is
Port (a, b, s : in std_logic;
      f : out std_logic);
End multiplexer_kodu;
```

```
Architecture davranis of multiplexer_kodu is
Begin
   With s Select
      f <= a When '0',
           b When '1',
           'Z' When Others;
End davranis;
```

Sentezlenemedi



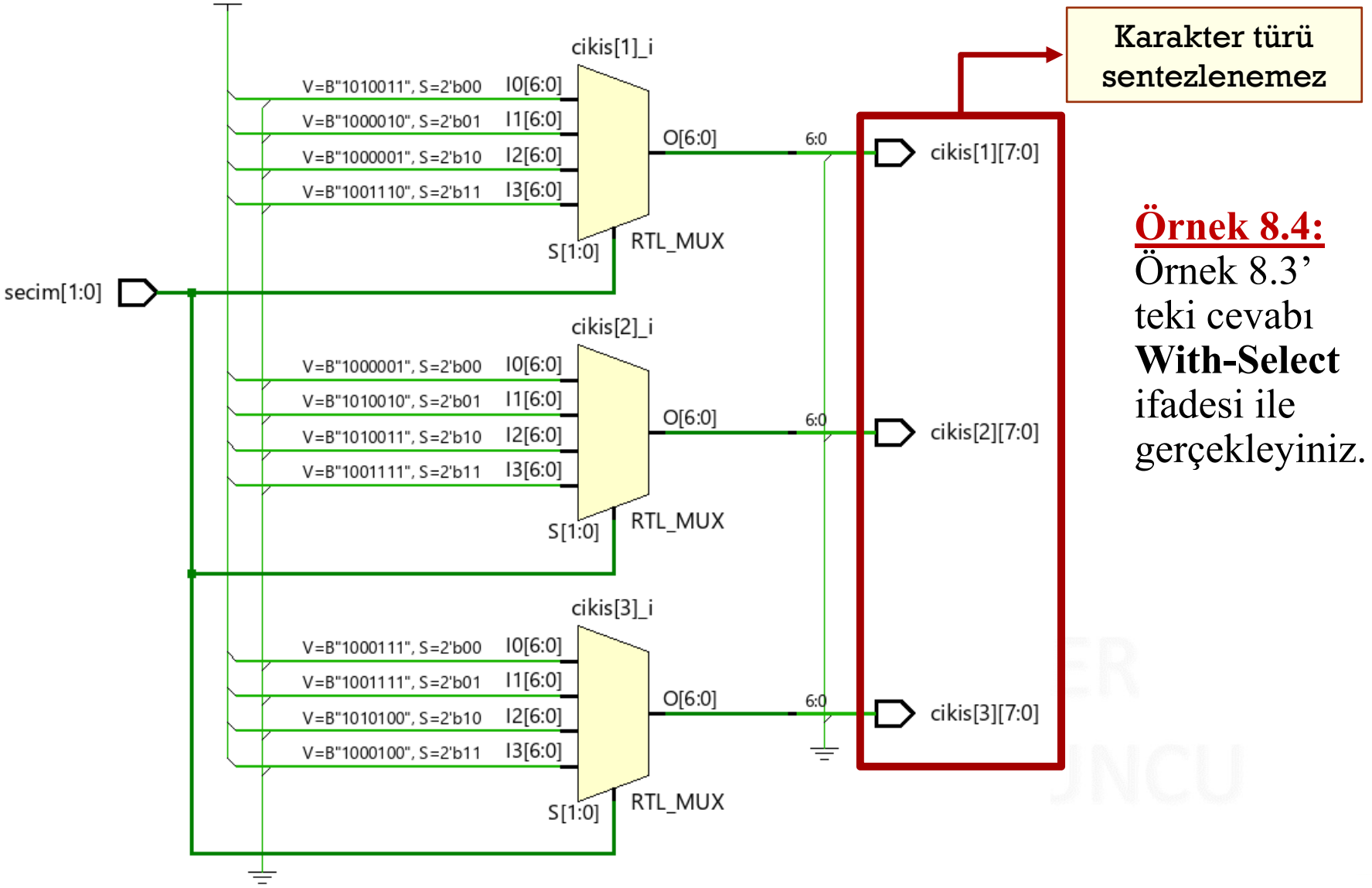
8.4 Seçimli Atamalar (With-Select)

Örnek 8.4: Örnek 8.3' teki cevabı **With-Select** ifadesi ile gerçekleyiniz.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use std.textio.all;

Entity cikis_birimi is
Port (secim : in std_logic_vector (1 downto 0);
        cikis : out string (1 to 3));
End cikis_birimi;

Architecture davranis of cikis_birimi is
Begin
   With secim Select
      cikis <= "SAG" When "00",
               "BRO" When "01",
               "AST" When "10",
               "NOD" When "11",
               "XXX" When Others;
End davranis;
```

8.4 Seçimli Atamalar (With-Select)

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

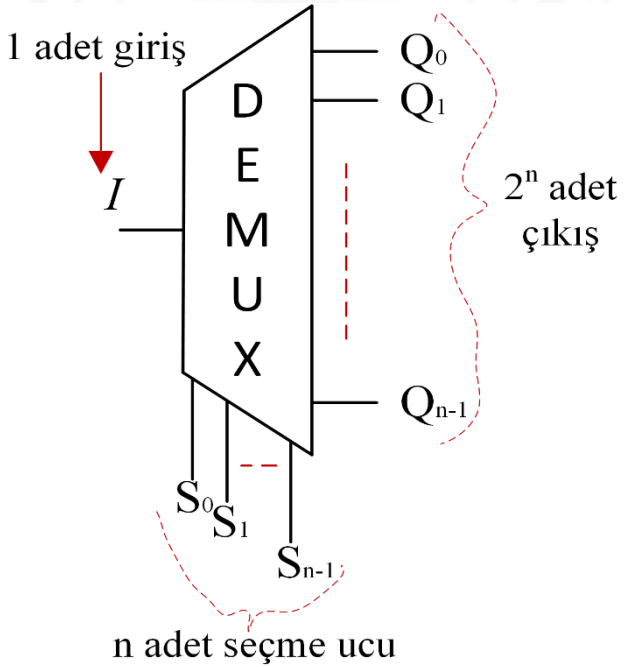


Örnek 8.4:
Örnek 8.3'teki cevabı **With-Select** ifadesi ile gerçekleyiniz.

8.5 Tekleyiciler

- Çoklayıcıların gerçekleştirdiği işlevin **tam tersini** sağlarlar**.**

- Giriş portundaki veriyi, **seçme uçlarına göre istenilen çıkışa aktarır**. Tekleyici gösterimi, **çoklayıcının dikeyde aynalanmış halidir**.



Tekleyici (Demultiplexer) yapısı

8.5 Tekleyiciler

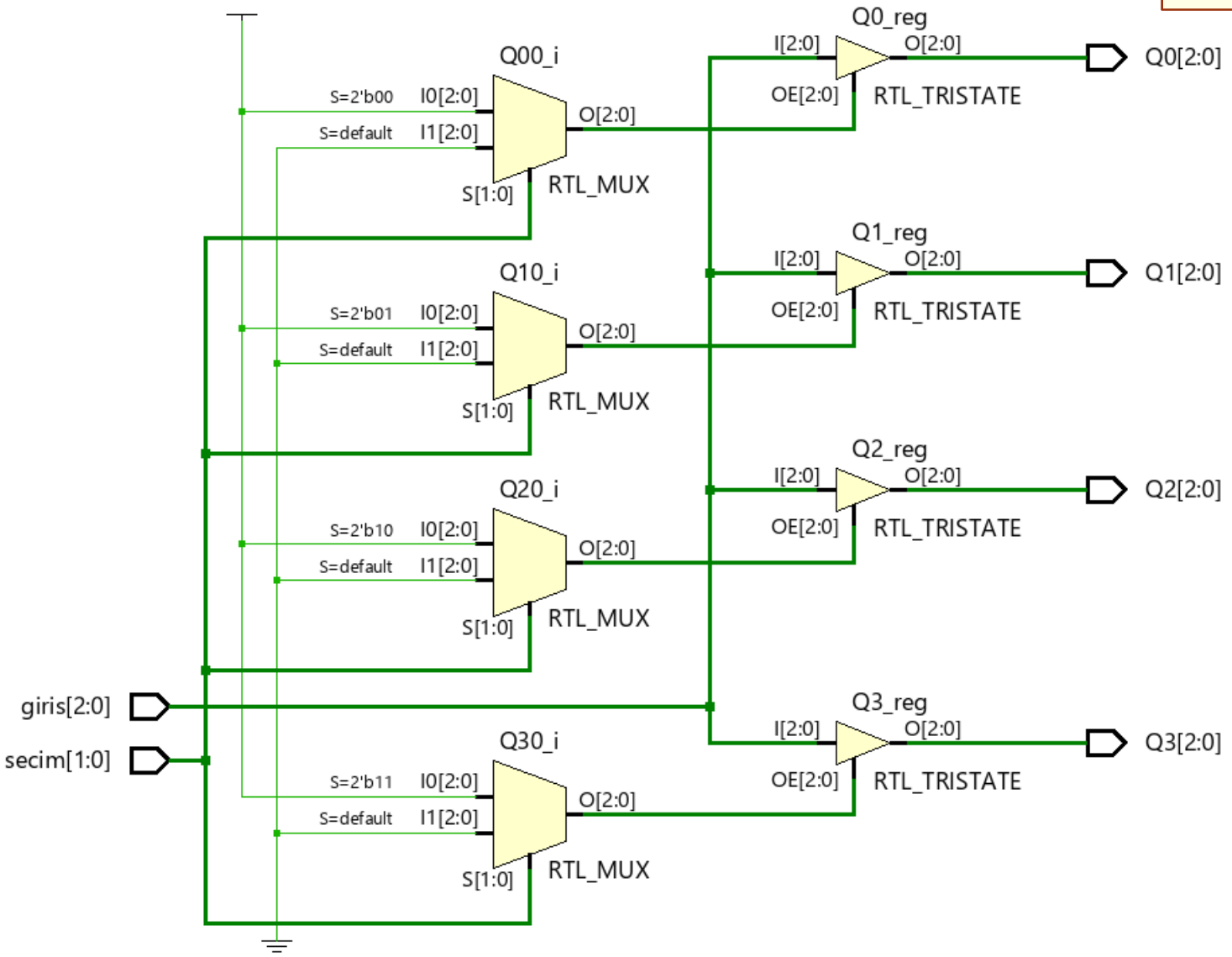
- **1:4 Tekleyici (Demultiplexer)** için gerekli VHDL konunu, **3 bitlik veri** işletilecek şekilde (**With-Select** ifadesi üzerinden yazımı daha fazla kod satırı içereceğinden) **When-Else** ifadesi üzerinden yazalım.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity demux_1e4 is
Port (giris : in std_logic_vector (2 downto 0);
      secim : in std_logic_vector (1 downto 0);
      Q0, Q1, Q2, Q3 : out std_logic_vector (2 downto 0));
End demux_1e4;

Architecture davranis of demux_1e4 is
Begin
      Q0 <= giris When secim="00" Else "ZZZ";
      Q1 <= giris When secim="01" Else "ZZZ";
      Q2 <= giris When secim="10" Else "ZZZ";
      Q3 <= giris When secim="11" Else "ZZZ";
End davranis;
```

8.5 Tekleyiciler

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

Derleyici yüksek empedansı temsili olarak tri-state buffer karşılığı ile gerçekledi.



8.6 Kodlama Devresi

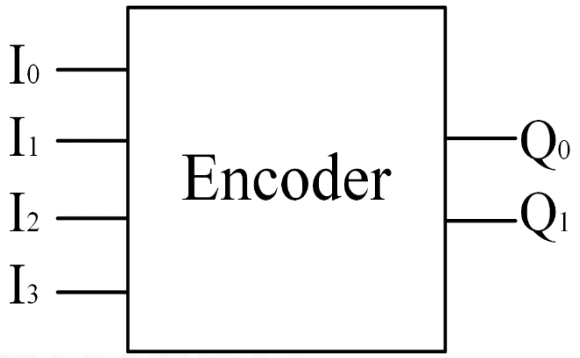
- **Kodlama**, **veriyi farklı formalara dönüştürmek** için işletilir.

- BCD, Gray, Excess-3 kodlama gibi farklı formlarda birçok kodlama yöntemi mevcuttur.

- Yine lojik tabanda **kodlayıcı** (**encoder**) yapısı **en temel kodlayıcı formudur**.

- **Kodlayıcılar**, çoklayıcı ve tekleyicilerde olduğu gibi ***giriş : çıkış*** basamaklarına göre isimlendirilir.

- Buna göre **4 girişi** ve **2 çıkışı** olan bir kodlayıcı yapısının gösterimi için **4:2 Kodlayıcı** ifadesi yazılır.



4:2 Encoder yapısı

8.6 Kodlama Devresi

- Lojik tabanda en temel kodlayıcı olan **encoder** yapısı için **4:2** → **giriş:çıkış** olacak şekilde VHDL kodunu **When-Else** ifadesi ile yazalım.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity encoder_4e2 is
Port (giris : in std_logic_vector (3 downto 0);
      cikis : out std_logic_vector (1 downto 0));
End encoder_4e2;
```

```
Architecture davranis of encoder_4e2 is
Begin
      cikis <= "00" When giris = "0001" Else
               "01" When giris = "0010" Else
               "10" When giris = "0100" Else
               "11" When giris = "1000" Else
               "ZZ";
End davranis;
```

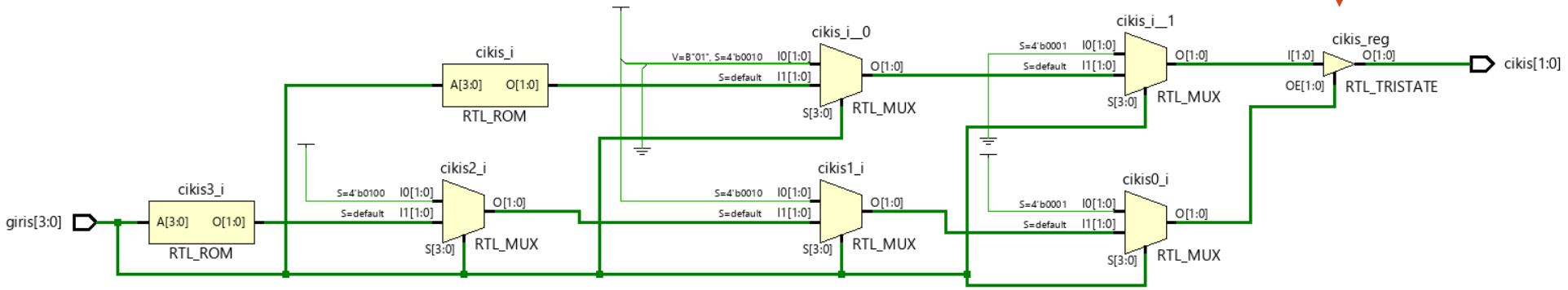
8.6 Kodlama Devresi

- Lojik tabanda en temel kodlayıcı olan **encoder** yapısı için **4:2** → **giriş:çıkış** olacak şekilde VHDL kodunu **When-Else** ifadesi ile yazalım.

```
Architecture davranis of encoder_4e2 is
Begin
        cikis <= "00" When giris = "0001" Else
                 "01" When giris = "0010" Else
                 "10" When giris = "0100" Else
                 "11" When giris = "1000" Else
                 "ZZ";
End davranis;
```

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

Derleyici yüksek empedansı temsili olarak tri-state buffer karşılığı ile gerçekledi.



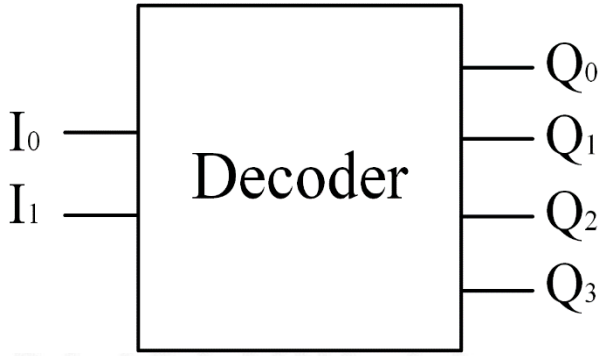
8.7 Kod Çözücü Devre Tasarımı

- **Kod çözücü** birimler, kodlayıcılar tarafından **kodlanmış verinin ilk halini elde etmek amacıyla işletilir.**

- **Decoder** olarak da bilinen bu yapılar; RAM ve tampon belleklerde gerekli bellek bölümünü adreslemek, 7-segment display sürmek ve veri çoklama gibi birçok amaç için kullanılır.

- Kod çözücülerde de kodlayıcılarda olduğu gibi isimlendirme yapılır.

- Örneğin; **2 girişi** ve **4 çıkışı** olan bir **decoder** yapısının gösterimi için **2:4 Decoder** ifadesi yazılır.



2:4 Decoder yapısı

8.7 Kod Çözücü Devre Tasarımı

- Bu kod çözücünün VHDL kodunu **When-Else** ifadesi üzerinden yazalım

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity decoder_2e4 is
Port (giris : in std_logic_vector (1 downto 0);
      cikis : out std_logic_vector (3 downto 0));
End decoder_2e4;
```

```
Architecture davranis of decoder_2e4 is
Begin
      cikis <= "0001" When giris = "00" Else
               "0010" When giris = "01" Else
               "0100" When giris = "10" Else
               "1000" When giris = "11" Else
               "ZZZZ";
End davranis;
```

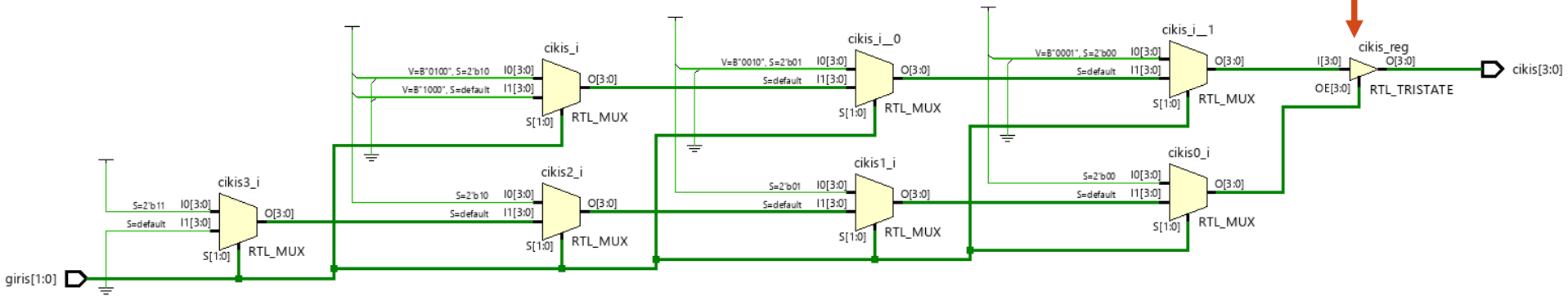
8.7 Kod Çözücü Devre Tasarımı

- Bu kod çözücünün VHDL kodunu **When-Else** ifadesi üzerinden yazalım

```
Architecture davranis of decoder_2e4 is
Begin
        cikis <= "0001" When giris = "00" Else
                 "0010" When giris = "01" Else
                 "0100" When giris = "10" Else
                 "1000" When giris = "11" Else
                 "ZZZZ";
End davranis;
```

Derleyici yüksek empedansı temsili olarak tri-state buffer karşılığı ile gerçekledi.

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

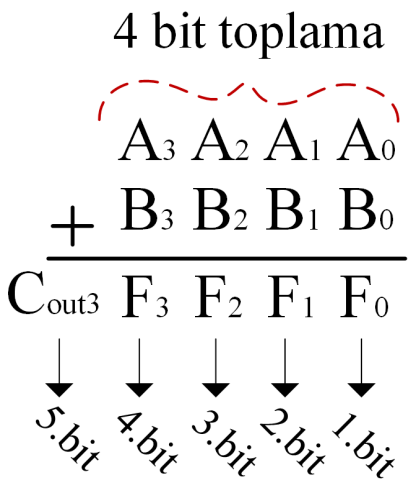


8.8 Toplama Devreleri

▪Lojik tasarımın en temel bölümlerinden biri olduğundan birçok **toplama devresi** tasarımı mevcuttur.

▪ **Aritmetik işlemler**in sağlanabilmesi için **Std_logic_arith / Numeric_std** paketinin tanımlanması gerekir.

▪ Normal bir toplamada **iki adet n bitlik sayı toplandığı zaman**, **(n+1) bit çıkış elde edilecektir**. Bu noktada **(n+1). bit elde çıkışını tutar**.



4 bitlik toplama işlemi örneği

8.8 Toplama Devreleri

- 4 bitlik iki sayının toplanmasını sağlayan **toplama devresi** için gerekli kodu VHDL dilinde yazalım.

- Kod yazmada kolaylık sağlanması için, **generic** deyimi ile bu işlemi gerçekleştirelim.

Tasarım-1

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Use ieee.std_logic_unsigned.all;

Entity toplayici_4bit is
Generic (k : natural := 4);
Port (A : in std_logic_vector (k-1 downto 0);
      B : in std_logic_vector (k-1 downto 0);
      elde : out std_logic;
      fcik : out std_logic_vector (k-1 downto 0));
End toplayici_4bit;

Architecture davranis of toplayici_4bit is
  signal ara_islem : std_logic_vector (k downto 0);
Begin
      ara_islem <= ('0' & A) + ('0' & B);
      fcik <= ara_islem (k-1 downto 0);
      elde <= ara_islem (k);
End davranis;
```

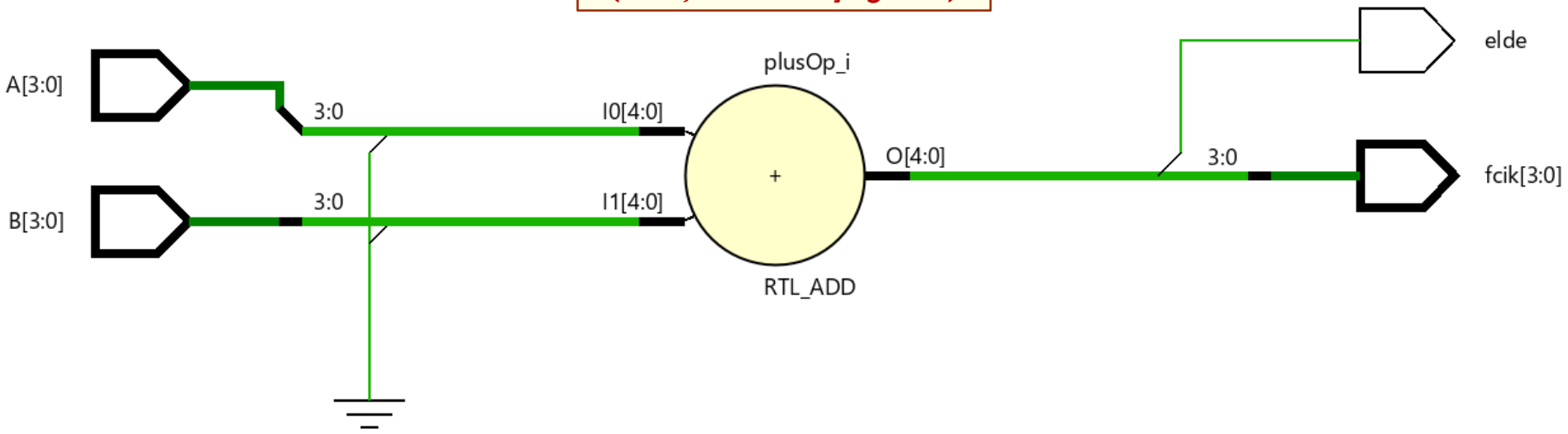
8.8 Toplama Devreleri

- 4 bitlik iki sayının toplanmasını sağlayan **toplama devresi** için gerekli kodu VHDL dilinde yazalım.

Tasarım-1

```
Architecture davranis of toplayici_4bit is
   signal ara_islem : std_logic_vector (k downto 0);
Begin
        ara_islem <= (‘0’ & A) + (‘0’ & B);
        fcik <= ara_islem (k-1 downto 0);
        elde <= ara_islem (k);
End davranis;
```

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



8.8 Toplama Devreleri

Tasarım-2

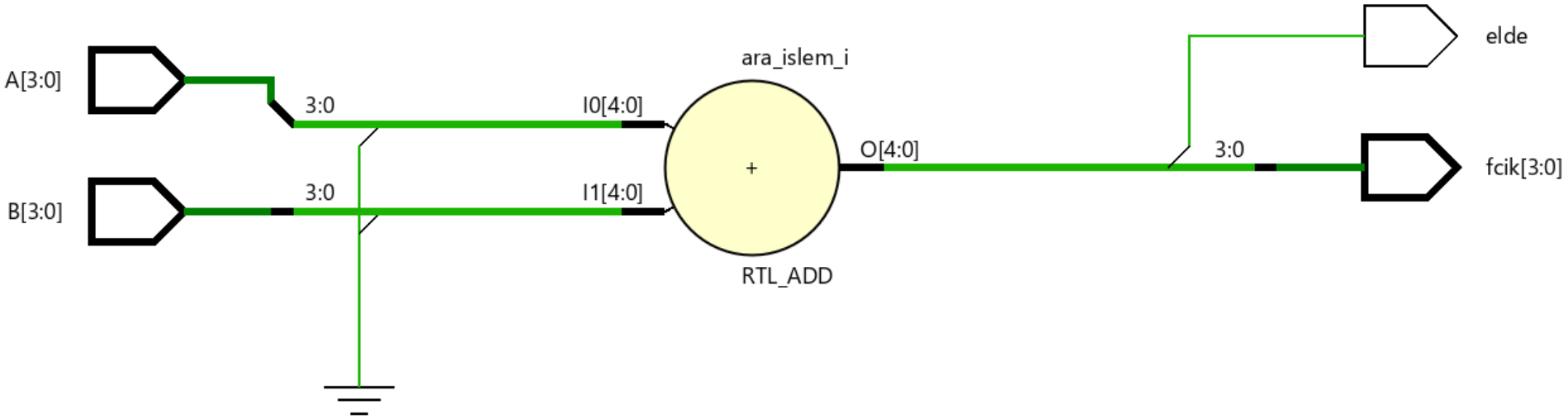
```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity toplayici_4bit is
Generic (k : natural := 4);
Port (A, B : in std_logic_vector (k-1 downto 0);
        elde : out std_logic;
        fcik : out std_logic_vector (k-1 downto 0));
End toplayici_4bit;

Architecture davranis of toplayici_4bit is
   signal ara_islem : unsigned (k downto 0);
Begin
        ara_islem <= unsigned('0' & A) + unsigned('0' & B);
        fcik <= std_logic_vector(ara_islem (k-1 downto 0));
        elde <= ara_islem (k);
End davranis;
```

8.8 Toplama Devreleri

Tasarım-2

```
Architecture davranis of toplayici_4bit is
   signal ara_islem : unsigned (k downto 0);
Begin
        ara_islem <= unsigned('0' & A) + unsigned('0' & B);
        fcik <= std_logic_vector(ara_islem (k-1 downto 0));
        elde <= ara_islem (k);
End davranis;
```

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



8.9 Karşılaştırma Devresi

- **Karşılaştırma işlemi**, **iki terimin üç koşulda kıyaslanması** amacıyla kullanılır.

- Matematiksel anlamda da kıyaslamada; değerlerden birisinin diğerinden **küçük**, diğerinden **büyük** veya **birbirlerine eşit** olma durumları meydana gelebilir.

- Bu bağlamda kıyaslama devreleri, **n bitlik iki girişe göre üç farklı durumun analizini sağlarlar.**

- 4 bitlik iki sayının kıyaslanmasını sağlayan devreyi **When-Else** ifadesi üzerinden yazalım.

8.9 Karşılaştırma Devresi

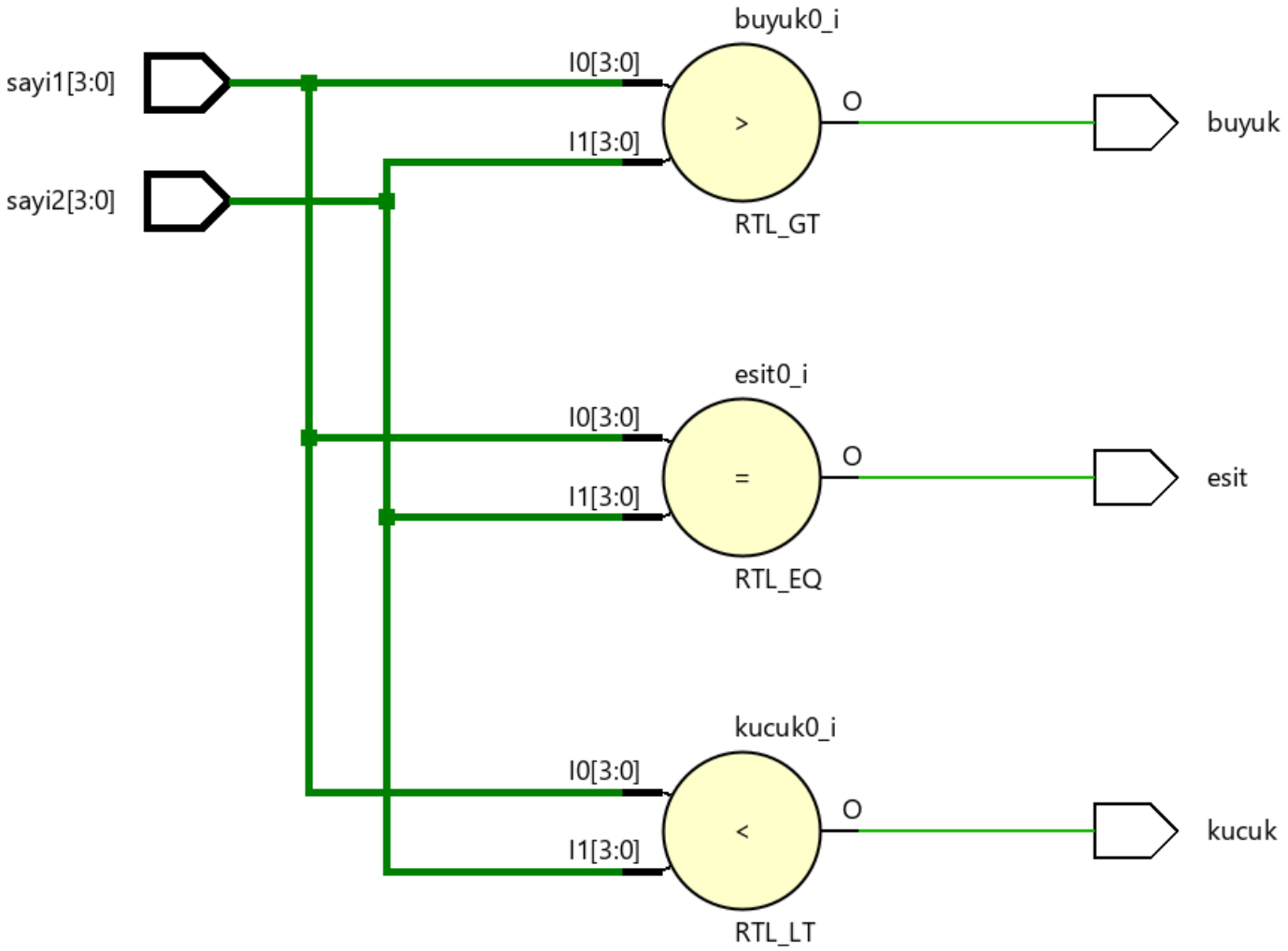
- Bu noktada **matematiksel olarak üç durumdan yalnızca birisi etkin olabileceği için**, **herbir seçenekte Else sonrası '0' yazılması gereklidir.**

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity karsilastir_4bit is
Generic (n : natural := 4);
Port (sayi1, sayi2 : in std_logic_vector (n-1 downto 0);
        kucuk, esit, buyuk : out std_logic);
End karsilastir_4bit;
```

```
Architecture davranis of karsilastir_4bit is
Begin
        kucuk <= '1' When sayi1<sayi2 Else '0';
        esit <= '1' When sayi1=sayi2 Else '0';
        buyuk <= '1' When sayi1>sayi2 Else '0';
End davranis;
```

8.9 Karşılaştırma Devresi

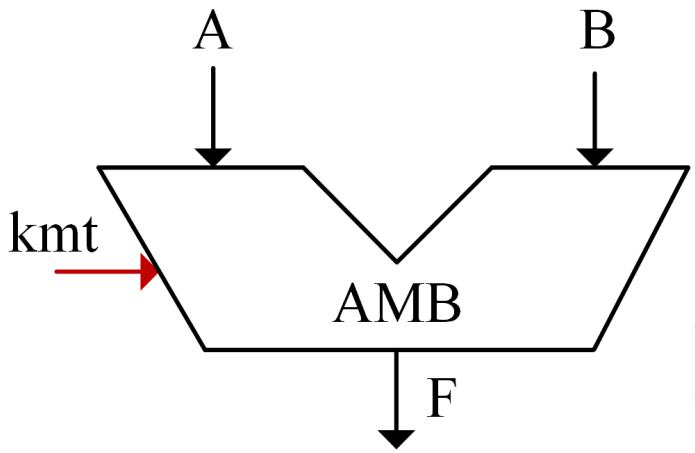
Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



8.10 Aritmetik Mantık Birimi

- **Aritmetik Mantık Birimi** (**AMB**); aritmetik ve lojik işlemleri bir arada gerçekleştirmeyi sağlayan, mikrodenetleyiciden mikroişlemciye kadar birçok işlemci biriminin yapıtaşı niteliğinde olan sayısal devredir.

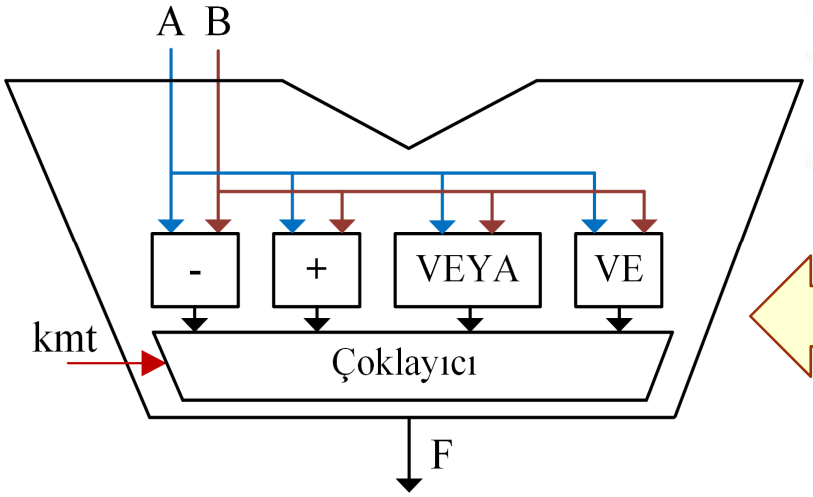
- Aşağıdaki şekilde görüldüğü üzere; ***A*** ve ***B*** **iki veriye dair giriş portlarını**, ***kmt*** **işlem seçme ucunu** ve ***F*** ise **sistem çıkış portunu** simgeler.



Aritmetik Mantık Birimi (AMB) genel şeması

8.10 Aritmetik Mantık Birimi

- Aşağıdaki şekilde ise kullanıcı isteği doğrultusunda tasarlanmış bir **AMB içyapısı** yer almakta, ***seçme ucuna göre işlem sırası*** ise ***tabloda*** sunulmaktadır.



Tasarlanan Örnek AMB iç yapısı

Tasarlanan AMB' nin işlem sırası

İşlem Kodu (kmt)	İşlem
00	Çıkarma
01	Toplama
10	VEYA
11	VE

8.10 Aritmetik Mantık Birimi

- Giriş verileri 4 bit olduğu durum için gerekli VHDL kodunu önce **With-Select** sonra da **When-Else** ifadeleri üzerinden yazalım.

Tasarım-1

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;

Entity ornek_AMB is
Port (A, B : in std_logic_vector (3 downto 0);
       kmt : in std_logic_vector (1 downto 0);
       F : out std_logic_vector (3 downto 0));
End ornek_AMB;
```

```
Architecture davranis of ornek_AMB is
Begin
   With kmt Select
      F <= A - B  When  "00",
           A + B  When  "01",
           A or B  When  "10",
           A and B  When  "11",
           "ZZZZ"  When  Others;
 End davranis;
```

Tasarım-2

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_unsigned.all;
Use ieee.std_logic_arith.all;

Entity ornek_AMB is
Port (A, B : in std_logic_vector (3 downto 0);
       kmt : in std_logic_vector (1 downto 0);
       F : out std_logic_vector (3 downto 0));
End ornek_AMB;
```

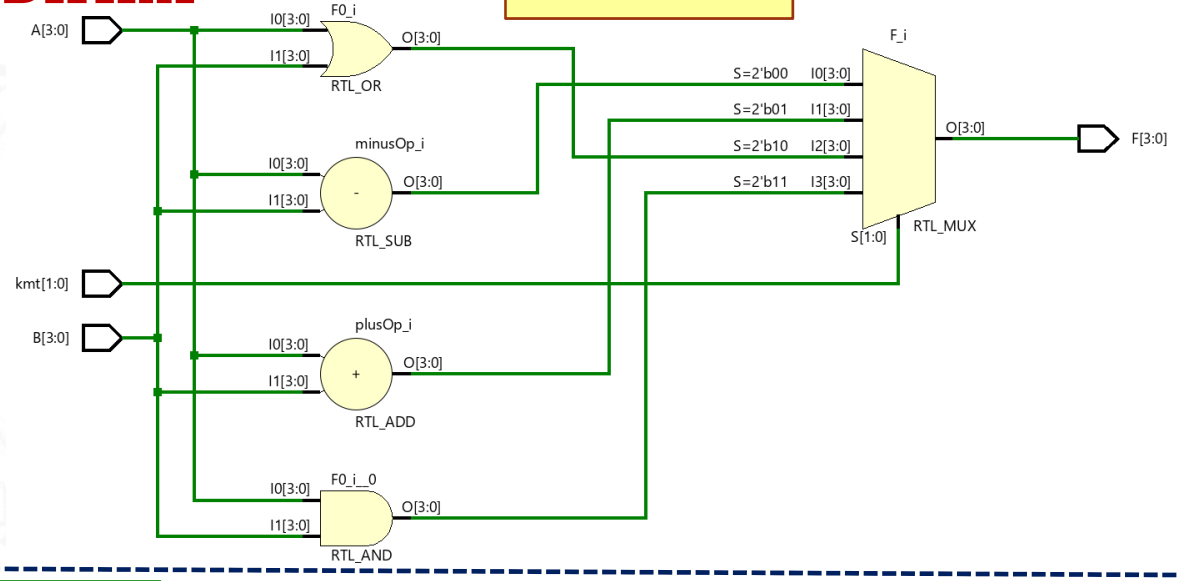
```
Architecture davranis of ornek_AMB is
Begin
   F <= A - B  When  kmt = "00" Else
        A + B  When  kmt = "01" Else
        A or B  When  kmt = "10" Else
        A and B  When  kmt = "11" Else
        "ZZZZ";
End davranis;
```

8.10 Aritmetik Mantık Birimi

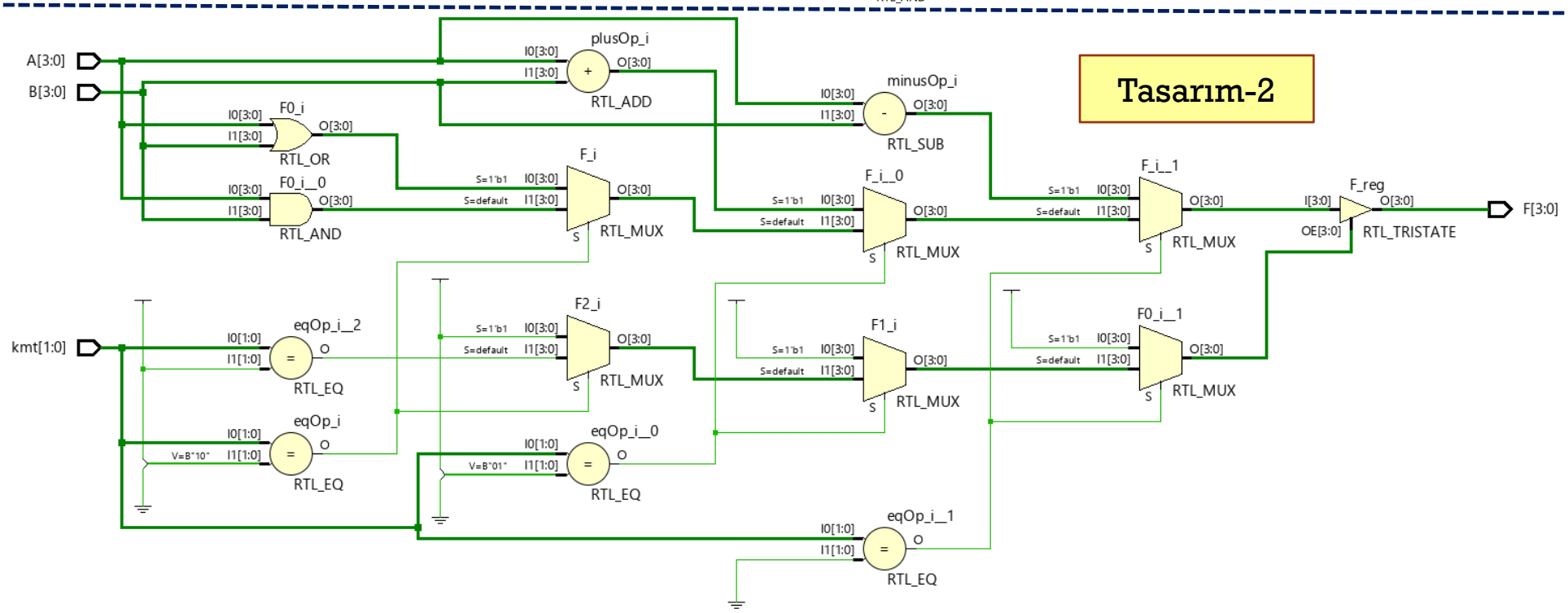
- Giriş verileri 4 bit olduğu durum için gerekli VHDL kodunu önce **With-Select** sonra da **When-Else** ifadeleri üzerinden yazalım.

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

Tasarım-1



Tasarım-2



8.10 Aritmetik Mantık Birimi

- Giriş verileri 4 bit olduğu durum için gerekli VHDL kodunu önce **With-Select** sonra da **When-Else** ifadeleri üzerinden yazalım.

Tasarım-3

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity ornek_AMB is
Port (A, B : in std_logic_vector (3 downto 0);
        kmt : in std_logic_vector (1 downto 0);
        F : out std_logic_vector (3 downto 0));
End ornek_AMB;
```

```
Architecture davranis of ornek_AMB is
Begin
   With kmt Select
      F <= std_logic_vector(unsigned(A) - unsigned (B))  When  "00",
           std_logic_vector(unsigned(A) + unsigned (B))  When  "01",
           A or B  When  "10",
           A and B  When  "11",
           "ZZZZ"  When  Others;
 End davranis;
```

8.10 Aritmetik Mantık Birimi

- Giriş verileri 4 bit olduğu durum için gerekli VHDL kodunu önce **With-Select** sonra da **When-Else** ifadeleri üzerinden yazalım.

Tasarım-4

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity ornek_AMB is
Port (A, B : in std_logic_vector (3 downto 0);
        kmt : in std_logic_vector (1 downto 0);
        F : out std_logic_vector (3 downto 0));
End ornek_AMB;
```

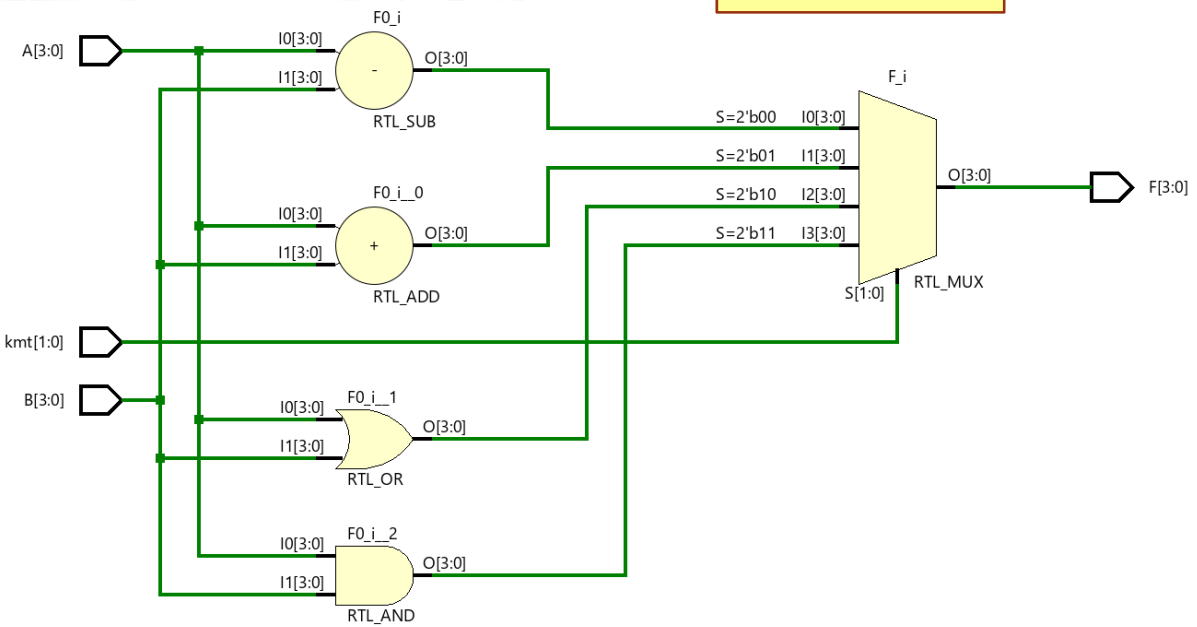
```
Architecture davranis of ornek_AMB is
Begin
   F <= std_logic_vector(unsigned(A) - unsigned (B)) When  kmt = "00" Else
          std_logic_vector(unsigned(A) + unsigned (B)) When  kmt = "01" Else
          A or B  When  kmt = "10" Else
          A and B  When  kmt = "11" Else
          "ZZZZ";
End davranis;
```

8.10 Aritmetik Mantık Birimi

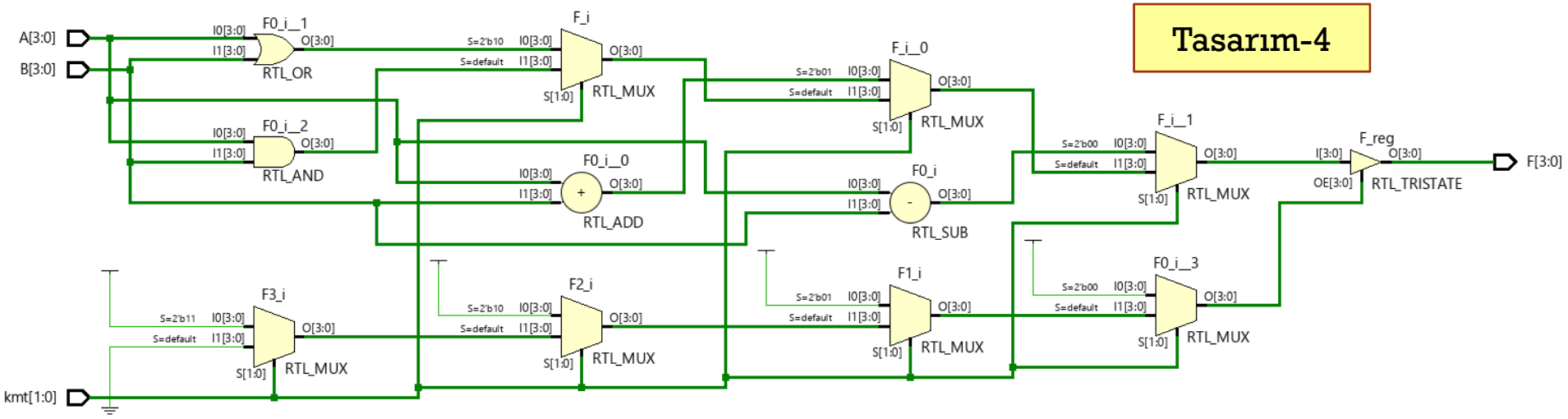
- Giriş verileri 4 bit olduğu durum için gerekli VHDL kodunu önce **With-Select** sonra da **When-Else** ifadeleri üzerinden yazalım.

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

Tasarım-3



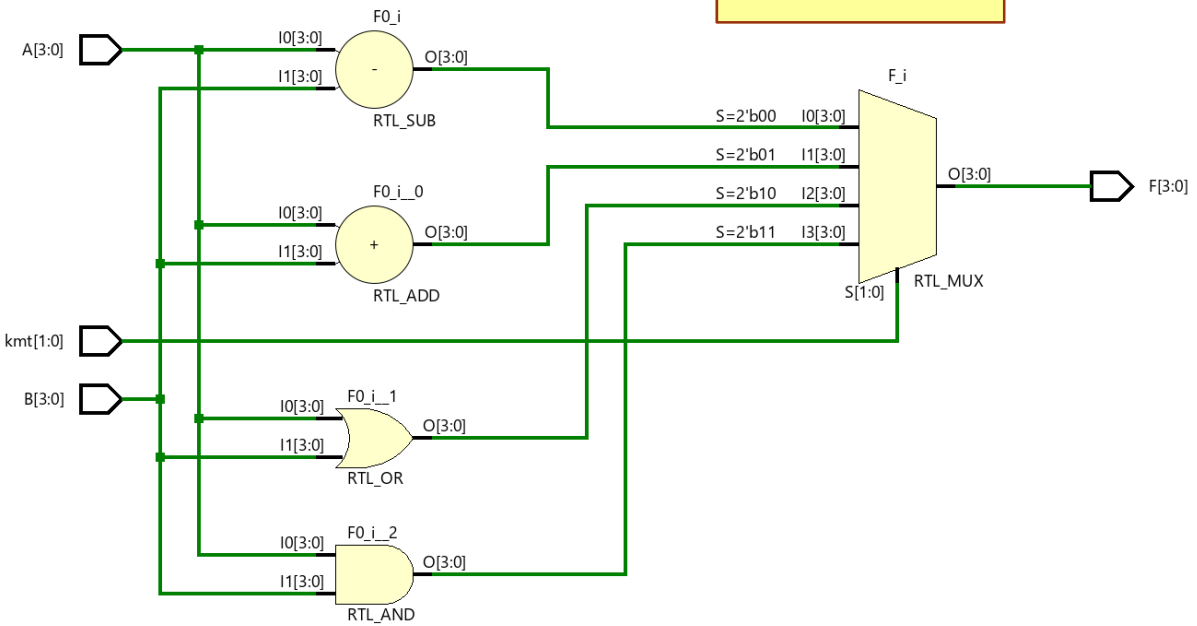
Tasarım-4



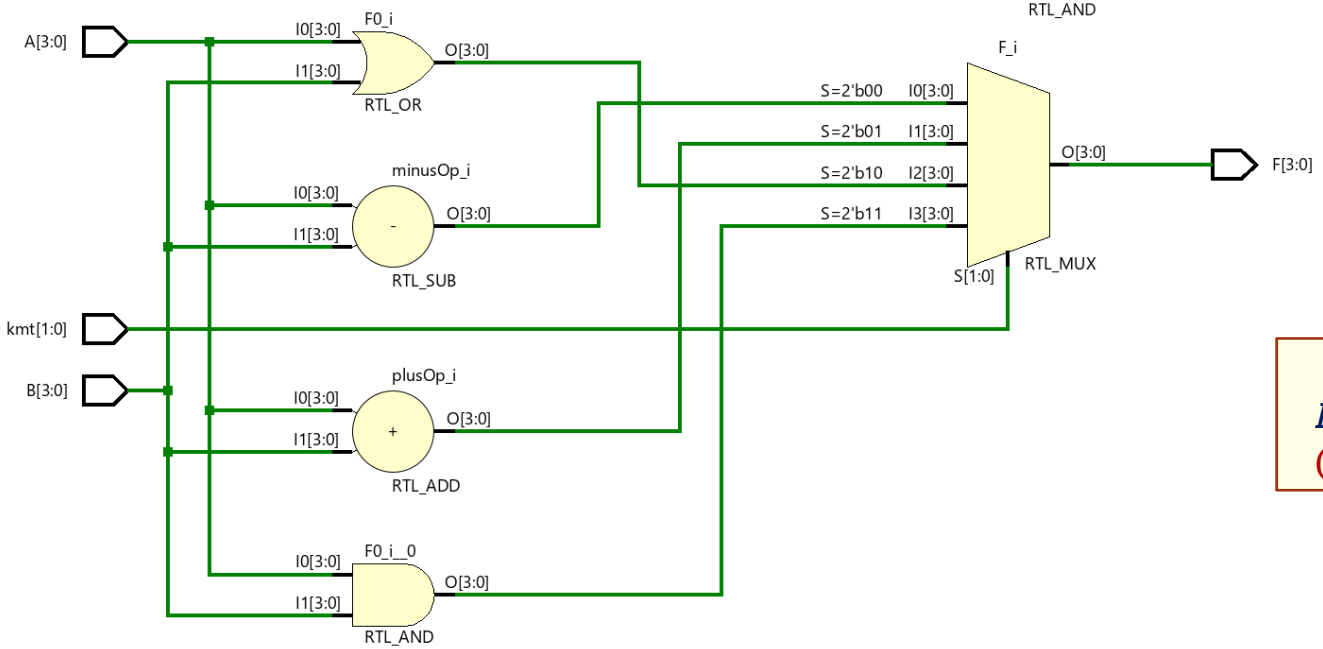
8.10 Aritmetik Mantık Birimi

- Giriş verileri 4 bit olduğu durum için gerekli VHDL kodunu önce **With-Select** sonra da **When-Else** ifadeleri üzerinden yazalım.

Tasarım-3



Tasarım-1

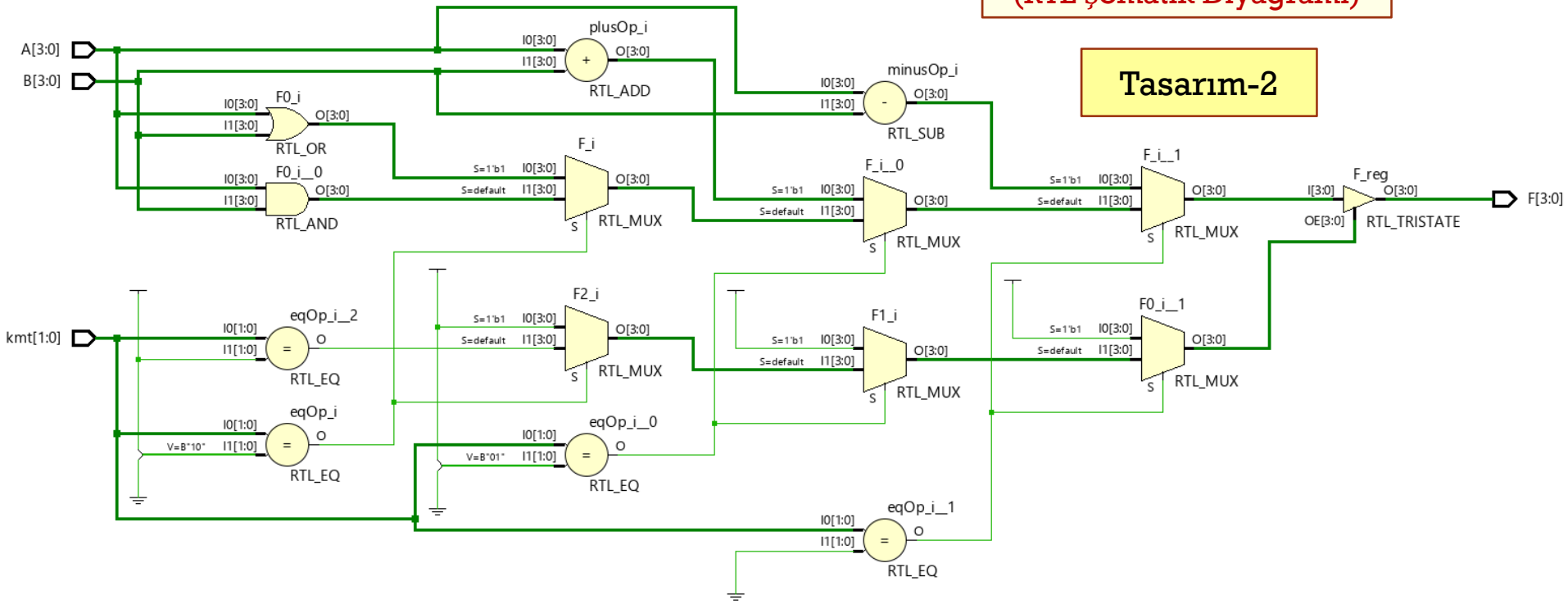


Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

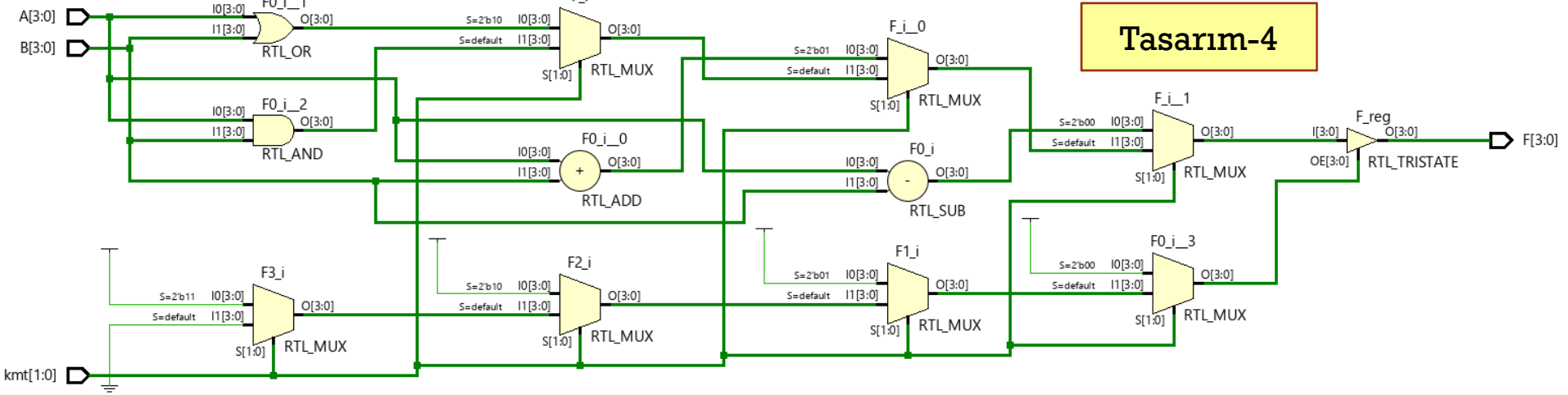
8.10 Aritmetik Mantık Birimi

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

Tasarım-2



Tasarım-4



8.11 For-Generate Ataması

- Tekrar edilmesi istenen basit işlemler için kullanılır. Tasarım noktasında **atama bloğu eş zamanlı (paralel) çalışır.**

- Sayısal devre karşılığı olarak, **bir mantıksal yapıyı (döngü üzerinden) çoğaltır** (***örn:*** sıralı XOR işleminin paralel tanımlanması).

- Bu atamayla **üretilen elemanlar birbirine benzer türetilir** ve **üretilecek kopya adedince döngü değişkeni için aralık ayarlanır.** Bu **döngü değişkeni, dizi elemanlarını indekslemek için kullanılır.**

- **Etiket kullanımı zorunludur.**

- ***For*** ifadesi **ardışıl işlem olmasına karşın**, ***Generate*** ifadesi ile kullanıldığında **elemanların eş zamanlı (paralel)** bağlantıları yapılır.

8.11 For-Generate Ataması

- **For-Generate** ataması için genel tanımlama aşağıdaki görüldüğü gibidir. Tanımlı olduğu **aralık statik (*generic* veya *sabit*) belirtilmelidir**.

```
Etiket: For parametre in baslangic_deger to bitis_deger generate
   Eşzamanlı ifade(ler)…
End generate Etiket;
```

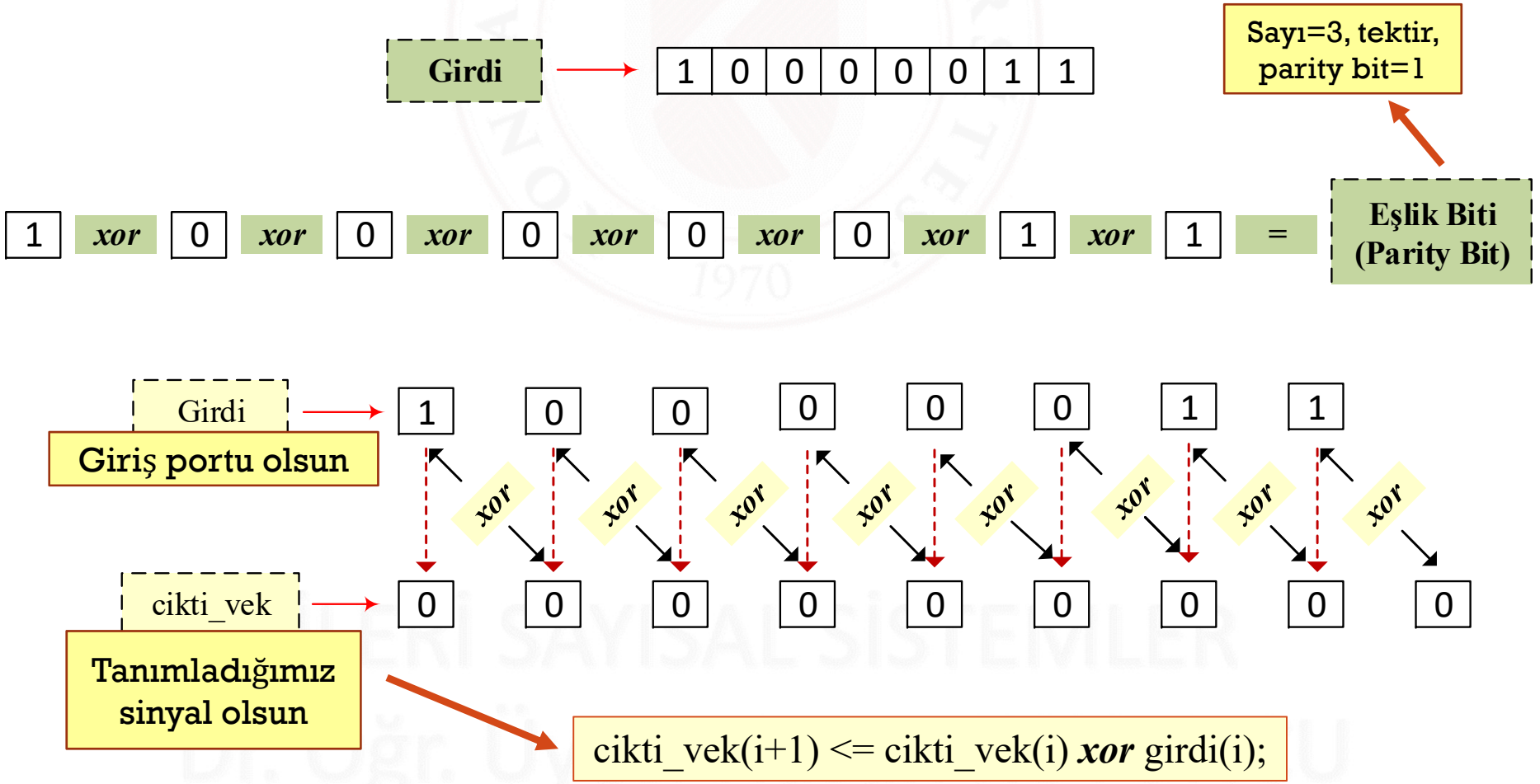
- **Kullanım alanı sınırlıdır.**

- **Çıkış portuna doğrudan atama yapılamaz** (vektörel çıkış için bit düzeyinde teker teker atama yapılabilir).

- **Yine giriş portundan doğrudan veri üzerinde işlem yapamaz** (vektörel giriş için bit düzeyinde teker teker okuma yapılabilir).

8.11 For-Generate Ataması

• 8 bitlik girdide **eşlik bitini** (**girdideki '1' lerin sayısı çift ise 0, tek ise 1 değerini alan bit – parity biti**) sağlayan VHDL kodunu **For-Generate** ifadesi üzerinden yazalım.



8.11 For-Generate Ataması

• 8 bitlik girdide **eşlik bitini** (**girdideki '1' lerin sayısı çift ise 0, tek ise 1 değerini alan bit – parity biti**) sağlayan VHDL kodunu **For-Generate** ifadesi üzerinden yazalım.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity eslik_biti is
Generic (n: natural := 8);
Port (girdi : in std_logic_vector (n-1 downto 0);
        es_cikis : out std_logic);
End eslik_biti;

Architecture davranis of eslik_biti is
Signal cikti_vek : std_logic_vector (n downto 0):= (others=> '0');
Begin

Eslik_biti: For i in 0 to n-1 generate
   cikti_vek(i+1) <= cikti_vek(i) xor girdi(i);
End generate Eslik_biti;

es_cikis <= cikti_vek(n);

End davranis;
```

cikti_vek(0) <= '0'; yazılırsa ilk değer sürekli lojik-0 olur. Sentezleme aşaması garanti altına alınır.

8.11 For-Generate Ataması

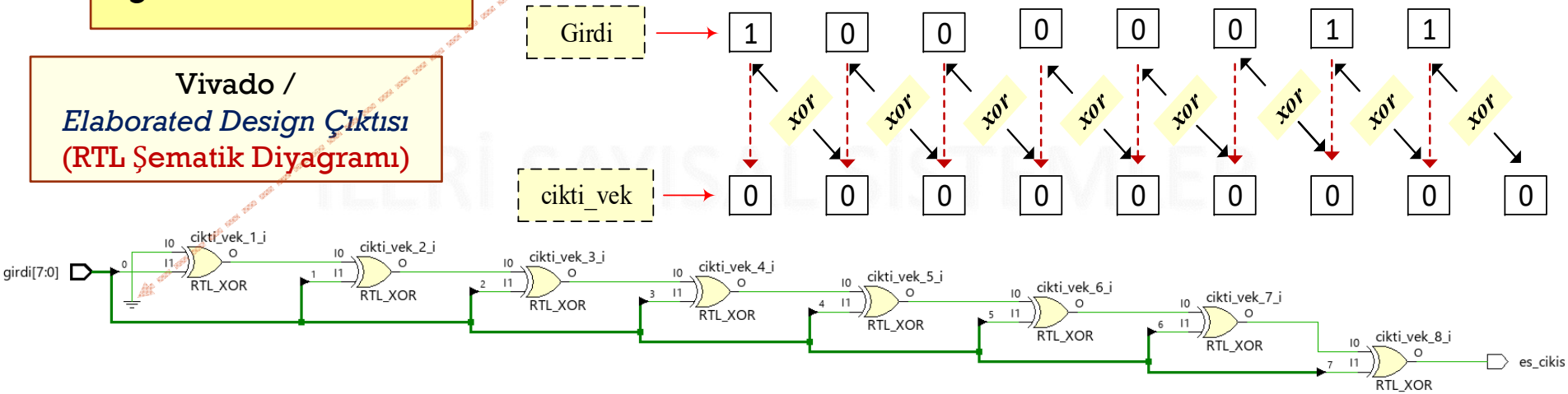
• 8 bitlik girdide **eşlik bitini** (**girdideki '1' lerin sayısı çift ise 0, tek ise 1 değerini alan bit – parity biti**) sağlayan VHDL kodunu **For-Generate** ifadesi üzerinden yazalım.

```
Architecture davranis of eslik_biti is
Signal cikti_vek : std_logic_vector (n downto 0):= (others=> '0');
Begin

Eslik_biti: For i in 0 to n-1 generate
   cikti_vek(i+1) <= cikti_vek(i) xor girdi(i);
End generate Eslik_biti;

es_cikis <= cikti_vek(n);

End davranis;
```

cikti_vek(0) <= '0'; yazılırsa ilk değer sürekli lojik-0 olur. Sentezleme aşaması garanti altına alınır.

Bu örnekte derleyici, bu komut yazılmamasına karşın sentezlemeyi doğru yapmış.

Vivado / *Elaborated Design Çıktısı* (RTL Şematik Diyagramı)



8.12 If-Generate Ataması

- Bir elemanın veya komponentin devreye eklenmesi **bir koşula bağlı olduğu durumda** kullanılır.

- Bu atama kapsamında **koşullu ifade için dallanma olmaz** (ardışıl kontrol mekanizması olarak işletilen ***Else*** veya ***Elsif*** kalıpları ***If*** sonrası kullanılamaz).

- **Eş zamanlı – koşullu durum için kullanılır** ve port bilgilerine dair şart kullanılamaz. **Statik bir parametre üzerinden koşul sağlanır** (bu nedenle nadir kullanılır). Koşul, sentezde bilinen (kesin) ***generic*** veya ***sabitler*** üzerinden tanımlanabilir.

- **If-Generate** ataması için genel tanımlama aşağıdaki görüldüğü gibidir.

Etiket: If (koşul) **generate**
  Eşzamanlı ifade(ler)…
End generate Etiket;

8.12 If-Generate Ataması

Örnek: Şifreleme amacıyla istenilen bir tasarımda, **6 veya üzeri çift sayılarda tanımlanabilen (n bitlik) girdi port bilgisi** için;

1) Bilginin ilk yarı bölümünün (**n/2-1** ***downto*** **0**), sabit bir anahtar olan ve giriş bit sayısının yarısına göre tanımlanan, ilk iki değeri **'0'** ve diğer değerleri **'1'** olan anahtar isimli **sabit** ile **XOR** işlemine tabi tutulması,

2) Bilginin ikinci yarı bölümündeki en büyük indisli terim harici bilgilerin (**n-2** ***downto*** **n/2**) terslenmesi,

3) Bilginin en büyük indisli (**n-1 indisli**) verisinin olduğu gibi aktarılması

gerekmektedir. Elde edilen bilgi **cikis** isimli çıkış portuna yazılacaktır. Gerekli VHDL kodunu **For-Generate** ve **If-Generate** ifadeleri üzerinden yazınız.

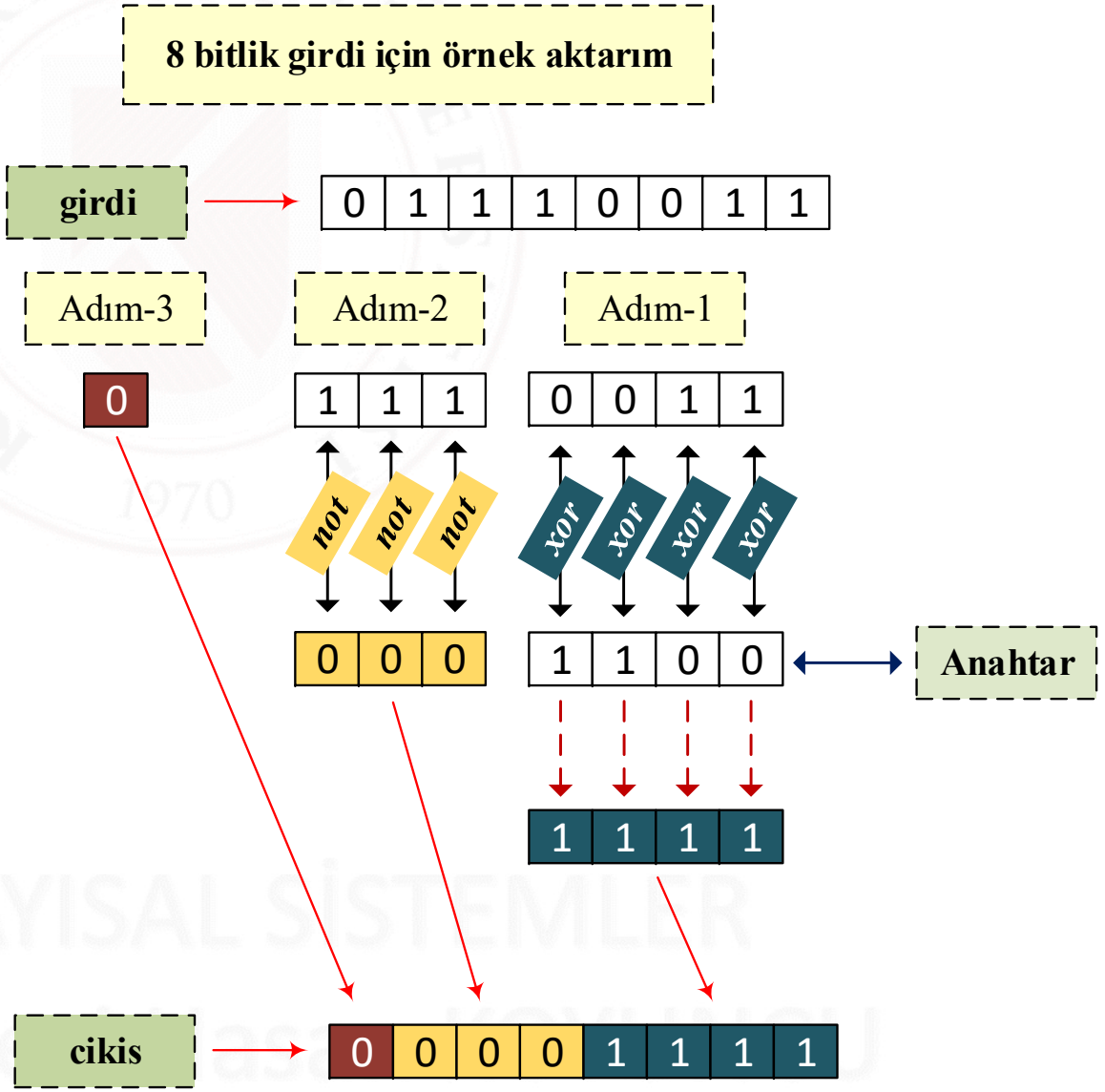
8.12 If-Generate Ataması

Girdi 8 bit iken;

1) Bilginin ilk yarı bölümünün (**n/2-1** ***downto*** **0**), sabit bir anahtar olan ve giriş bit sayısının yarısına göre tanımlanan, ilk iki değeri **'0'** ve diğer değerleri **'1'** olan anahtar isimli **sabit** ile **XOR** işlemine tabi tutulması,

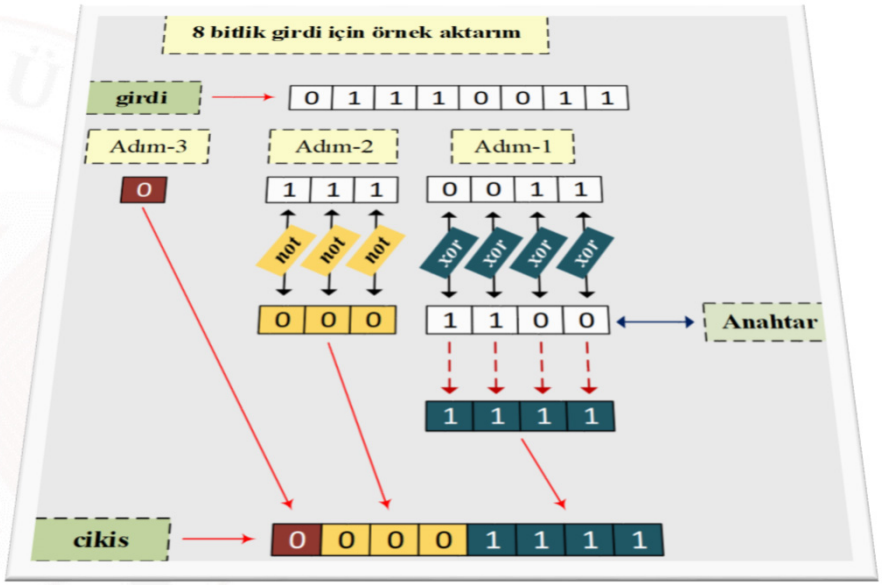
2) Bilginin ikinci yarı bölümündeki en büyük indisli terim harici bilgilerin (**n-2** ***downto*** **n/2**) terslenmesi,

3) Bilginin en büyük indisli (**n-1 indisli**) verisinin olduğu gibi aktarılması



8.12 If-Generate Ataması

Gerekli VHDL kodunu **For-Generate** ve **If-Generate** ifadeleri üzerinden yazınız.



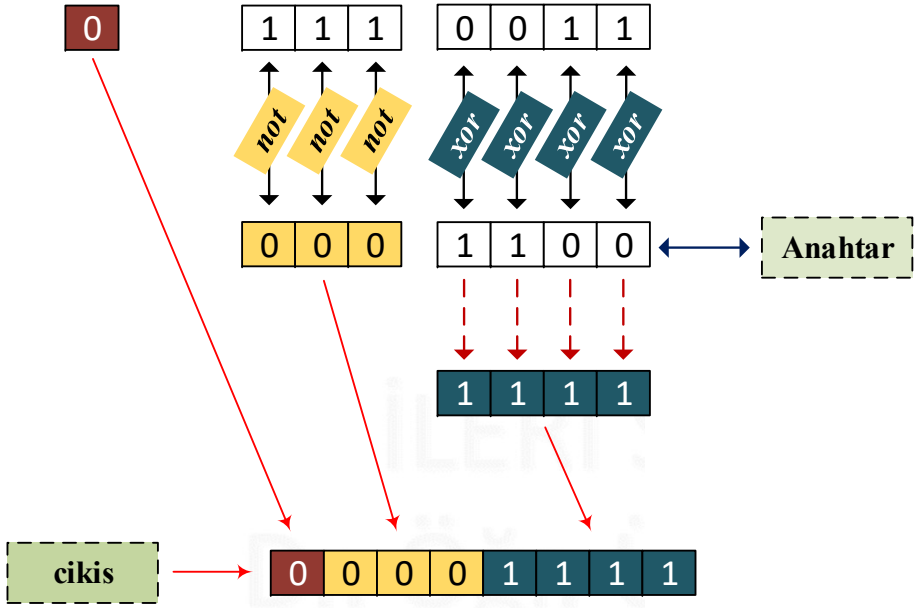

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity sifre1 is
Generic (n: natural := 8); -- tasarım iç yapısı gereği n çift sayı olmalı
Port (girdi : in std_logic_vector (n-1 downto 0);
      cikis : out std_logic_vector (n-1 downto 0));
End sifre1;
```

8.12 If-Generate Ataması

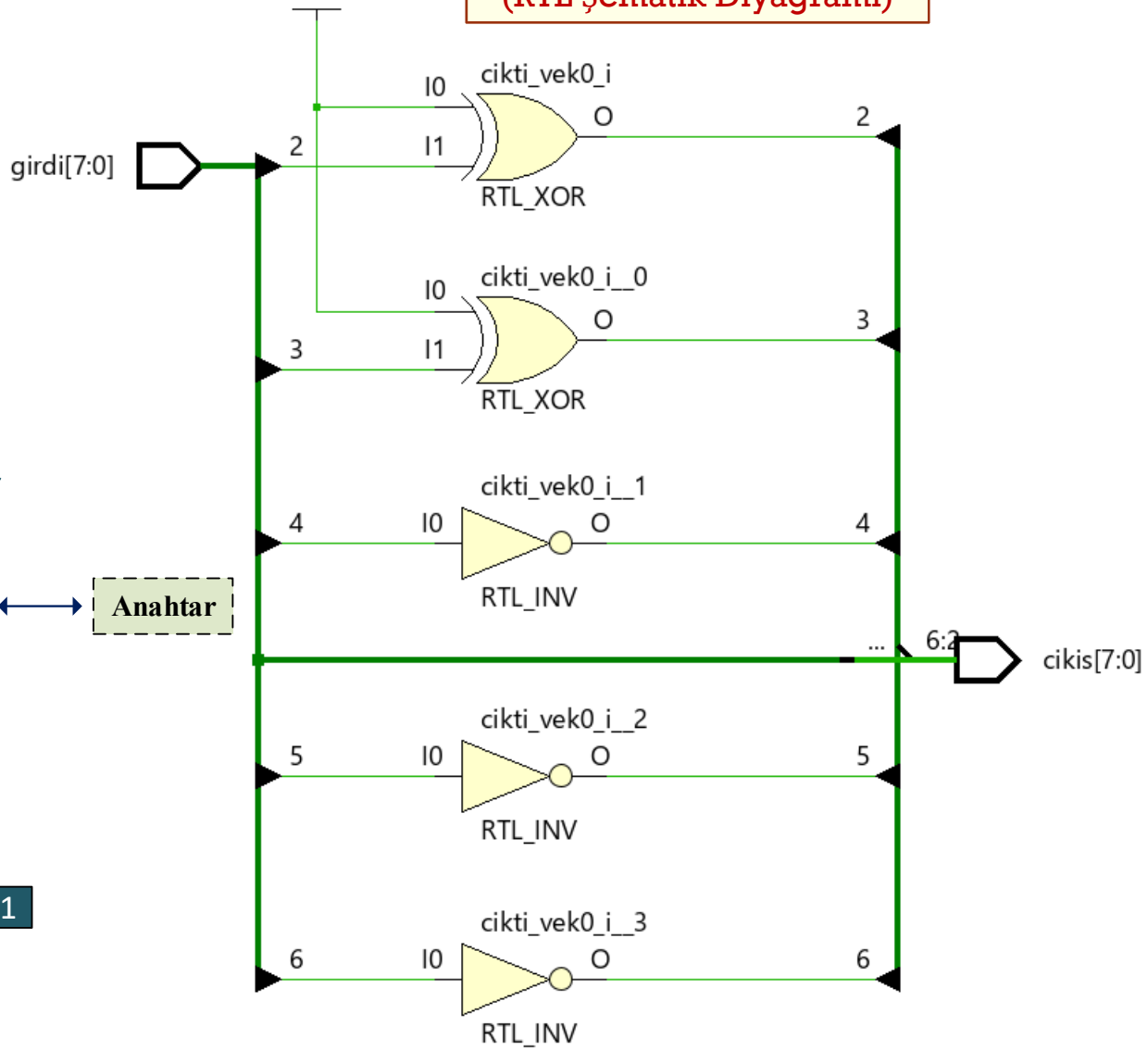
```
Architecture davranis of sifre1 is
Constant anahtar: std_logic_vector (n/2-1 downto 0):= (0=>'0', 1=>'0', others =>'1');
Signal cikti_vek : std_logic_vector (n-1 downto 0);
Begin

sifrele: For i in 0 to n-1 generate

   first_part: If (i < n/2) generate  -- n statik ve çift sayı olmalı
      cikti_vek(i) <= anahtar(i) xor girdi(i);
   End generate first_part;

   second_part: If (i >= n/2 and i<n-1) generate  -- n statik ve çift sayı olmalı
      cikti_vek(i) <= not girdi(i);
   End generate second_part;

   third_part: If (i = n-1) generate  -- n statik ve çift sayı olmalı
      cikti_vek(i) <= girdi(i);
   End generate third_part;

End generate sifrele;

cikis <= cikti_vek;

End davranis;
```

8.12 If-Generate Ataması

```
sifrele: For i in 0 to n-1 generate

   first_part: If (i < n/2) generate
      cikti_vek(i) <= anahtar(i) xor girdi(i);
   End generate first_part;

   second_part: If (i >= n/2 and i<n-1) generate
      cikti_vek(i) <= not girdi(i);
   End generate second_part;

   third_part: If (i = n-1) generate
      cikti_vek(i) <= girdi(i);
   End generate third_part;

End generate sifrele;
```



Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

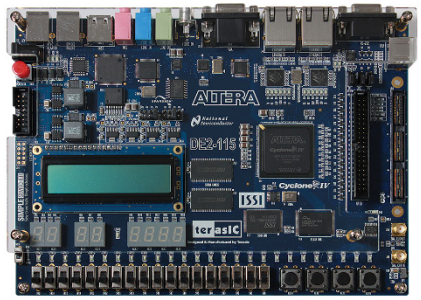


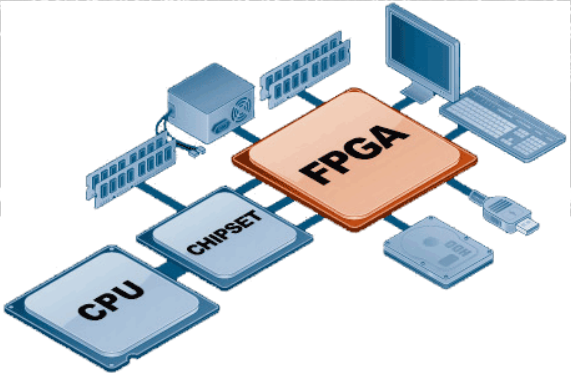
SONRAKİ DERS KONUSU

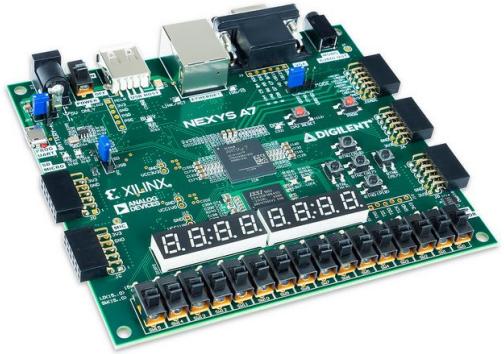
9- ARDIŞIL DEVRE TASARIMI

İLERİ SAYISAL SİSTEMLER

9- ARDIŞIL DEVRE TASARIMI







Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

9. ARDIŞIL DEVRE TASARIMI

- Kombinasyonel devrelerde **hafıza birimleri bulunmadığından** elde edilen çıkışlar süreklidir - kesintisizdir.

- Bu noktada **elde edilen bir çıktının hafızada tutulması gerekiyorsa** veya **başka bir devreye verinin aktarılması gerekli ise** ***Ardışıl Devre Tasarımı*** işletilmektedir.

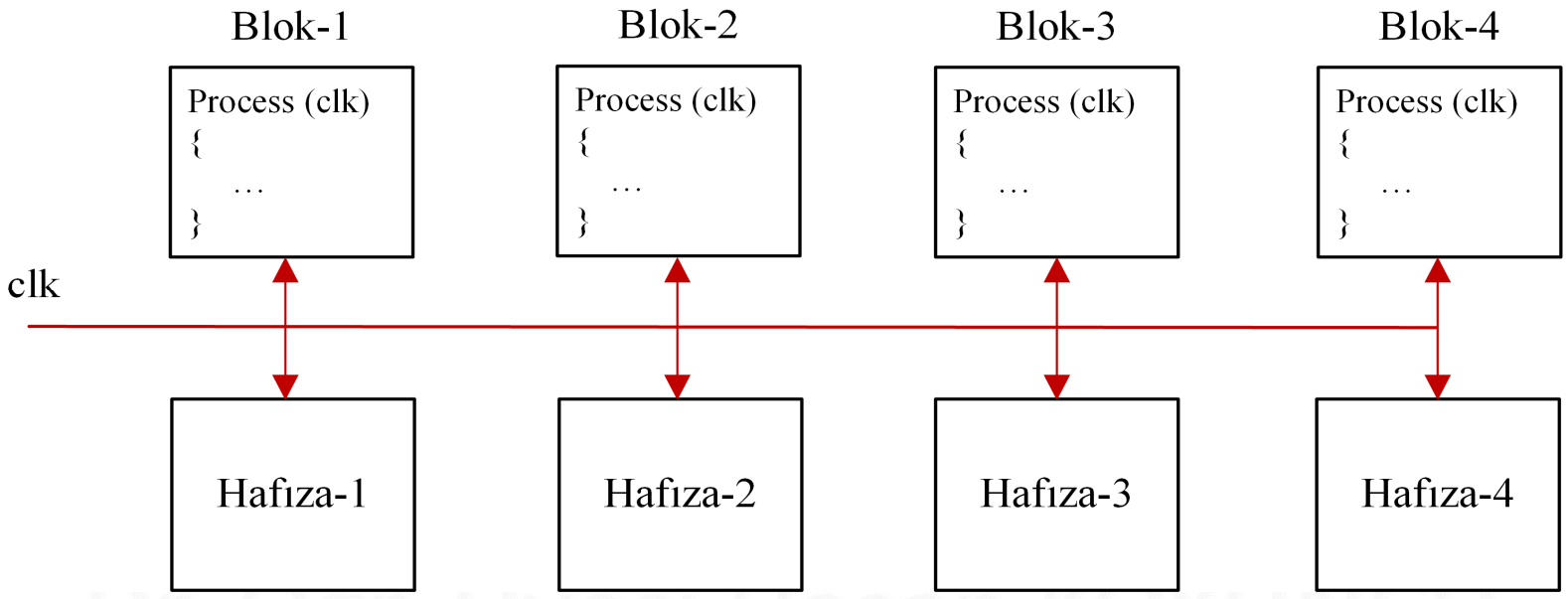
- VHDL dili kapsamında ardışıl devre tasarımı için farklı ifade ve yapılar bulunmaktadır.

- Bunlar temel olarak **process**, **function**, **procedure**, **case**, **loop**, **for** ve **while** şeklinde sıralanabilir.

9. ARDIŞIL DEVRE TASARIMI

- Ardışıl devre tasarımı **senkron** ve **asenkron** olarak iki tipte gerçekleştirilir.

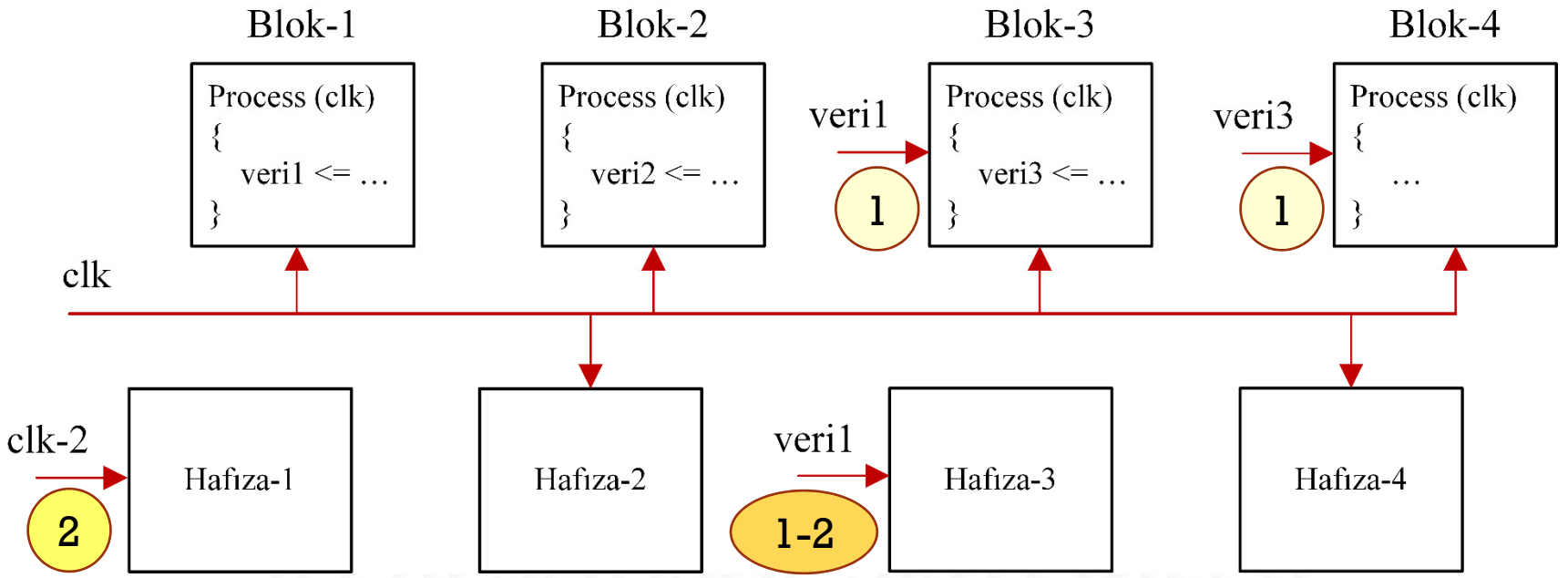
- Tasarım içindeki **bloklar aynı clock tetiği ile aktif hale geliyorsa gerçeklenen tasarım senkron devre karakteristiğindedir.**



Senkron devre örneği

9. ARDIŞIL DEVRE TASARIMI

- Tasarım içindeki **bloklar; başka bir blok işlemine bağlıysa** veya **farklı sinyal tetiklemeleri ile aktif hale geliyorlarsa gerçeklenen tasarım asenkron devre olarak bilinir.**



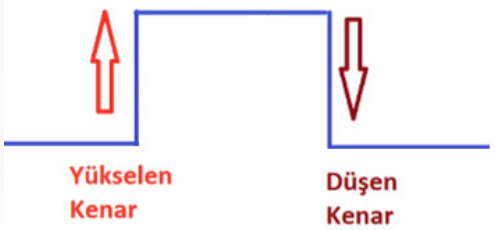
Asenkron devre örneği

9. ARDIŞIL DEVRE TASARIMI

- Hafıza biriminin bulunduğu ardışıl devre tasarımlarında saat darbesi (clock sinyali) kare dalga şeklindedir ve bu sinyal ile iki tip tetikleme işletilir:

  1-) Yükselen kenar tetiklemeli

  2-) Düşen kenar tetiklemeli



- Kare dalgada lojik-0'dan lojik-1'e geçiş yapıldığı kenar **yükselen**, lojik-1' den lojik-0' a geçiş yapıldığı kenar ise **düşen** kenar olarak bilinir.

- Yükselen ve düşen kenar tetiklemeleri de belirtilen bu durumlara göre meydana gelir.

- Sayısal tasarımda clock sinyalleri **process yapılarını** ve **hafıza birimlerini** aktive etmek için kullanılır.

9. ARDIŞIL DEVRE TASARIMI

- Saat darbesinin durumlarına dair aşağıdaki örnek tanımlamalar ile VHDL karşılık sağlanır.

// **yükselen kenar tetiklemeli işlem**	// **düşen kenar tetiklemeli işlem**
If rising_edge(clk) **then**	**If** falling_edge(clk) **then**
If clk'event **and** clk= '1' **then**	**If** clk'event **and** clk= '0' **then**

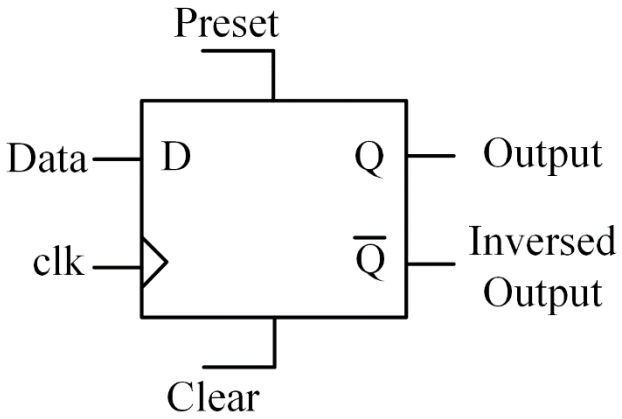
9.1 D-Flip Flop ve Kaydediciler

- Sayısal elektronikte 1 bitlik veri sıklıkla **D tipi FF** içinde tutulur, çünkü çalışma mantığı basittir ve D-FF'ler tasarım kolaylığı sağlarlar.

- Bu nedenle **sayısal tasarımda hafıza gereksinimleri sıklıkla D-FF'ler üzerinden sağlanır.**

9.1 D-Flip Flop ve Kaydediciler

- D-FF'lere dair doğruluk tablosu ve genel şema aşağıda görüldüğü gibidir.



D-FF genel şeması

D-FF doğruluk tablosu

$Q_{(t)}$	D	$Q_{(t+1)}$
0	0	0
0	1	1
1	0	0
1	1	1

- Yukarıda görülen D-FF yapısına dair gerekli VHDL kodu yazalım.

- Bu noktada **sistem çıkışı**; **clock tetiği ile sistem giriş durumuna**, **preset girişi durumuna** ve **clear girişi durumuna** göre elde edilmelidir.

9.1 D-Flip Flop ve Kaydediciler

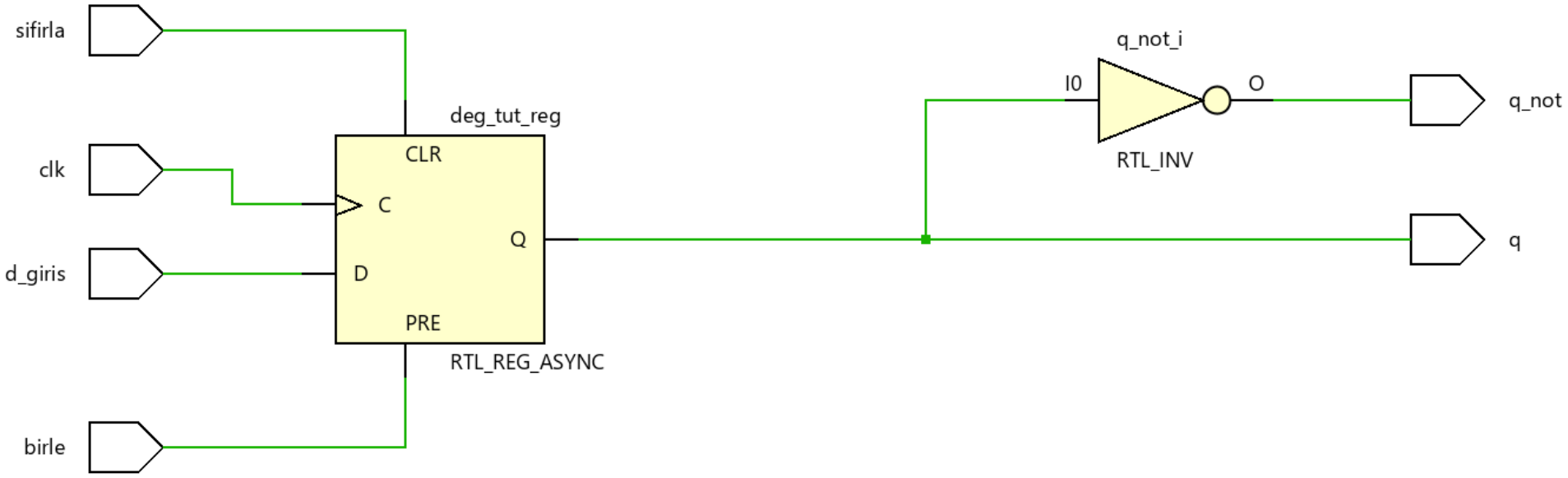
- Bu noktada **sistem çıkışı**; **clock tetiği ile sistem giriş durumuna**, **preset girişi durumuna** ve **clear girişi durumuna** göre elde edilmelidir.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity dtypeff is
Port (clk, d_giris, birle, sifirla : in std_logic;
        q, q_not : out std_logic);
End dtypeff;

Architecture davranis of dtypeff is
Signal deg_tut: std_logic := '0';
Begin
   Process (sifirla, birle, clk)
   Begin
     If sifirla = '1' then
        deg_tut <= '0';
     Elsif birle = '1' then
        deg_tut <= '1';
     Elsif clk'event and clk= '1' then
        deg_tut <= d_giris;
     End if;
  End process;
  q <= deg_tut;
  q_not <= not deg_tut;
End davranis;
```

9.1 D-Flip Flop ve Kaydediciler

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

- FPGA'deki temel D-FF yapıları genellikle **sadece Q üretir**. **$\bar{Q}$ istenirse,** sentez aracı çıkışa otomatik olarak bir **NOT kapısı ekler**.



9.2 Process Yapısı

▪Ardışıl devre tasarımında **sıralı ifadelerin işletilmesi** genelde **process** ifadesi ile sağlanır.

▪ **Process** ifadesi sonrası parantez içine bu bloğu tetikleyecek olan ifadeler (***duyarlılık listesi***) yazılır.

▪ Örneğin; D-FF için **clear**, **preset** ve **clock** bu liste içine yazılabilir.

▪ Söz konusu **liste içinde belirtilen sinyallerin değeri değiştiğinde** (**etkin olduklarında**) **process bloğu tetiklenecektir.**

▪ **Etiket başlığı** ise opsiyonel olarak kullanılabilir.

9.2 Process Yapısı

- **Etiket başlığı** ise opsiyonel olarak kullanılabilir.

```
Etiket: Process (duyarlılık listesi)
   variable [değişken tanımlama]
Begin
   Basit sinyal atamaları
   If-else kullanımı
   Döngü (while, for, loop) kullanımı
   Case kullanımı …
End process;
```

Örnek 9.1: Akım (A), gerilim (V) ve sıcaklık (T) kontrolü yapılacak bir elektronik kart için, belirtilen parametrelerin kontrol girişi olduğu bir sistem tasarlanacaktır.

Sistem kapsamında kontrol birimi için herhangi **bir değer sınırı aştığında ikaz-1**, **iki değer sınırı aştığında ikaz-2**, **her üç değer sınırı aştığında ise ikaz-3** çıkışı etkin olacak, gerekli üç çıkış **clock tetiği ile senkron** elde edilecektir.

Gerekli çıkışları **doğruluk tablosu üzerinden elde ederek** gerekli VHDL kodunu yazınız.

9.2 Process Yapısı

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity kontrol_birimi is
Port (A,V,T,clk : in std_logic;
        ikaz_1,ikaz_2,ikaz_3 : out std_logic);
End kontrol_birimi;

Architecture davranis of kontrol_birimi is
Begin
   Process (clk)
   Begin
      If rising_edge(clk) then
         ikaz_1 <= ((not A) and (V xor T)) or (A and (not V) and (not T));
         ikaz_2 <= (T and (A xor V)) or (A and V and (not T));
         ikaz_3 <= (A and V and T);
      End if;
   End process;
End davranis;
```

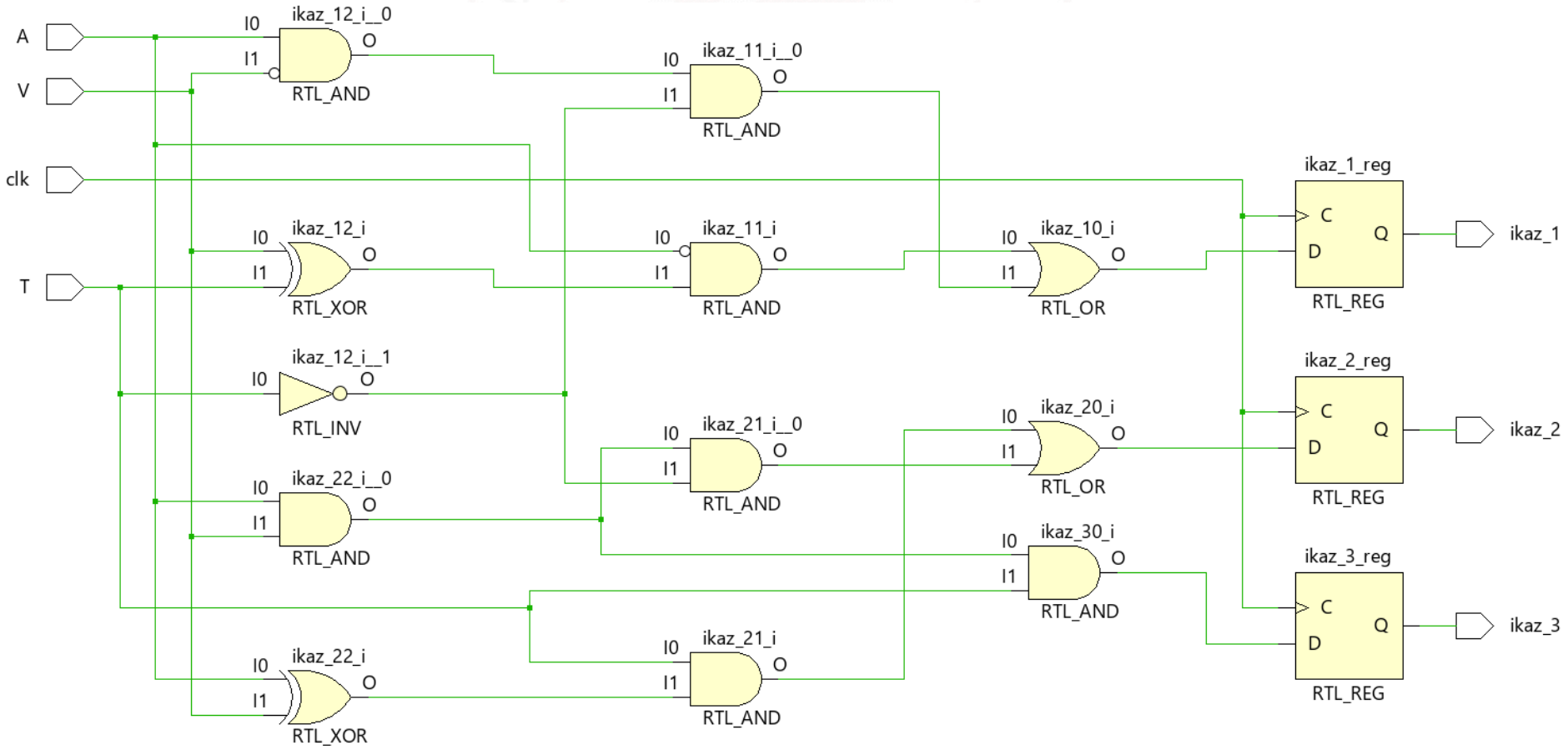
Doğruluk tablosu

A	V	T	ikaz-1	ikaz-2	ikaz-3
0	0	0	0	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	0	1	0
1	1	1	0	0	1

$$ikaz\_1 = \bar{A}\bar{V}T + \bar{A}V\bar{T} + A\bar{V}\bar{T}$$
$$= \bar{A}(V \oplus T) + A\overline{VT}$$
$$ikaz\_2 = \bar{A}VT + A\bar{V}T + AV\bar{T}$$
$$= T(A \oplus V) + AV\bar{T}$$
$$ikaz\_3 = AVT$$

9.2 Process Yapısı

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



9.3 If-Else Yapısı

▪**Process** yapısı içinde, **belli koşul(lar)a göre tasarımı yönlendirmek** için kullanılan **sıralı ifade komutlarından birisidir.**

▪ Karşılaştırma yapısı olarak da atfedilir.

▪ **If** komutu ile **en az bir şartın** yazılması gerekir.

▪ **İki durum olduğunda If-Else** ifadesi (*Else → genel gerçekleştirilmesi gereken işlem için*) veya **If-Elsif** ifadesi kullanılır.

▪ **İkiden fazla şart olduğunda If-Elsif-Elsif** komutu kullanılabilmekte, **If-Elsif-Else** yapısı ise **akışın sağlanmasında meydana gelebilecek hataların önlenmesini sağlamak amacıyla** (*Else → genel gerçekleştirilmesi gereken işlem için*) işletilebilmektedir.

9.3 If-Else Yapısı

• **If** komutu ile **en az bir şartın** yazılması gerekir. **İki durum olduğunda If-Else** ifadesi (*Else → genel gerçekleştirilmesi gereken işlem için*) veya **If-Elsif** ifadesi kullanılır. **İkiden fazla şart olduğunda If-Elsif-Elsif** komutu kullanılabilmekte, **If-Elsif-Else** yapısı ise **akışın sağlanmasında meydana gelebilecek hataların önlenmesini sağlamak amacıyla** (*Else → genel gerçekleştirilmesi gereken işlem için*) işletilebilmektedir.

If ve türevlerinin kullanım örnekleri

If	*If-Else*	*If-Elsif*	*If-Elsif-Elsif*	*If-Elsif-Else*

If koşul_1 **then**
İfade(ler)…
End
	If koşul_1 **then**
	İfade(ler)…
	Else
	İfade(ler)…
	End
		If koşul_1 **then**
		İfade(ler)…
		Elsif koşul_2 **then**
		İfade(ler)
		End
			If koşul_1 **then**
			İfade(ler)…
			Elsif koşul_2 **then**
			İfade(ler)
			Elsif koşul_3 **then**
			İfade(ler)…
			Elsif koşul_4 **then**
			İfade(ler)…
			End
				If koşul_1 **then**
				İfade(ler)…
				Elsif koşul_2 **then**
				İfade(ler)
				Elsif koşul_3 **then**
				İfade(ler)…
				Else
				İfade(ler)…
				End

9.3 If-Else Yapısı

Örnek 9.2: Bir arabanın **motor**, **emniyet kemeri** ve **kapı** durumları sensörler üzerinden analiz edilerek **ikaz** devreye alınacaktır. Sırasıyla **motor**, **emniyet** ve **kapi** giriş portlarından sensör bilgileri alınmaktadır.

Motor çalışıyor (lojik-1) ise emniyet kemerinin takılmamış (lojik-0) veya kapılardan birisi açık (lojik-0) olması durumunda ikaz devreye girecektir.

Opsiyonel olarak sistemde arıza gelebileceği düşünülerek **resetleme** işlemi de tasarıma eklenecektir. Gerekli VHDL kodunu yazınız.

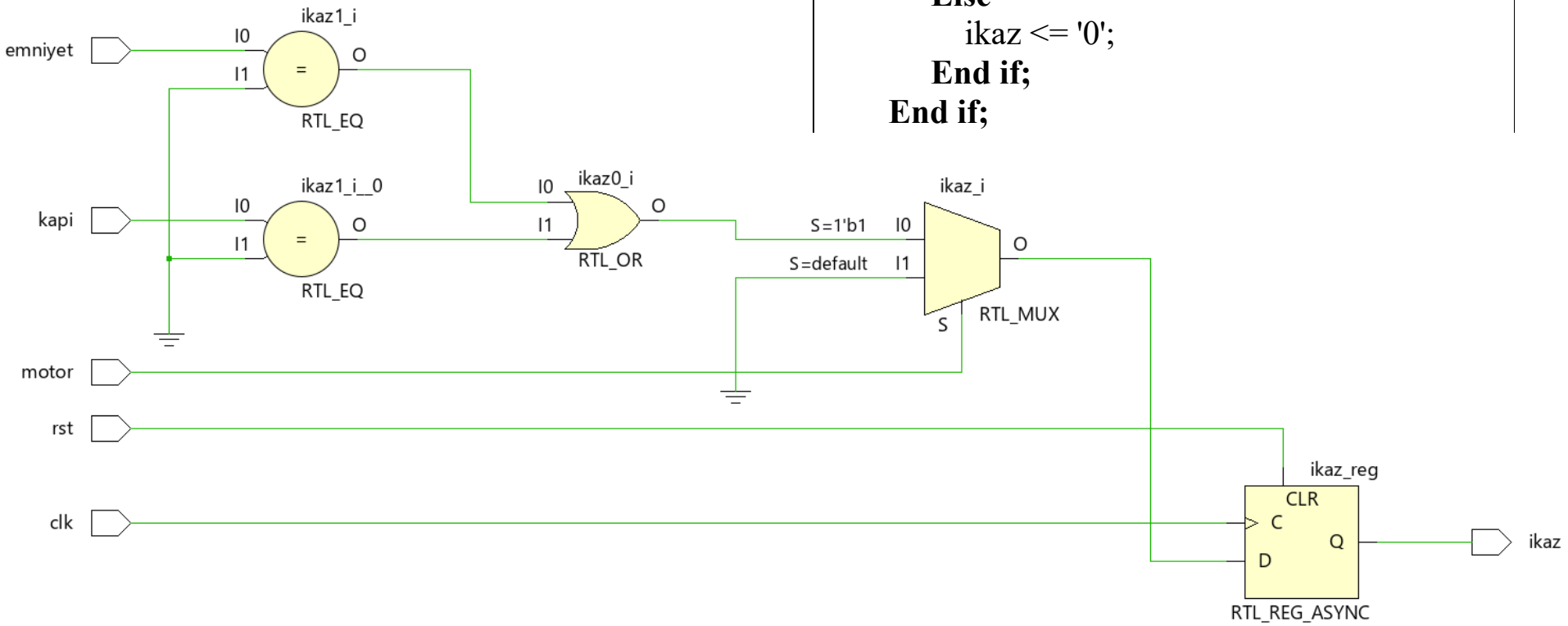
```
ikaz <= motor and ((not (emniyet)) or (not (kapi)));
```

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity kontrol_birimi is
Port (motor, emniyet, kapi, clk, rst : in std_logic;
      ikaz : out std_logic);
End kontrol_birimi;

Architecture davranis of kontrol_birimi is
Begin
  Process (clk, rst)
  Begin
    If (rst= '1') then
      ikaz <= '0';
    Elsif rising_edge(clk) then
      If (motor = '1') then
        If (emniyet = '0' or kapi = '0') then
          ikaz <= '1';
        Else
          ikaz <= '0';
        End if;
      Else
        ikaz <= '0';
      End if;
    End if;
  End process;
End davranis;
```

9.3 If-Else Yapısı

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

```
If (rst= '1') then
    ikaz <= '0';
Elsif rising_edge(clk) then
    If (motor = '1') then
        If (emniyet = '0' or kapi = '0') then
            ikaz <= '1';
        Else
            ikaz <= '0';
        End if;
    Else
        ikaz <= '0';
    End if;
End if;
```



9.4 Case Yapısı

- **Case** yapısı sıralı ifadelerden birisi olup, **kontrol devrelerinde** ve **durum makinelerinde** sıklıkla kullanılır.

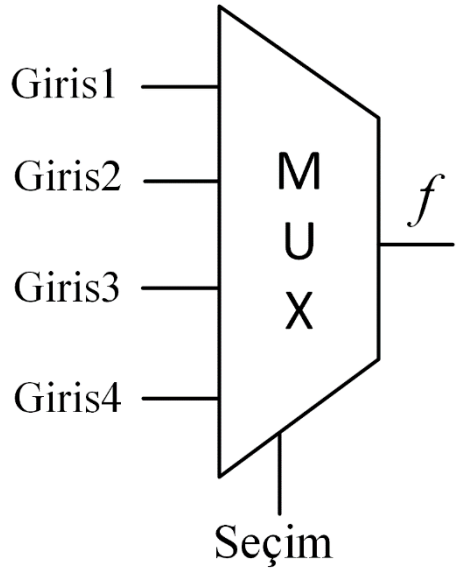
- **If-Else** ifadesinin yerine de kullanılan bu yapı, **bir değişken veya sinyalin alabileceği değerlere göre sağlanacak alternatif işlem seçiminde** kullanılır.

- Case yazım şekli aşağıda görüldüğü gibidir.

```
Case degisken_veya_sinyal is
   When deger_1 | deger_2 | deger_3 => -- birden fazla değer
      ifadeler
   When deger_4 to deger_6 => -- değer aralığı
      ifadeler
   When deger_7 => -- tek bir değer
      ifadeler
   When Others =>
      ifadeler
End Case;
```

9.4 Case Yapısı

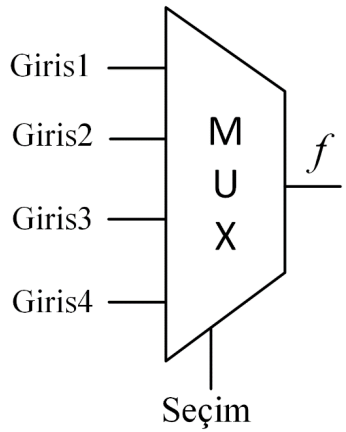
Örnek 9.3: Şekilde görülen **4:1 çoklayıcı** için gerekli VHDL kodunu **Case** ifadesi üzerinden yazınız.



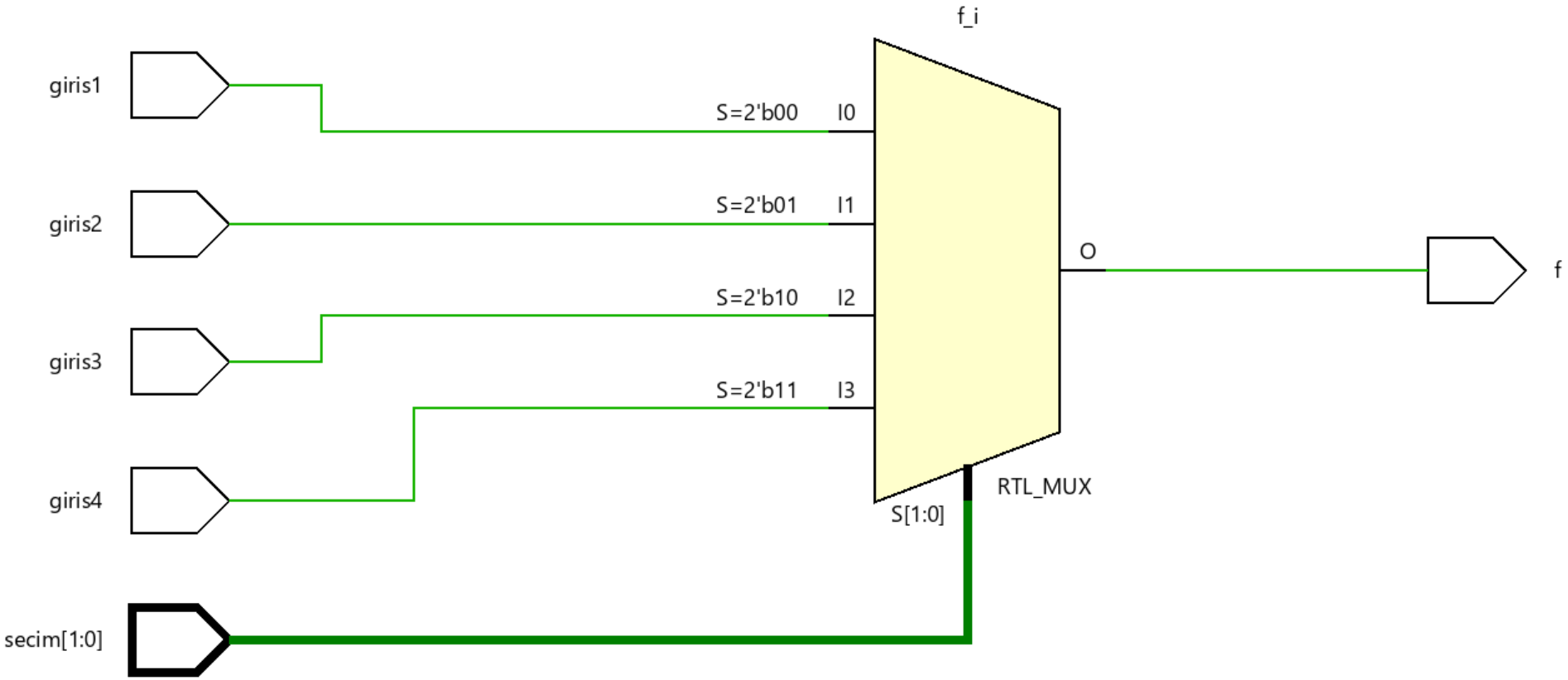

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity colkayici_4e1 is
Port (giris1, giris2, giris3, giris4 : in std_logic;
      secim : in std_logic_vector (1 downto 0);
      f : out std_logic);
End colkayici_4e1;

Architecture davranis of colkayici_4e1 is
Begin
   Process (secim, giris1, giris2, giris3, giris4)
   Begin
      Case secim is
         When "00" => f <= giris1;
         When "01" => f <= giris2;
         When "10" => f <= giris3;
         When "11" => f <= giris4;
      End case;
   End process;
End davranis;
```

9.4 Case Yapısı

```
Process (secim, giris1, giris2, giris3, giris4)
Begin
   Case secim is
      When "00" => f <= giris1;
      When "01" => f <= giris2;
      When "10" => f <= giris3;
      When "11" => f <= giris4;
   End case;
End process;
```



Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



9.4 Case Yapısı

Örnek 9.4: Üretim bandında 4 farklı genişlikte kavanozun çapı fotoseller vasıtasıyla ölçülecektir.

- **Küçük kavanoz 1 fotoseli**, **orta kavanoz 2 fotoseli**, **büyük kavanoz 3 fotoseli** ve **ekstra büyük kavanoz 4 fotoseli** kesecek şekilde sistem tasarımı sağlanmıştır.

- Bantta kavanoz yokken fotosel bilgisi "0000" olarak gelmektedir ve bu durumda çıkışa **kavanoz yok** bilgisi iletilmelidir.

- Diğer durumlar ise **üretim hatası** olarak çıkışa aktarılacaktır.

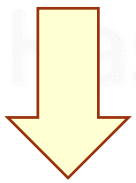
- Sistem çıkışında **kavanoz çap büyüklükleri ve diğer durumlar ikili tabanda ifade edilecek** şekilde gerekli VHDL kodunu **Case** ifadesi üzerinden yazınız.

9.4 Case Yapısı

Durum / Giriş	İkili Karşılık
Kavanoz yok	000
Küçük kavanoz	001
Orta kavanoz	010
Büyük kavanoz	011
Extra büyük kavanoz	100
Üretim Hatası	101

Şeklinde ifade edelim.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity kav_kontrol is
Port (clk, rst : in std_logic;
        fotosel : in std_logic_vector (3 downto 0);
        cikis : out std_logic_vector (2 downto 0));
End kav_kontrol;
```



9.4 Case Yapısı

```
Architecture davranis of kav_kontrol is
Begin
   Process (clk, rst)
   Begin
      If (rst= '1') then
         cikis <= "000";
      Elsif rising_edge (clk) then
         Case fotosel is
            When x"0" =>
               cikis <= "000";    -- kavanoz yok
            When x"1" | x"2" | x"4" | x"8" =>
               cikis <= "001";    -- küçük kavanoz
            When x"3" | x"6" | x"C" =>
               cikis <= "010";    -- orta kavanoz
            When x"7" | x"E" =>
               cikis <= "011";    -- büyük kavanoz
            When x"F" =>
               cikis <= "100";    -- extra büyük kavanoz
            When others =>
               cikis <= "101";     -- üretim hatası
       End case;
      End if;
  End process;
End davranis;
```

Durum / Giriş	İkili Karşılık
Kavanoz yok	000
Küçük kavanoz	001
Orta kavanoz	010
Büyük kavanoz	011
Extra büyük kavanoz	100
Üretim Hatası	101

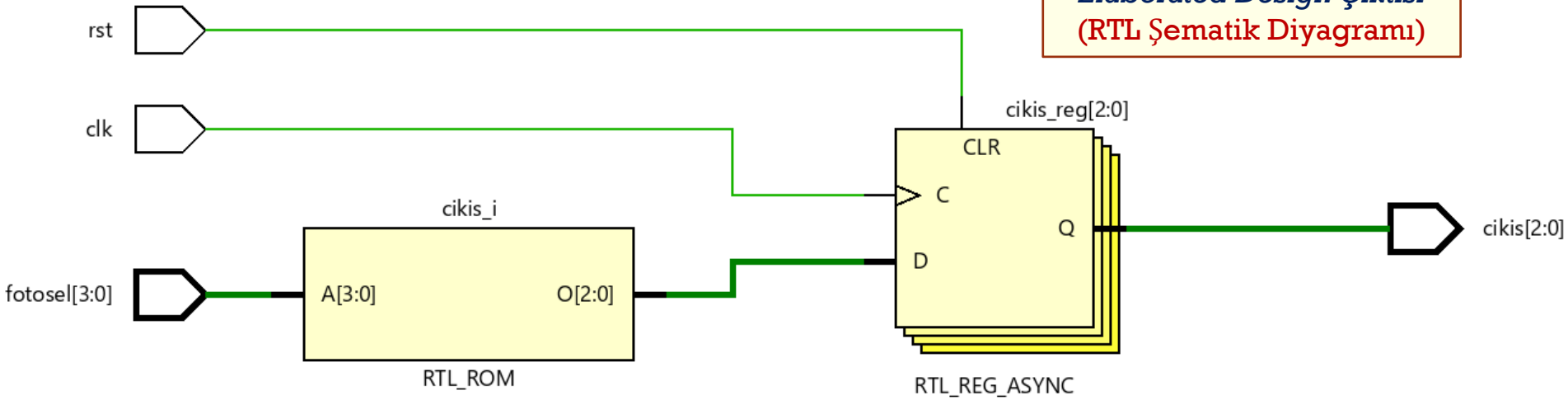
9.4 Case Yapısı

Durum / Giriş	İkili Karşılık
Kavanoz yok	000
Küçük kavanoz	001
Orta kavanoz	010
Büyük kavanoz	011
Extra büyük kavanoz	100
Üretim Hatası	101

```
Case fotosel is
   When x"0" =>
      cikis <= "000";    -- kavanoz yok
   When x"1" | x"2" | x"4" | x"8" =>
      cikis <= "001";    -- küçük kavanoz
   When x"3" | x"6" | x"C" =>
      cikis <= "010";    -- orta kavanoz
   When x"7" | x"E" =>
      cikis <= "011";    -- büyük kavanoz
   When x"F" =>
      cikis <= "100";    -- extra büyük kavanoz
   When others =>
      cikis <= "101";    -- üretim hatası
End case;
```

- Derleyici, **4 bitlik fotoset girdisini** ve buna göre **gerekli 6 farklı çıktı üretimini** ***ROM yapısı temelli*** modelledi.

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



9.5 For Yapısı

- İç bölümünde tanımlı kodları belirli bir sayıda (iterasyon sayısınca) çalıştırır.

- İçinde **bildirilen parametre için**; **tanımlı aralığın başlangıç değerini parametreye aktaran**, bu değeri **her döngüde 1 artıran** ve **bitiş değerinde döngüyü sonlandıran** yapıdır.

- Aralıkta belirtilen **başlangıç** ve **bitiş değerleri farkına +1 eklendiğinde iterasyon sayısı tespit edilir.**

- Tanımlamada etiket başlığı ***opsiyonel*** olarak kullanılabilir.

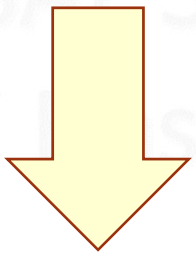
Etiket: **For** parametre **in** baslangic_deger **to** bitis_deger **loop**
İfade(ler)…
End loop;

9.5 For Yapısı

Örnek 9.5: 32 bitlik bir sayıda kaç adet lojik-1 olduğunu tespit etmek için gerekli VHDL kodunu ***For*** ifadesi üzerinden yazınız.

Tasarım-1

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Use ieee.std_logic_unsigned.all;

Entity birleri_say is
Generic (n: natural := 32);
Port (clk, rst : in std_logic;
        girdi : in std_logic_vector (n-1 downto 0);
        cikti : out std_logic_vector (5 downto 0)); -- en fazla 32 olabilir
End birleri_say;
```



9.5 For Yapısı

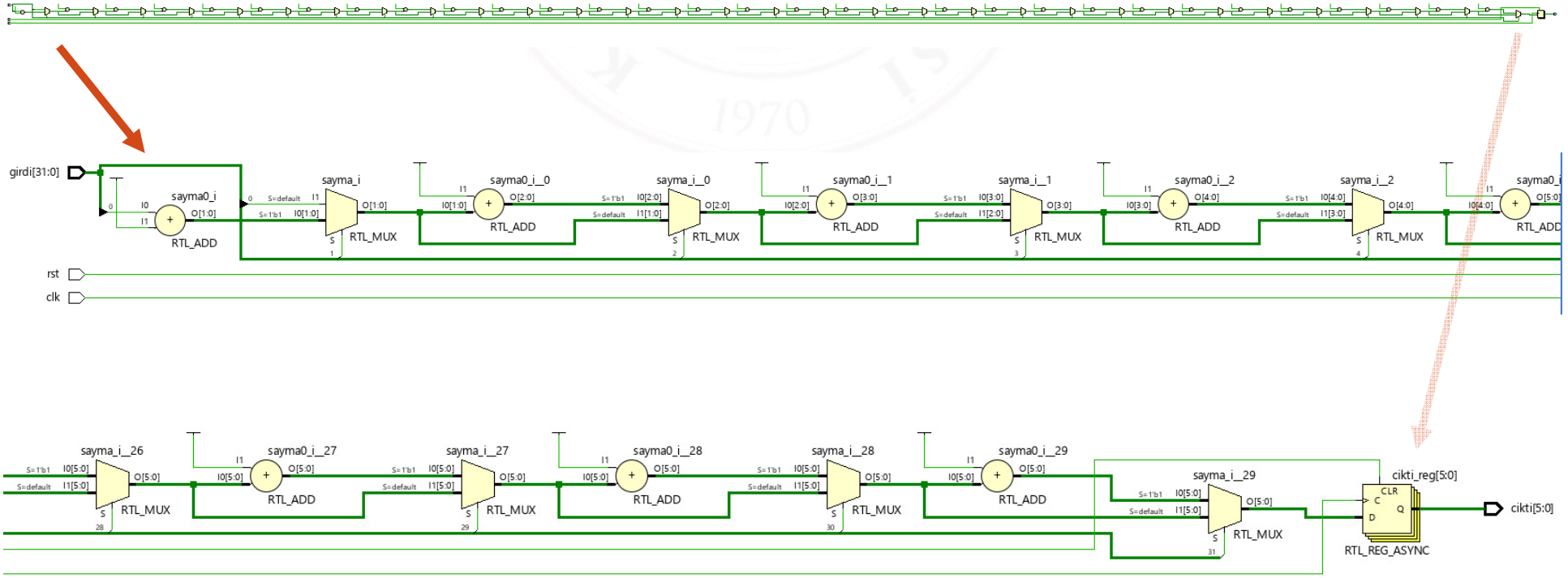
Tasarım-1

```
Architecture davranis of birleri_say is
Begin
   Process (clk, rst)
      variable sayma: integer range 0 to n;
   Begin
      If (rst= '1') then
         sayma := 0;
         cikti <= (others=>'0');
      Elsif rising_edge (clk) then
         sayma := 0;
         For i in 0 to n-1 loop
            If girdi(i) = '1' then
               sayma := sayma + 1;
            End if;
         End loop;
         cikti <= conv_std_logic_vector (sayma, cikti'length);
      End if;
   End process;
End davranis;
```

9.5 For Yapısı

Tasarım-1

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

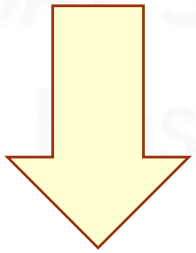


9.5 For Yapısı

Örnek 9.5: 32 bitlik bir sayıda kaç adet lojik-1 olduğunu tespit etmek için gerekli VHDL kodunu ***For*** ifadesi üzerinden yazınız.

Tasarım-2

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity birleri_say is
Generic (n: natural := 32);
Port (clk, rst : in std_logic;
        girdi : in std_logic_vector (n-1 downto 0);
        cikti : out std_logic_vector (5 downto 0)); -- en fazla 32 olabilir
End birleri_say;
```



9.5 For Yapısı

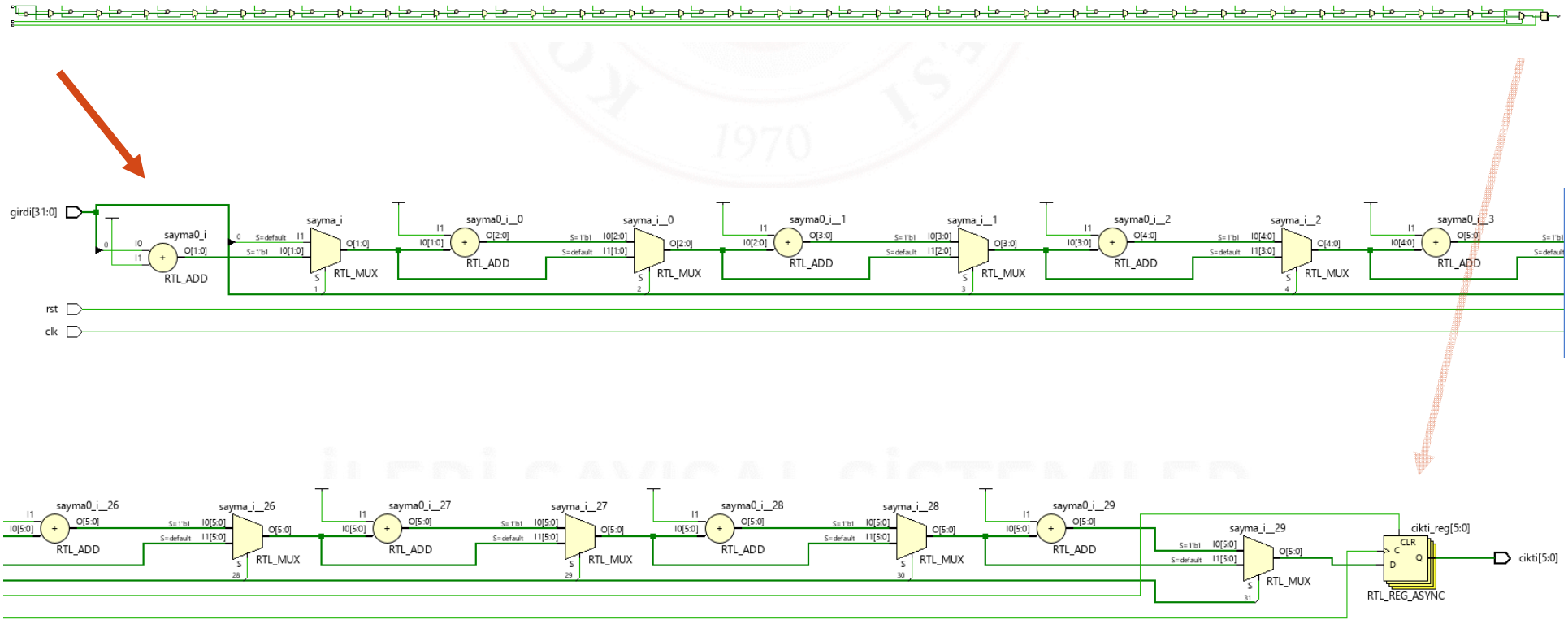
Tasarım-2

```
Architecture davranis of birleri_say is
Begin
   Process (clk, rst)
      variable sayma: integer range 0 to n;
   Begin
      If (rst= '1') then
         sayma := 0;
         cikti <= (others=>'0');
      Elsif rising_edge (clk) then
         sayma := 0;
         For i in 0 to n-1 loop
            If girdi(i) = '1' then
               sayma := sayma + 1;
            End if;
         End loop;
         cikti <= std_logic_vector (to_unsigned(sayma, cikti'length));
      End if;
   End process;
End davranis;
```

9.5 For Yapısı

Tasarım-2

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



9.6 While Yapısı

- Şartlı döngü yapısındadır.

- **While** komutu önünde yazılı durumlar sağlandığı müddetçe döngü devam eder ve blok içi işlemler iteratif sağlanır.

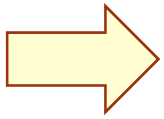
- Şart sağlanmadığı durumda döngü sonlandırılır.

```
While şart loop
   İfade(ler)…
End loop;
```

Örnek 9.5: 32 bitlik bir sayıda kaç adet lojik-1 olduğunu tespit etmek için gerekli VHDL kodunu ***While*** ifadesi üzerinden yazınız.

9.6 While Yapısı

Örnek 9.5: 32 bitlik bir sayıda kaç adet lojik-1 olduğunu tespit etmek için gerekli VHDL kodunu ***While*** ifadesi üzerinden yazınız.



for döngüsü (**Tasarım-2**) örneğinin ***while*** ile gerçeklenmesi

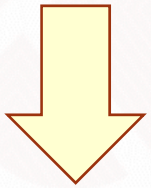
Tasarım-1 ve Tasarım-2'deki aynı RTL Şematik Diyagramı elde edilir.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity birleri_say is
Generic (n: natural := 32);
Port (clk, rst : in std_logic;
        girdi : in std_logic_vector (n-1 downto 0);
        cikti : out std_logic_vector (5 downto 0)); -- en fazla 32 olabilir
End birleri_say;

Architecture davranis of birleri_say is
Begin
  Process (clk, rst)
    variable sayma, i: integer range 0 to n;
  Begin
    If (rst= '1') then
      sayma := 0;
      i := 0;
      cikti <= (others=>'0');
    Elsif rising_edge (clk) then
      sayma := 0;
      i := 0;
      While (i<n) loop
        If girdi(i) = '1' then
          sayma := sayma + 1;
        End if;
        i := i + 1;
      End loop;
      cikti <= std_logic_vector (to_unsigned(sayma, cikti'length));
    End if;
  End process;
End davranis;
```

9.6 While Yapısı

Örnek 9.6: 8 bitlik girdide **eşlik bitini** (**girdideki '1' lerin sayısı çift ise 0, tek ise 1 değerini alan bit**) sağlayan VHDL kodunu **While** ifadesi üzerinden yazınız.

Tasarım-1
(Eşlik Biti)



```
Library ieee;
Use ieee.std_logic_1164.all;

Entity eslik_biti is
Generic (n: natural := 8);
Port (clk, rst : in std_logic;
      girdi : in std_logic_vector (n-1 downto 0);
      es_cikis : out std_logic);
End eslik_biti;
```

9.6 While Yapısı

Tasarım-1
(Eşlik Biti)

```
Architecture davranis of eslik_biti is
Begin
   Process (clk, rst)
      variable deg_tut: std_logic;
      variable i: integer range 0 to n;
   Begin
      If (rst= '1') then
         deg_tut := '0';
         i := 0;
         es_cikis <= '0';
      Elsif rising_edge (clk) then
         deg_tut := '0';
         i := 0;
         While (i < n) loop
            deg_tut := deg_tut xor girdi(i);
            i := i + 1;
         End loop;
         es_cikis <= deg_tut;
      End if;
   End process;
End davranis;
```

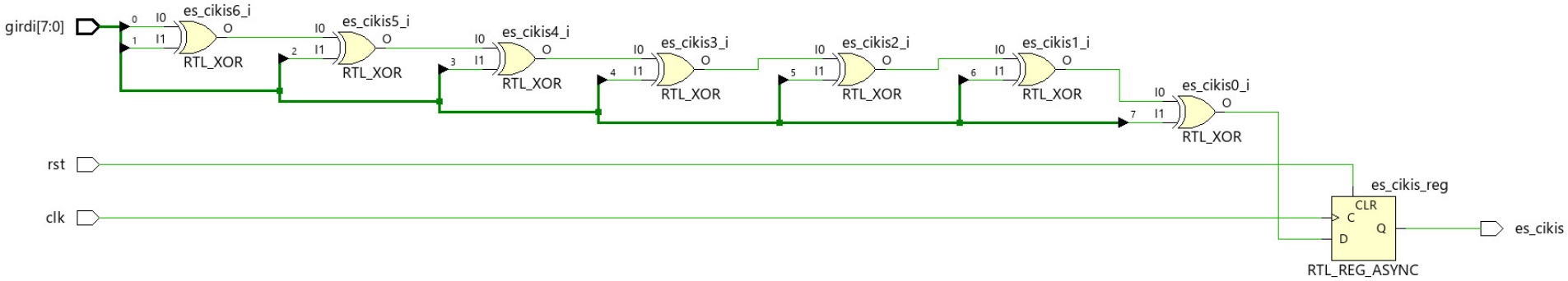
9.6 While Yapısı

Örnek 9.6: 8 bitlik girdide **eşlik bitini** (**girdideki '1' lerin sayısı çift ise 0, tek ise 1 değerini alan bit**) sağlayan VHDL kodunu **While** ifadesi üzerinden yazınız.

Tasarım-1
(Eşlik Biti)

```
Architecture davranis of eslik_biti is
Begin
   Process (clk, rst)
      variable deg_tut: std_logic;
      variable i: integer range 0 to n;
   Begin
      If (rst= '1') then
         deg_tut := '0';
         i := 0;
         es_cikis <= '0';
      Elsif rising_edge (clk) then
         deg_tut := '0';
         i := 0;
         While (i < n) loop
            deg_tut := deg_tut xor girdi(i);
            i := i + 1;
         End loop;
         es_cikis <= deg_tut;
      End if;
   End process;
End davranis;
```

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



9.7 Loop Yapısı

- Diğer döngü yapılarında döngü sonlandırma için kaç kez tekrar edileceği bilgisi sunulur.

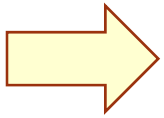
- Ancak **Loop** döngüsünde sonlandırma için yalnızca gerekli koşul (**Exit** ifadesinden sonra) bildirilir.

- **Kontrollü loop döngüsünün tanımı aşağıdaki gibidir:**

```
Etiket: loop
   İfade(ler)…
   Exit Etiket When Kosul;
End loop Etiket;
```

9.7 Loop Yapısı

Örnek 9.5: 32 bitlik bir sayıda kaç adet lojik-1 olduğunu tespit etmek için gerekli VHDL kodunu ***Loop*** ifadesi üzerinden yazınız.



for döngüsü (**Tasarım-2**) örneğinin ***loop*** ile gerçeklenmesi

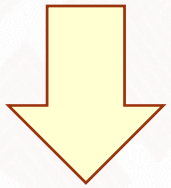
Tasarım-1 ve Tasarım-2'deki aynı RTL Şematik Diyagramı elde edilir.

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity birleri_say is
Generic (n: natural := 32);
Port (clk, rst : in std_logic;
          girdi : in std_logic_vector (n-1 downto 0);
          cikti : out std_logic_vector (5 downto 0)); -- en fazla 32 olabilir
End birleri_say;

Architecture davranis of birleri_say is
Begin
   Process (clk, rst)
      variable sayma, i: integer range 0 to n;
   Begin
      If (rst= '1') then
         sayma := 0;
         i := 0;
         cikti <= (others=>'0');
      Elsif rising_edge (clk) then
         sayma := 0;
         i := 0;
         dongu: loop
            If girdi(i) = '1' then
               sayma := sayma + 1;
            End if;
            i := i + 1;
            Exit dongu When i = n; -- veya Exit dongu When i > n-1;
         End loop dongu;
         cikti <= std_logic_vector (to_unsigned(sayma, cikti'length));
      End if;
   End process;
End davranis;
```

9.7 Loop Yapısı

Örnek 9.7: Örnek 9.6' daki eşlik biti devre tasarımı için gerekli VHDL kodununun **architecture** bölümünü **Loop** ifadesi üzerinden yazınız.

Tasarım-2
(Eşlik Biti)



```
Library ieee;
Use ieee.std_logic_1164.all;

Entity eslik_biti is
Generic (n: natural := 8);
Port (clk, rst : in std_logic;
        girdi : in std_logic_vector (n-1 downto 0);
        es_cikis : out std_logic);
End eslik_biti;
```

9.7 Loop Yapısı

Tasarım-2
(Eşlik Biti)

```
Architecture davranis of eslik_biti is
Begin
   Process (clk, rst)
      variable deg_tut: std_logic;
      variable i: integer range 0 to n;
   Begin
      If (rst='1') then
         deg_tut := '0';
         i := 0;
         es_cikis <= '0';
      Elsif rising_edge (clk) then
         deg_tut := '0';
         i := 0;
         dongu: loop
            deg_tut := deg_tut xor girdi(i);
            i := i + 1;
            Exit dongu When i = n; -- veya Exit dongu When i > n-1;
         End loop dongu;
         es_cikis <= deg_tut;
      End if;
   End process;
End davranis;
```

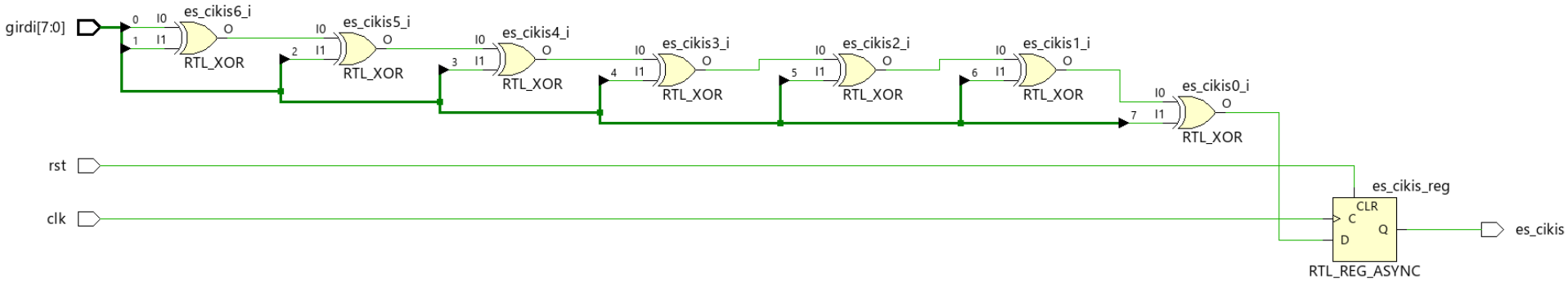
9.7 Loop Yapısı

Örnek 9.6: 8 bitlik girdide **eşlik bitini** (**girdideki '1' lerin sayısı çift ise 0, tek ise 1 değerini alan bit**) sağlayan VHDL kodunu **Loop** ifadesi üzerinden yazınız.

Tasarım-2 (Eşlik Biti)

```
Architecture davranis of eslik_biti is
Begin
   Process (clk, rst)
      variable deg_tut: std_logic;
      variable i: integer range 0 to n;
   Begin
      If (rst='1') then
         deg_tut := '0';
         i := 0;
         es_cikis <= '0';
      Elsif rising_edge (clk) then
         deg_tut := '0';
         i := 0;
         dongu: loop
            deg_tut := deg_tut xor girdi(i);
            i := i + 1;
            Exit dongu When i = n; -- veya Exit dongu When i > n-1;
         End loop dongu;
         es_cikis <= deg_tut;
      End if;
   End process;
End davranis;
```

Vivado / *Elaborated Design Çıktısı* (RTL Şematik Diyagramı)



Tasarım-1 (Eşlik Biti) ve Tasarım-2 (Eşlik Biti) RTL şematik diyagramları aynıdır.

9.8 Alt Programlar

- **Alt programlar**, ana program içerisinde **belirli görevleri yerine getirmek için tasarlanmış algoritmaları içeren** VHDL yapılarıdır.

- Komplike tasarımlarda önemli kolaylık sağlayan iki farklı alt program bulunur: **Fonksiyonlar** ve **Prosedürler**.

- **Fonksiyonlar** çoklu girişe (parametrelere) göre belirli bir tipte **tekil değer döndüren** yapılardır.

- **Prosedürler** ise çoklu girişe göre **çoklu çıkış üretmeyi** sağlarlar, **değer döndürmezler.**

9.8.1 Fonksiyonlar

- Belirli bir tipte **tekil değer döndüren** alt programdır.

- İçinde değişken ve sabit tanımlanabilmesine rağmen, **sinyal tanımlaması gerçekleştirilmez.**

- RAM ve ROM gibi birimlerde, benzer işlemlerin defalarca yapılmasında işlevsellik sağlar (adres çözümleme, veri dönüştürme, vb gibi).

- Fonksiyonlar bir değer döndürdüğünden, prosedürlerden farklı olarak **return** komutu içerir.

```
Function fonksiyon_ismi (giris_1, giris_2, …, giris_n : tür) return tür is
  Sabit_veya_degisken_tanımlamalari
Begin
  İfade(ler)…
End fonksiyon_ismi;
```

9.8.1 Fonksiyonlar

- **Fonksiyon** tanımlaması, **architecture** içinde kullanılması halinde **begin** kelimesinden önce yapılmalıdır.

- **Girişindeki parametreler sinyal olabilmektedir**, ancak **işlem yapılacağı zaman önceden de belirtildiği gibi değişken veya sabitler üzerinden görevler sağlanmalıdır.**

- Fonksiyon isimlendirmesi yapılırken genelde "f" harfi sonrası farklı ifadelerin yazılması tercih edilir.

Örnek 9.8: Komplike bir tasarım içinde $(\bar{A} + B)C$ lojik ifadesinin sürekli işletilmesi gerekiyor. Buna göre **A**,**B** ve **C**'ye karşılık **giris1**, **giris2** ve **giris3** sinyalleri sistem girişi olacağı durumda gerekli VHDL kodunu yazınız.

9.8.1 Fonksiyonlar

Örnek 9.8: Komplike bir tasarım içinde $(\bar{A} + B)C$ lojik ifadesinin sürekli işletilmesi gerekiyor. Buna göre **A**,**B** ve **C**'ye karşılık **giris1**, **giris2** ve **giris3** sinyalleri sistem girişi olacağı durumda gerekli VHDL kodunu yazınız.

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity lojik_islem is
Port (giris1, giris2, giris3 : in std_logic;
      cikti : out std_logic);
End lojik_islem;

Architecture davranis of lojik_islem is
   Function f3islem (A, B, C : std_logic)
      return std_logic is
   variable ara_deg: std_logic;
   Begin
      ara_deg := ((not A) or B) and C;
      return ara_deg;
   End f3islem;
Begin
   cikti <= f3islem(giris1, giris2, giris3);
End davranis;
```

9.8.2 Prosedürler

- Herhangi bir değer döndürmeyen, **yapısında tanımlı girişlere göre devreyi işleterek sonucu yine yapısında tanımlı çıkış değer(ler)ine aktaran** alt programdır.

- **Prosedürün tanımlama yeri**, fonksiyon tanımlamalarında da olduğu gibi **architecture ifadesinden sonra begin komutundan önce** olmalıdır.

- Prosedür isimlendirmesi yapılırken genelde "p" harfi sonrası farklı ifadelerin yazılması tercih edilir.

- Fonksiyondan farklı olarak **girdi sinyalleri üzerinde direk işlem yapabilir**, ama **içinde sinyal tanımlaması yapılamaz** (prosedür ve process'lerde de aynı kural geçerlidir).

9.8.2 Prosedürler

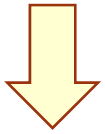
- Procedure ifadesi sonrası, parantez içerisinde girdiler ve gerekli çıktı(lar) belirtilir.

- Burada fonksiyonlarda olduğu gibi **sıralamanın dikkatli şekilde yapılması gerekir.**

```
Procedure prosedur_ismi (nesne_tipi girişler: in tür;
                         nesne_tipi cikislar: out tür) is
  Sabit_veya_degisken_tanımlamalari
Begin
  İfade(ler)…
End prosedur_ismi;
```

9.8.2 Prosedürler

Örnek 9.9: Örnek 9.8' de lojik ifadeyi **prosedür** içinde tanımlayarak gerekli VHDL kodunu yazınız.



Komplike bir tasarım içinde $(\bar{A} + B)C$ lojik ifadesinin sürekli işletilmesi gerekiyor. Buna göre **A**,**B** ve **C**' ye karşılık **giris1**, **giris2** ve **giris3** sinyalleri sistem girişi olacağı durumda gerekli VHDL kodunu yazınız.

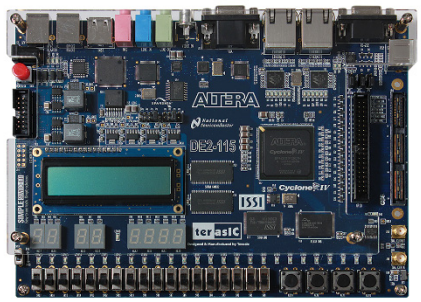
```
Library ieee;
Use ieee.std_logic_1164.all;

Entity lojik_islem is
Port (giris1, giris2, giris3 : in std_logic;
      cikti : out std_logic);
End lojik_islem;

Architecture davranis of lojik_islem is
   Procedure p3islem (signal A, B, C : in std_logic;
                      signal elde : out std_logic) is
   Begin
      elde <= ((not A) or B) and C;
   End p3islem;
Begin
   p3islem(giris1, giris2, giris3, cikti);
End davranis;
```

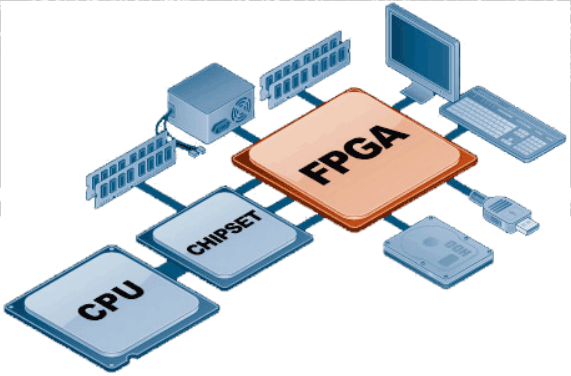
SONRAKİ DERS KONUSU

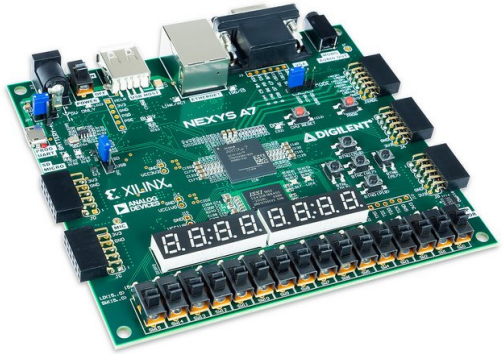
10- ÖRNEK UYGULAMALAR

İLERİ SAYISAL SİSTEMLER

10- ÖRNEK UYGULAMALAR







Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

10. ÖRNEK UYGULAMALAR

Örnek 10.1: 8'li LED devresinde sol ve sağ butonlarının durumuna göre sağa ve sola kaydırma işlemi sağlayan VHDL kodunu yazınız. (**Not:** En sağdaki led ile başlangıç yapalım)

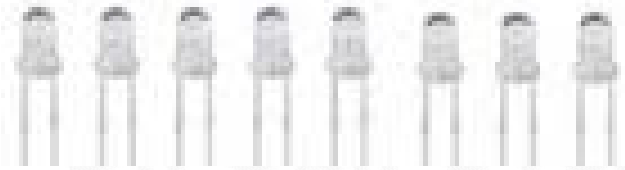
```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity led_uyg is
Port (clk, rst, sol, sag : in std_logic;
      cikti : out std_logic_vector (7 downto 0));
End led_uyg;
```

10. ÖRNEK UYGULAMALAR

-- Aralık belirtmek, derleyicinin integer işlemler için doğrudan 32 bitlik işlem yapmasını (gereksiz devre yoğunluğunu) engeller.

Örnek 10.1: 8'li LED devresinde sol ve sağ butonlarının durumuna göre sağa ve sola kaydırma işlemi sağlayan VHDL kodunu yazınız.

(**Not:** En sağdaki led ile başlangıç yapalım)

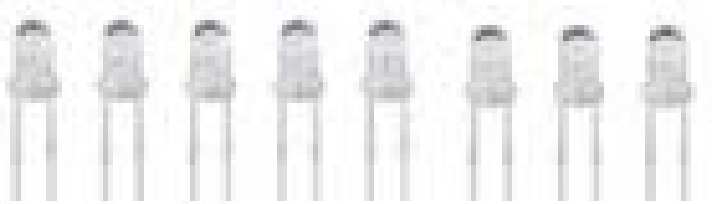


```
Architecture davranis of led_uyg is
   signal dizi : std_logic_vector (7 downto 0) := x"01";
   signal sayma : integer range 0 to 7 := 0;
Begin
   Process (clk, rst)
   Begin
      If rst = '1' then
         sayma <= 0;
         dizi <= (others => '0');
         dizi(0) <= '1';
         cikti <= dizi;
      Elsif rising_edge(clk) then
         If sol = '1' and sag='0' then
            If sayma = 7 then
               sayma <= 0;
            Else
               sayma <= sayma + 1;
            End if;
            dizi <= (others => '0');
            dizi(sayma) <= '1';
            cikti <= dizi;
         Elsif sol='0' and sag = '1' then
            If sayma = 0 then
               sayma <= 7;
            Else
               sayma <= sayma - 1;
            End if;
            dizi <= (others => '0');
            dizi(sayma) <= '1';
            cikti <= dizi;
         End if;
      End if;
   End process;
End davranis;
```

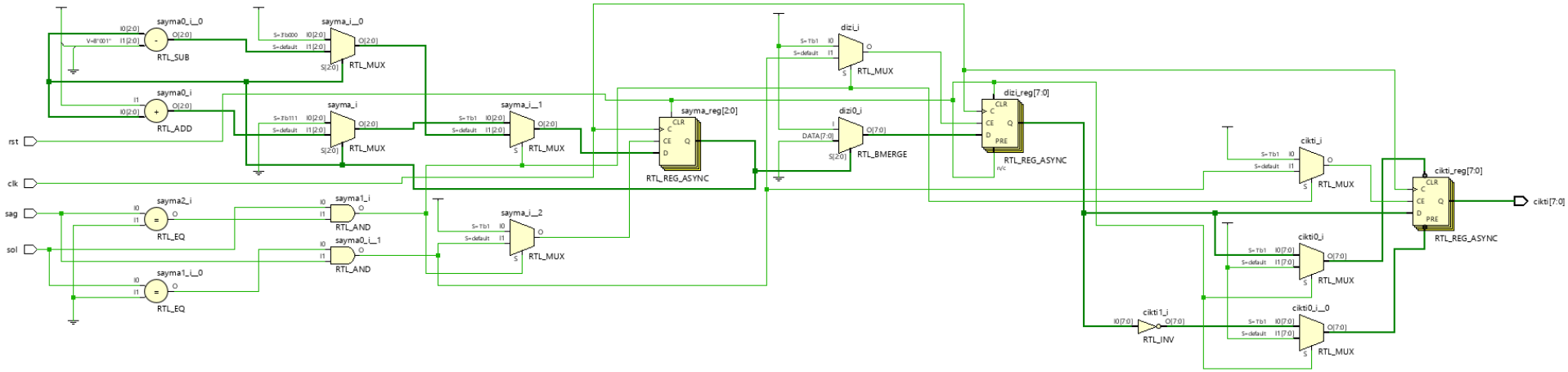
10. ÖRNEK UYGULAMALAR

Örnek 10.1: 8'li LED devresinde sol ve sağ butonlarının durumuna göre sağa ve sola kaydırma işlemi sağlayan VHDL kodunu yazınız.

(**Not:** En sağdaki led ile başlangıç yapalım)



Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



10. ÖRNEK UYGULAMALAR

-- Aralık belirtmek, derleyicinin integer işlemler için doğrudan 32 bitlik işlem yapmasını (gereksiz devre yoğunluğunu) engeller.

Ek Önemli Bilgi:

Gri arkaplanlı bölüm yalnızca belirtilen yere yazılırsa,

- sol='0' **and** sag='0' ile
- sol='1' **and** sag='1'

durumlarında da çalışmaya devam eder ve gereksiz güç kaybına (gereksiz yüklemeye) sebebiyet verebilir.

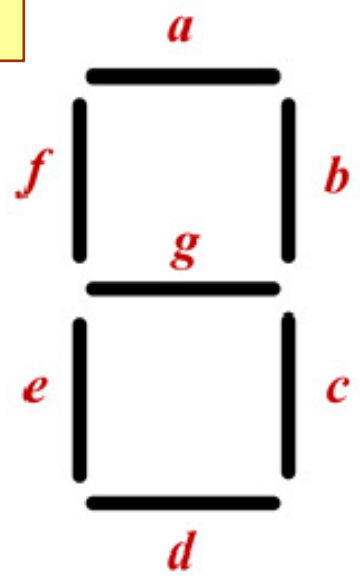
```
Architecture davranis of led_uyg is
   signal dizi : std_logic_vector (7 downto 0) := x"01";
   signal sayma : integer range 0 to 7 := 0;
Begin
   Process (clk, rst)
   Begin
      If rst = '1' then
         sayma <= 0;
         dizi <= (others => '0');
         dizi(0) <= '1';
         cikti <= dizi;
      Elsif rising_edge(clk) then
         If sol = '1' and sag='0' then
            If sayma = 7 then
               sayma <= 0;
            Else
               sayma <= sayma + 1;
            End if;
            dizi <= (others => '0');
            dizi(sayma) <= '1';
            cikti <= dizi;
         Elsif sol='0' and sag = '1' then
            If sayma = 0 then
               sayma <= 7;
            Else
               sayma <= sayma - 1;
            End if;
            dizi <= (others => '0');
            dizi(sayma) <= '1';
            cikti <= dizi;
         End if;
      End if;
   End process;
End davranis;
```

10. ÖRNEK UYGULAMALAR

Örnek 10.2: 4 bitlik giriş portu ile 0-F arası sayıları aygıt üzerindeki ilk 7-segmentte görüntüleyen kodlamayı yapınız.

Tasarım-1

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity seven_seg is
Port (clk, rst : in std_logic;
        girdi : in std_logic_vector (3 downto 0);
        secim : out std_logic_vector (7 downto 0);
        cikti : out std_logic_vector (7 downto 0));
End seven_seg;
```

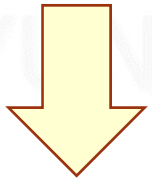


Bilgi aktarımı:
pabcdefg
(p→aktif etmek için lojik-1 olmalı)

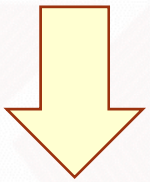
10. ÖRNEK UYGULAMALAR

Tasarım-1

```
Architecture davranis of seven_seg is
   signal lcd_bcd : std_logic_vector (3 downto 0) := (others => '0');
   signal lcd_out : std_logic_vector (7 downto 0) := "10000001";
Begin
   Process (lcd_bcd)
   Begin
      Case lcd_bcd is -- sayı pabcdefg şeklinde yazılır
         when "0000" => lcd_out <= "10000001"; -- “0”
         when "0001" => lcd_out <= "11001111"; -- “1”
         when "0010" => lcd_out <= "10010010"; -- “2”
         when "0011" => lcd_out <= "10000110"; -- “3”
         when "0100" => lcd_out <= "11001100"; -- “4”
         when "0101" => lcd_out <= "10100100"; -- “5”
         when "0110" => lcd_out <= "10100000"; -- “6”
         when "0111" => lcd_out <= "10001111"; -- “7”
         when "1000" => lcd_out <= "10000000"; -- “8”
         when "1001" => lcd_out <= "10000100"; -- “9”
         when "1010" => lcd_out <= "10000010"; -- “a”
         when "1011" => lcd_out <= "11100000"; -- “b”
         when "1100" => lcd_out <= "10110001"; -- “C”
         when "1101" => lcd_out <= "11000010"; -- “d”
         when "1110" => lcd_out <= "10110000"; -- “E”
         when "1111" => lcd_out <= "10111000"; -- “F”
         when others => lcd_out <= "10000001"; -- “0”
      End case;
   End process;
```



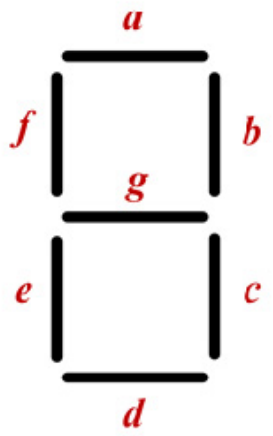
10. ÖRNEK UYGULAMALAR



Tasarım-1

```
Process (clk, rst)
Begin
    If rst = '1' then
        secim <= "11111110"; -- 7-segment display'lerden en sağdaki aktif
        lcd_bcd <= (others=>'0');
        cikti <= x"81";
    Elsif rising_edge(clk) then
        secim <= "11111110"; -- 7-segment display'lerden en sağdaki aktif
        lcd_bcd <= girdi;
        cikti <= lcd_out;
    End if;
  End process;
End davranis;
```

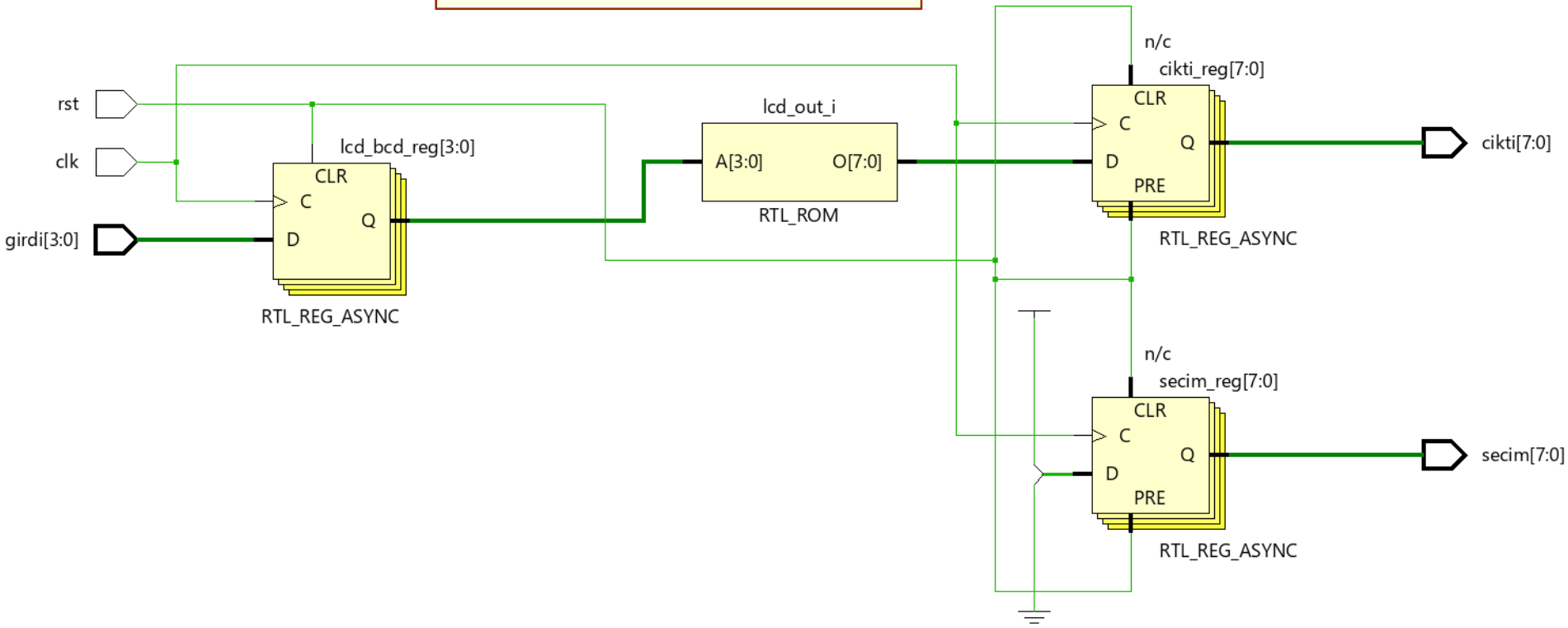
10. ÖRNEK UYGULAMALAR



Örnek 10.2: 4 bitlik giriş portu ile 0-F arası sayıları aygıt üzerindeki ilk 7-segmentte görüntüleyen kodlamayı yapınız.

Tasarım-1

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

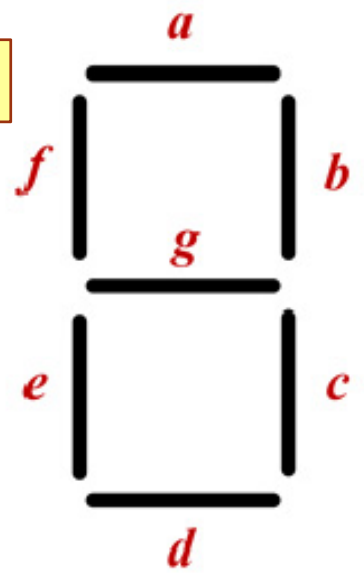


10. ÖRNEK UYGULAMALAR

Örnek 10.2: 4 bitlik giriş portu ile 0-F arası sayıları aygıt üzerindeki ilk 7-segmentte görüntüleyen kodlamayı yapınız.

Tasarım-2

```
Library ieee;
Use ieee.std_logic_1164.all;

Entity seven_seg is
Port (clk, rst : in std_logic;
        girdi : in std_logic_vector (3 downto 0);
        secim : out std_logic_vector (7 downto 0);
        cikti : out std_logic_vector (7 downto 0));
End seven_seg;
```

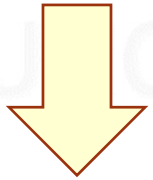


Bilgi aktarımı:
pabcdefg
(p→aktif etmek için lojik-1 olmalı)

10. ÖRNEK UYGULAMALAR

Tasarım-2

```
Architecture davranis of seven_seg is
   signal lcd_out : std_logic_vector (7 downto 0) := "10000001";
Begin
   secim <= "11111110";
   With girdi select
   lcd_out <= "10000001" when "0000", -- “0”
              "11001111" when "0001", -- “1”
              "10010010" when "0010", -- “2”
              "10000110" when "0011", -- “3”
              "11001100" when "0100", -- “4”
              "10100100" when "0101", -- “5”
              "10100000" when "0110", -- “6”
              "10001111" when "0111", -- “7”
              "10000000" when "1000", -- “8”
              "10000100" when "1001", -- “9”
              "10000010" when "1010", -- “a”
              "11100000" when "1011", -- “b”
              "10110001" when "1100", -- “C”
              "11000010" when "1101", -- “d”
              "10110000" when "1110", -- “E”
              "10111000" when "1111", -- “F”
              "10000001" when Others; -- “0”
```

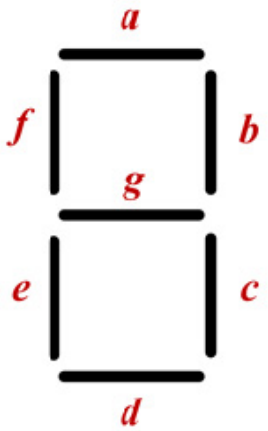


10. ÖRNEK UYGULAMALAR

Tasarım-2

```
Process (clk, rst)
Begin
    If rst = '1' then
        cikti <= x"81";
    Elsif rising_edge(clk) then
        cikti <= lcd_out;
    End if;
End process;
End davranis;
```

10. ÖRNEK UYGULAMALAR

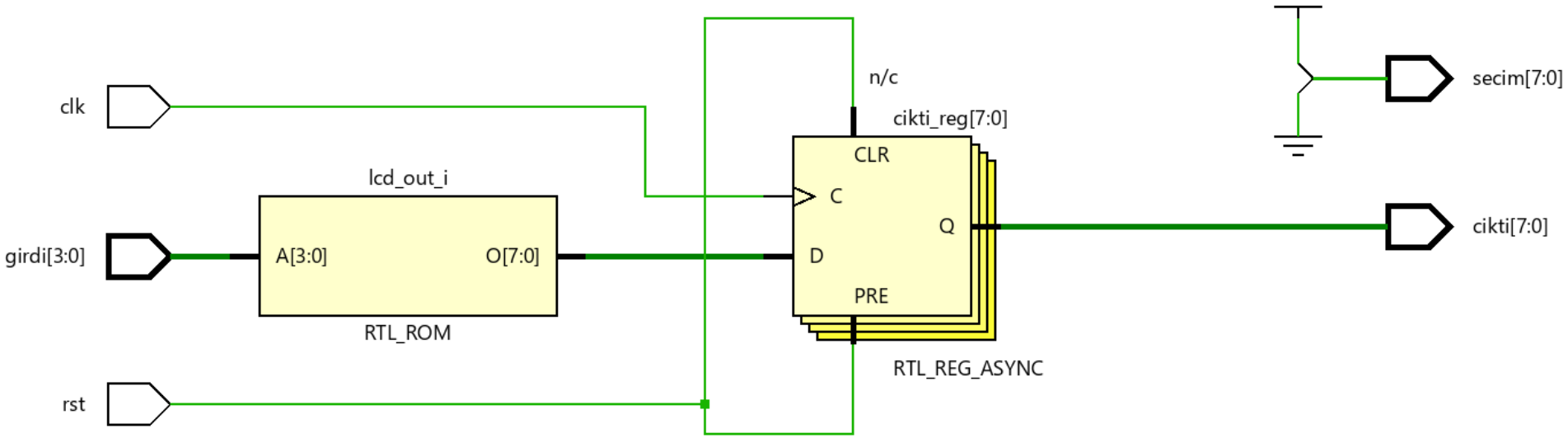


Bilgi aktarımı:
pabcdefg
(p→aktif etmek için lojik-1 olmalı)

Örnek 10.2: 4 bitlik giriş portu ile 0-F arası sayıları aygıt üzerindeki ilk 7-segmentte görüntüleyen kodlamayı yapınız.

Tasarım-2

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



10. ÖRNEK UYGULAMALAR

Örnek 10.3: 4 bitlik bilgiyi şifreyelerek karşıya ileten bir kodlama devresi tasarlanacaktır. Tasarımda;

1) 4 bitlik temel bilgi alınarak 1' e tümlenecek ve elde edilen bilgi temel bilgi soluna eklenecek,

2) 8 bitlik veri sağa 1 bit döndürülecek (**ror** komutu olmadan),

3) Adım-2' de elde edilen bilgi 11000011 anahtarı ile XOR işlemine tabi tutulacak.

Gerekli VHDL kodunu yazınız.

Ek Bilgi:

- Hem şifrelemede hem de çözümlemede **aynı anahtar** varsa **simetrik şifreleme**,
- Her iki durum için **farklı anahtar** kullanılacaksa **asimetrik şifreleme** söz konusudur.

Geleneksel Diğer Şifreleme Türleri (bkz.):

- **Blok Şifreleme**
- **Seri Şifreleme**
- **RC5 Şifreleme vb gibi.**

10. ÖRNEK UYGULAMALAR

Tasarım-1

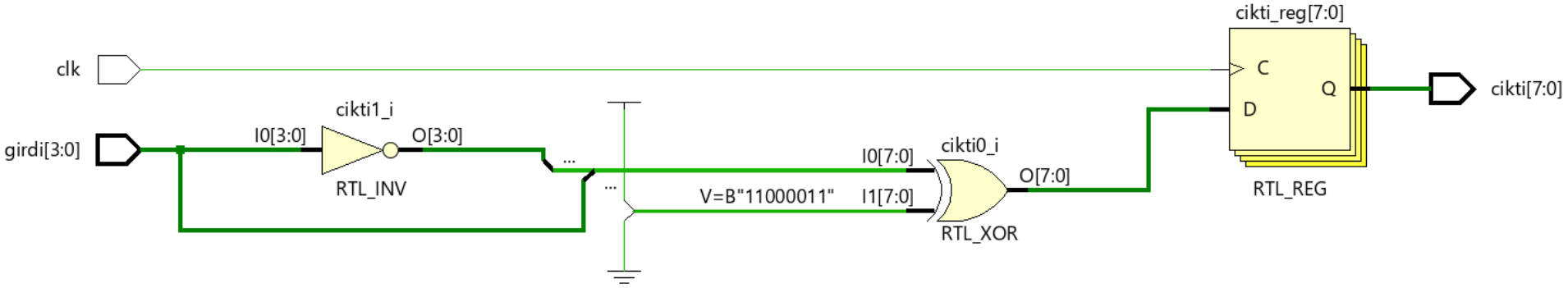
```
Library ieee;
Use ieee.std_logic_1164.all;

Entity sifreleme is
Port (clk : in std_logic;
        girdi : in std_logic_vector (3 downto 0);
        cikti : out std_logic_vector (7 downto 0));
End sifreleme;

Architecture davranis of sifreleme is
Begin
   Process (clk)
      variable deger: std_logic_vector (7 downto 0);
      constant anahtar: std_logic_vector (7 downto 0) := "11000011";
   Begin
      If rising_edge (clk) then
         deger := not girdi & girdi;
         deger := deger(0) & deger(7 downto 1);
         cikti <= deger xor anahtar;
      End if;
   End process;
End davranis;
```

10. ÖRNEK UYGULAMALAR

Tasarım-1

```
Architecture davranis of sifreleme is
Begin
   Process (clk)
      variable deger: std_logic_vector (7 downto 0);
      constant anahtar: std_logic_vector (7 downto 0) := "11000011";
   Begin
      If rising_edge (clk) then
         deger := not girdi & girdi;
         deger := deger(0) & deger(7 downto 1);
         cikti <= deger xor anahtar;
      End if;
   End process;
End davranis;
```

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



10. ÖRNEK UYGULAMALAR

ror komutu kullansaydık nasıl yazardık?

Tasarım-1'deki aynı RTL Şematik Diyagramı elde edilir.

Tasarım-2

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity sifreleme is
Port (clk : in std_logic;
        girdi : in std_logic_vector (3 downto 0);
        cikti : out std_logic_vector (7 downto 0));
End sifreleme;

Architecture davranis of sifreleme is
Begin
   Process (clk)
      variable deger: unsigned (7 downto 0);
      constant anahtar: unsigned (7 downto 0) := "11000011";
   Begin
      If rising_edge (clk) then
         deger := unsigned(not girdi & girdi);
         deger := deger ror 1;
         cikti <= std_logic_vector(deger xor anahtar);
      End if;
   End process;
End davranis;
```

10. ÖRNEK UYGULAMALAR

Örnek 10.4: 4 bitlik bilgi üzerinden saat (**clk**) sinyali temelli ***frekans bölücü devre*** tasarımı gerçekleştirilecektir. Buna göre;

- **rst** giriş portu aktif olduğunda frekans bölücü devre çıkışlarına lojik-0 bilgisi aktarılacaktır. (**4 bitlik** olduğu için çıkışlar **/2, /4, /8, /16** işlemlerini yapar)

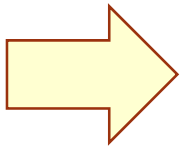
- **clk** giriş portunun ***yükselen kenarında*** sayma değeri bir artırılarak, **4 bitlik** sayıcı değişimi gerçekleştirilecektir. Bu dört bitlik bilginin herbir hanesi **/2, /4, /8, /16** işlemlerine karşılık ilgili çıkış portlarına aktarılacaktır.

- Sayma işlemine bağımlı analiz ve devre tasarımı yapıldığından, sayıcı 4 bitlik maksimum durum sonrası ***sıfırlanmalıdır***.

Gerekli VHDL kodunu yazınız.

10. ÖRNEK UYGULAMALAR

Tasarım-1



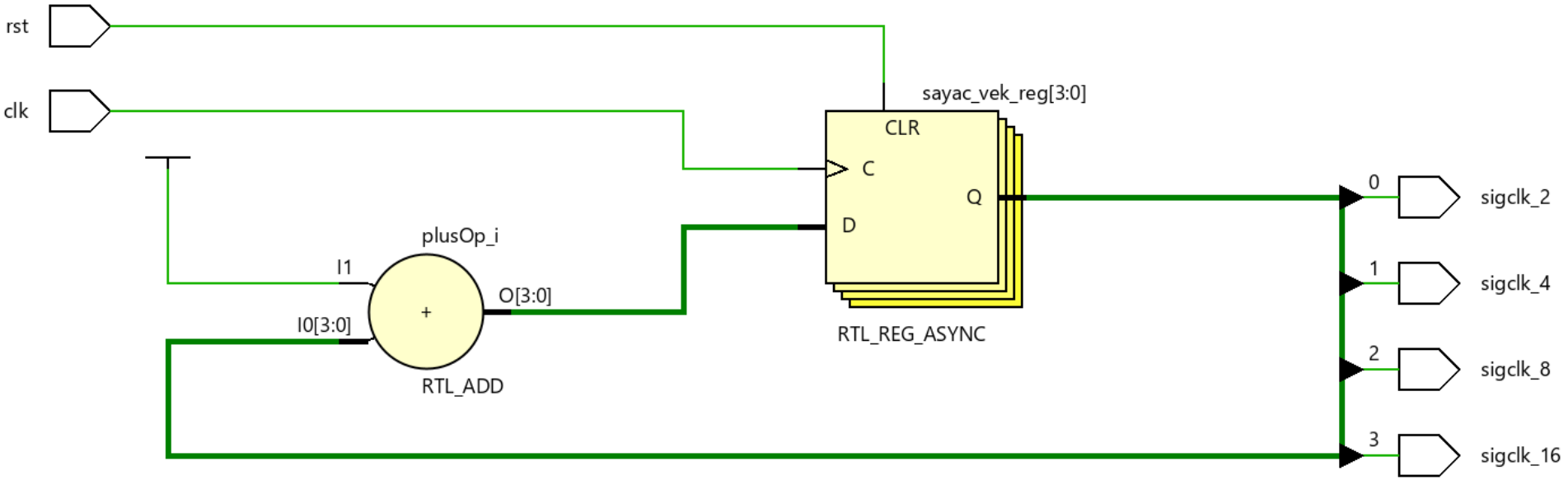
```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity freq_div is
Port (clk, rst : in std_logic;
        sigclk_2, sigclk_4, sigclk_8, sigclk_16 : out std_logic);
End freq_div;

Architecture davranis of freq_div is
Signal sayac_vek : std_logic_vector(3 downto 0) := (others => '0');
Begin

sigclk_2 <= sayac_vek (0); -- f/2
sigclk_4 <= sayac_vek (1); -- f/4
sigclk_8 <= sayac_vek (2); -- f/8
sigclk_16 <= sayac_vek (3); -- f/16

  Process(clk, rst)
  Begin
      If rst='1' then
          sayac_vek <= (others => '0');
      Elsif rising_edge(clk) then
          sayac_vek <= std_logic_vector(unsigned(sayac_vek) + 1);
      End if;
  End process;
End davranis;
```

10. ÖRNEK UYGULAMALAR

Tasarım-1

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

```
Architecture davranis of freq_div is
Signal sayac_vek : std_logic_vector(3 downto 0) := (others => '0');
Begin

sigclk_2 <= sayac_vek (0); -- f/2
sigclk_4 <= sayac_vek (1); -- f/4
sigclk_8 <= sayac_vek (2); -- f/8
sigclk_16 <= sayac_vek (3); -- f/16

 Process(clk, rst)
 Begin
     If rst='1' then
          sayac_vek <= (others => '0');
     Elsif rising_edge(clk) then
          sayac_vek <= std_logic_vector(unsigned(sayac_vek) + 1);
     End if;
 End process;
End davranis;
```

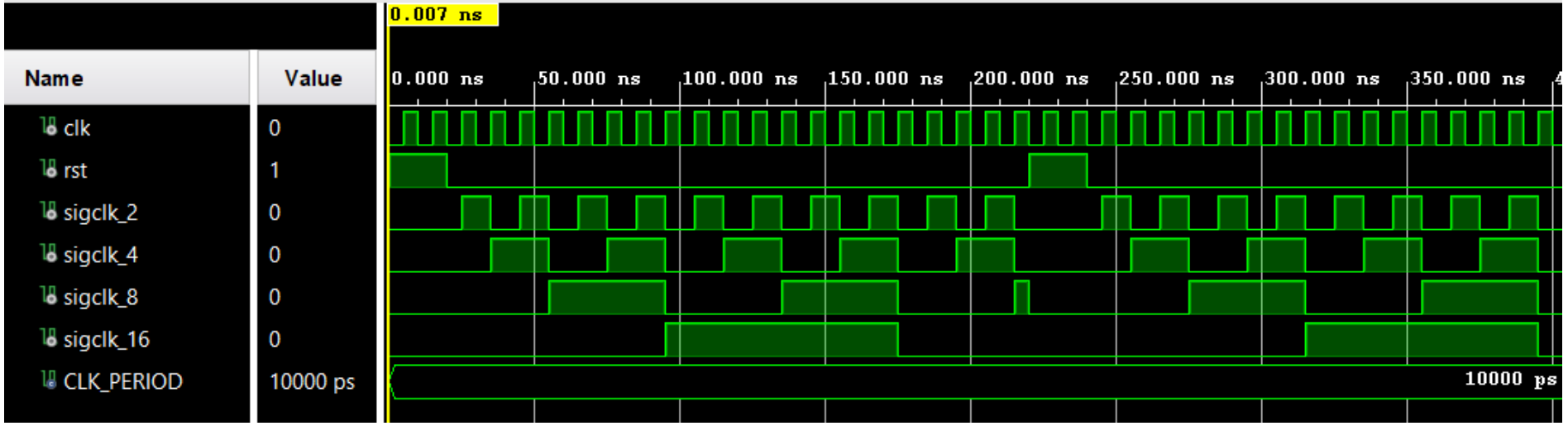


10. ÖRNEK UYGULAMALAR

Tasarım-1

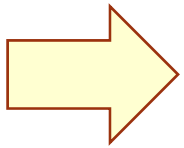
```
Architecture davranis of freq_div is
Signal sayac_vek : std_logic_vector(3 downto 0) := (others => '0');
Begin

sigclk_2 <= sayac_vek (0); -- f/2
sigclk_4 <= sayac_vek (1); -- f/4
sigclk_8 <= sayac_vek (2); -- f/8
sigclk_16 <= sayac_vek (3); -- f/16

  Process(clk, rst)
  Begin
      If rst='1' then
          sayac_vek <= (others => '0');
      Elsif rising_edge(clk) then
          sayac_vek <= std_logic_vector(unsigned(sayac_vek) + 1);
      End if;
  End process;
End davranis;
```

Vivado /
Simulation Çıktısı



10. ÖRNEK UYGULAMALAR

Tasarım-2



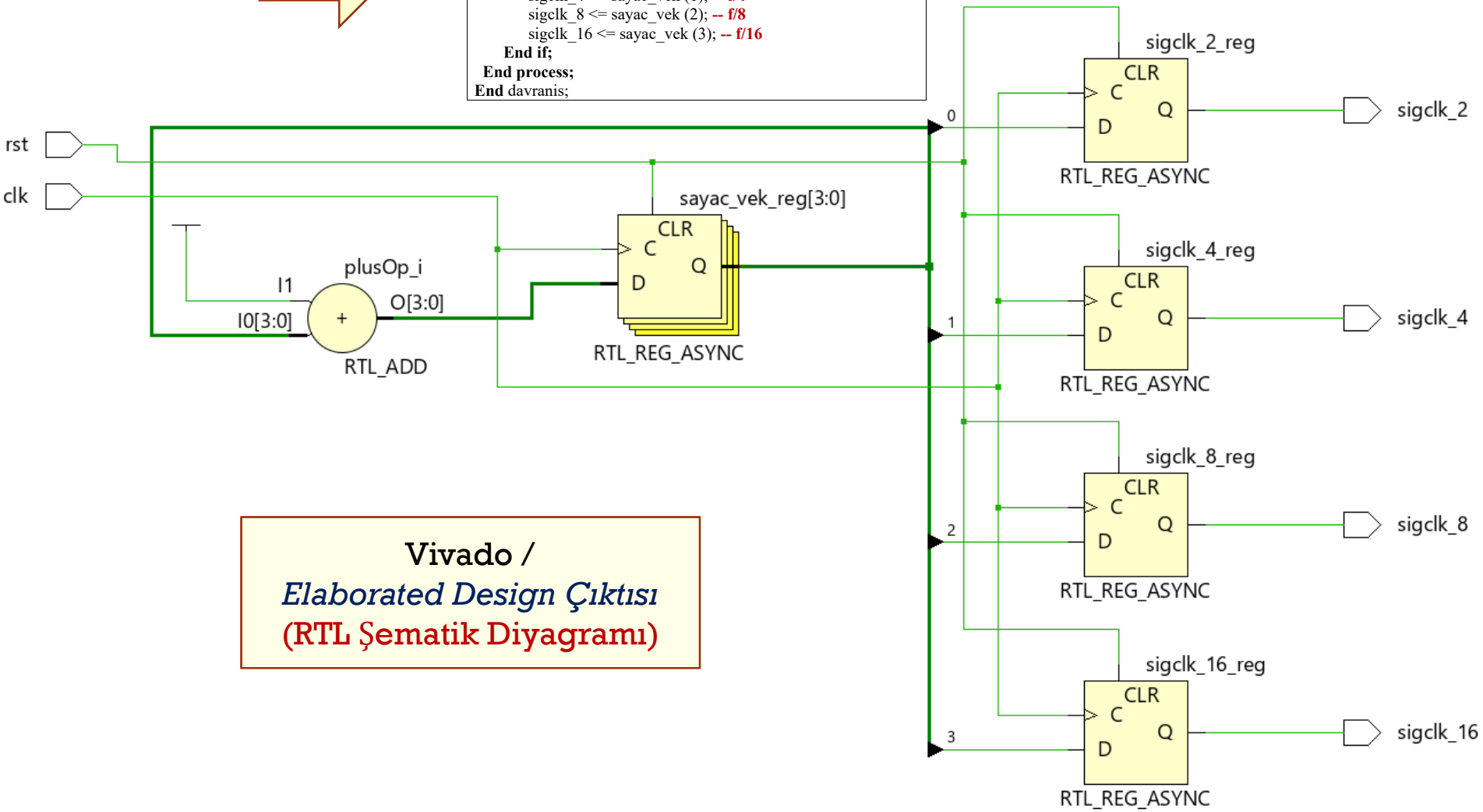
Glitch (Atlama) durumunu önlemeye yönelik tasarım

```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity freq_div is
Port (clk, rst : in std_logic;
      sigclk_2, sigclk_4, sigclk_8, sigclk_16 : out std_logic);
End freq_div;

Architecture davranis of freq_div is
Signal sayac_vek : std_logic_vector(3 downto 0) := (others => '0');
Begin
 Process(clk, rst)
 Begin
    If rst='1' then
        sayac_vek <= (others => '0');
        sigclk_2 <= '0';
        sigclk_4 <= '0';
        sigclk_8 <= '0';
        sigclk_16 <= '0';
    Elsif rising_edge(clk) then
        sayac_vek <= std_logic_vector(unsigned(sayac_vek) + 1);
        sigclk_2 <= sayac_vek (0); -- f/2
        sigclk_4 <= sayac_vek (1); -- f/4
        sigclk_8 <= sayac_vek (2); -- f/8
        sigclk_16 <= sayac_vek (3); -- f/16
    End if;
 End process;
End davranis;
```

10. ÖRNEK UYGULAMALAR

Tasarım-2

```
Architecture davranis of freq_div is
Signal sayac_vek : std_logic_vector(3 downto 0) := (others => '0');
Begin
  Process(clk, rst)
  Begin
     If rst='1' then
          sayac_vek <= (others => '0');
          sigclk_2 <= '0';
          sigclk_4 <= '0';
          sigclk_8 <= '0';
          sigclk_16 <= '0';
     Elsif rising_edge(clk) then
          sayac_vek <= std_logic_vector(unsigned(sayac_vek) + 1);
          sigclk_2 <= sayac_vek (0); -- f/2
          sigclk_4 <= sayac_vek (1); -- f/4
          sigclk_8 <= sayac_vek (2); -- f/8
          sigclk_16 <= sayac_vek (3); -- f/16
     End if;
  End process;
End davranis;
```



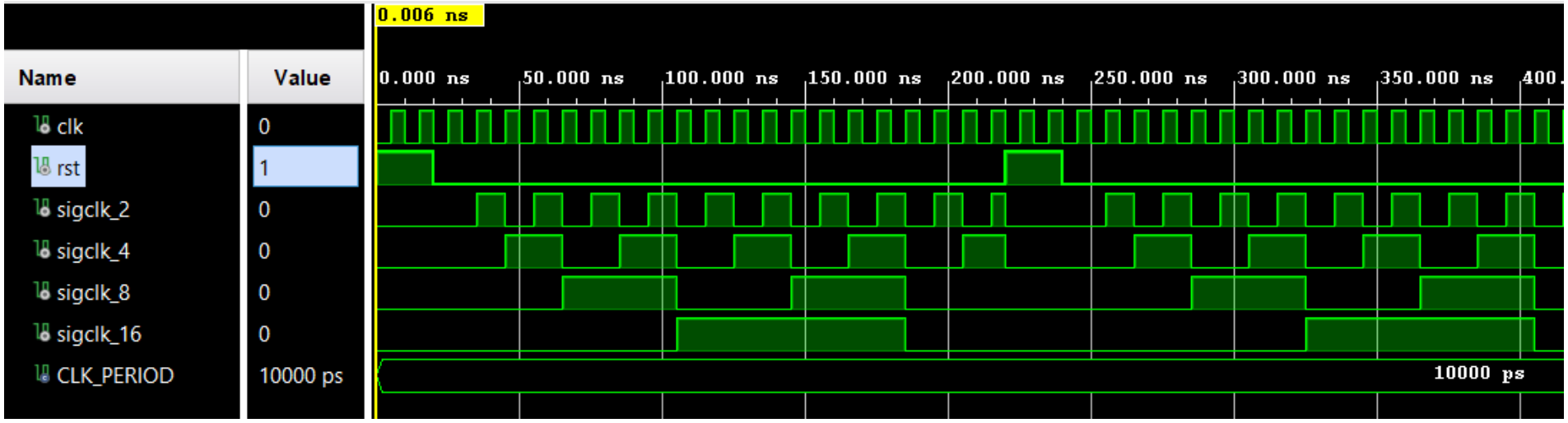
Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

10. ÖRNEK UYGULAMALAR

Tasarım-2

```
Architecture davranis of freq_div is
Signal sayac_vek : std_logic_vector(3 downto 0) := (others => '0');
Begin
  Process(clk, rst)
  Begin
     If rst='1' then
          sayac_vek <= (others => '0');
          sigclk_2 <= '0';
          sigclk_4 <= '0';
          sigclk_8 <= '0';
          sigclk_16 <= '0';
     Elsif rising_edge(clk) then
          sayac_vek <= std_logic_vector(unsigned(sayac_vek) + 1);
          sigclk_2 <= sayac_vek (0); -- f/2
          sigclk_4 <= sayac_vek (1); -- f/4
          sigclk_8 <= sayac_vek (2); -- f/8
          sigclk_16 <= sayac_vek (3); -- f/16
     End if;
  End process;
End davranis;
```

Vivado /
Simulation Çıktısı

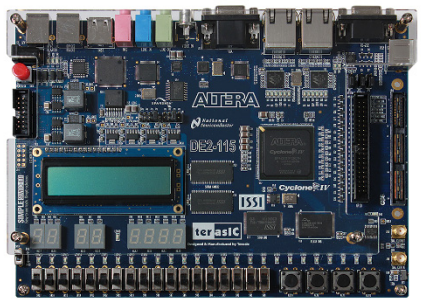


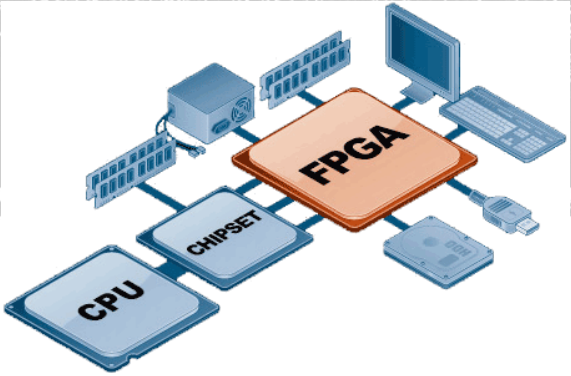
SONRAKİ DERS KONUSU

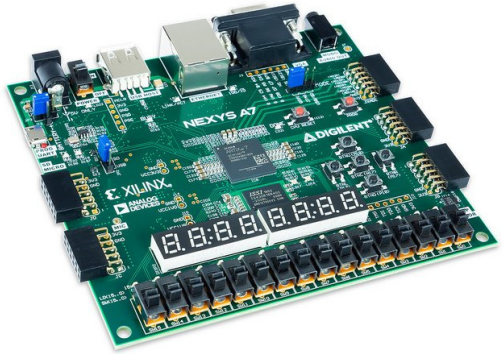
11- HAFIZA İŞLEMLERİ

İLERİ SAYISAL SİSTEMLER

11- HAFIZA İŞLEMLERİ







Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

11. HAFIZA İŞLEMLERİ

- Hafıza birimleri; **gelen veriyi tutmak**, yine bu **veriyi işletmek (başka birimlere aktarmak)** amacıyla sayısal tasarımlarda sıklıkla kullanılan yapılardır.

- Bu noktada birçok sayısal sistem, **veri tutma ve işleme opsiyonları** düşünülerek tasarlanır.

- Hafıza işlemlerinde 1-bitlik veri en yalın halde **FF'ler** üzerinde tutulmaktadır.

- Hafıza birimleri ise FF birimleri üzerinden **blok yapılar oluşturularak** elde edilir.

- ROM, PROM, EPROM, RAM vb. hafıza ünitelerinin en temel ortak paydası **adres yolu** ve **veri yolu** kavramlarıdır.

11. HAFIZA İŞLEMLERİ

- Adres yolu bit sayısı / Adres bit sayısı **(n)**, **kaç farklı verinin (2^n adet) tutulabileceğini** gösterir.

- Veri yolu bit sayısı / Veri bit sayısı **(m)** ise **kaç bitlik bilgiler üzerinde işlem yapılacağını** nitelemektedir.

	Adres	Veri
0	0000	10010110
1	0001	11111111
2	0010	00001111
.	.	.
.	.	.
.	.	.
15	1111	00001000

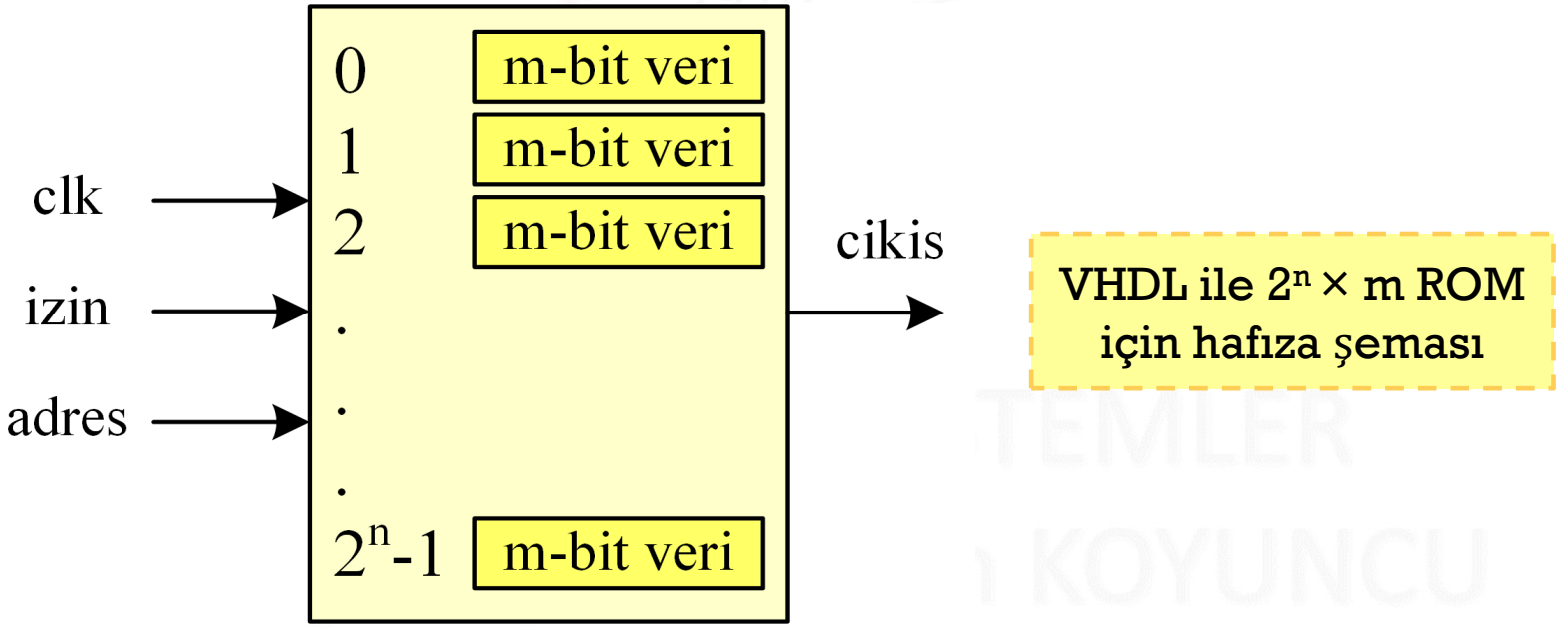
2^nxm ROM/RAM
2^4x8 ROM/RAM
16x8 ROM/RAM
n=adres yolu (bit sayısı)
m=veri uzunluğu (bit sayısı)
Adres<3:0> Veri<7:0>

Bir hafıza birimindeki adres yolu ve veri yolu (uzunluğu) temel gösterimi

11.1 ROM (Okunabilir) Hafıza

- ROM birimi yalnızca okuma yapılabilen hafıza yapısıdır.

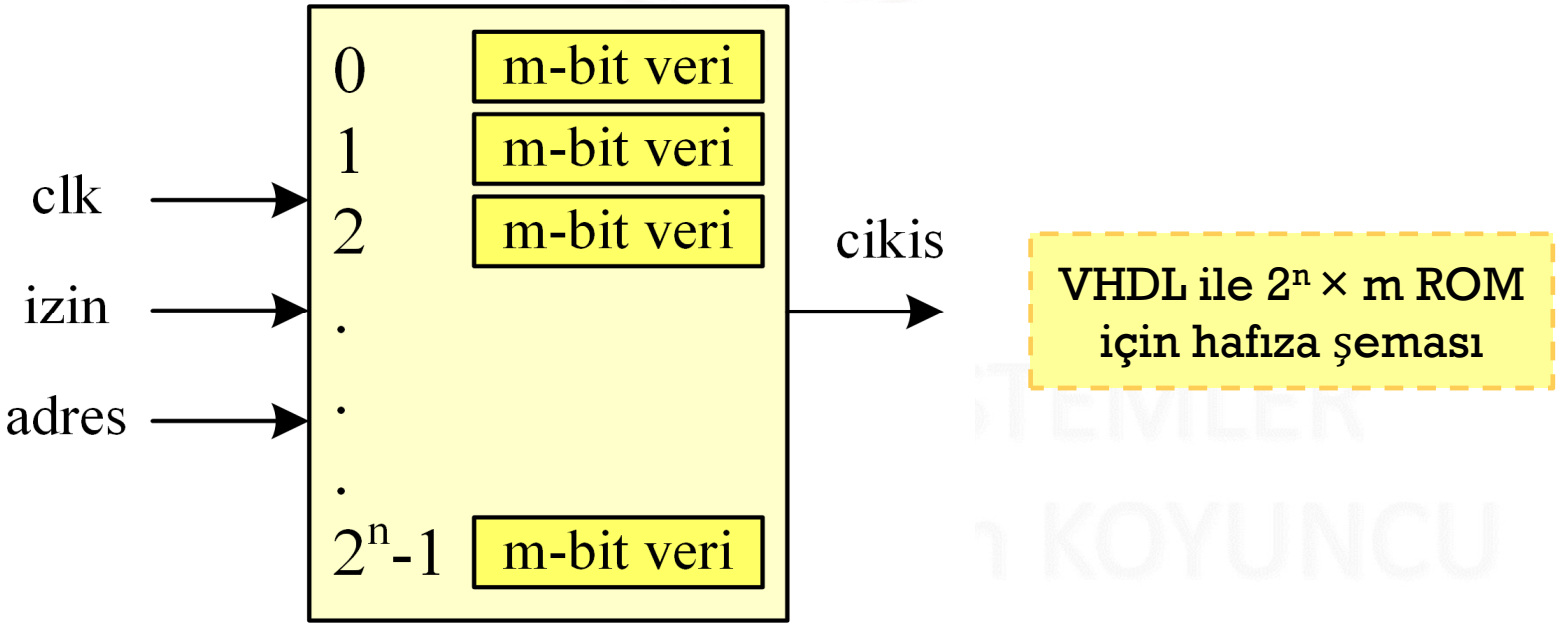
- Üretim esnasında doldurulan ve kolaylıkla değiştirilemeyen bir yapı içermektedir. VHDL üzerinden ROM blok şeması aşağıda görüldüğü gibidir.



11.1 ROM (Okunabilir) Hafıza

- Buna göre; her "**clk**" tetiğinde "**izin**" girişi (kontrol edilir) lojik-1 ise "**adres**" bilgisi okunarak, gerekli adresteki **m-bitlik veri** sistem çıkışına aktarılır.

- Aktarılan bu bilgi tasarım içi bir bölümde kullanılabilir veya genel bir sistem çıkışı olarak atfedilebilir.

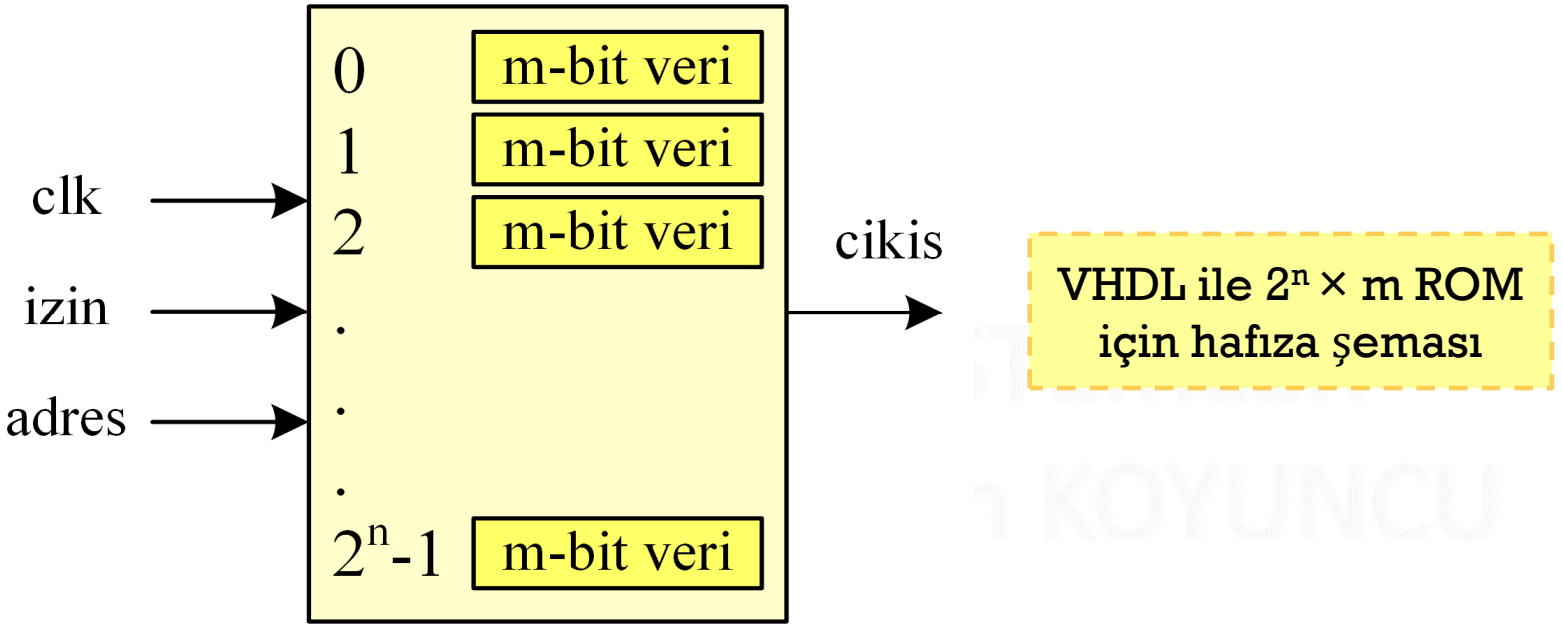


11.1 ROM (Okunabilir) Hafıza

- Sabit bilgilerin işletilmesi gerektiği birçok uygulamada ROM yapıları kullanılır.

- Mühendislikte karşılaşılan birçok yapıda **sabit bilgi işletilmesi** gerekir.

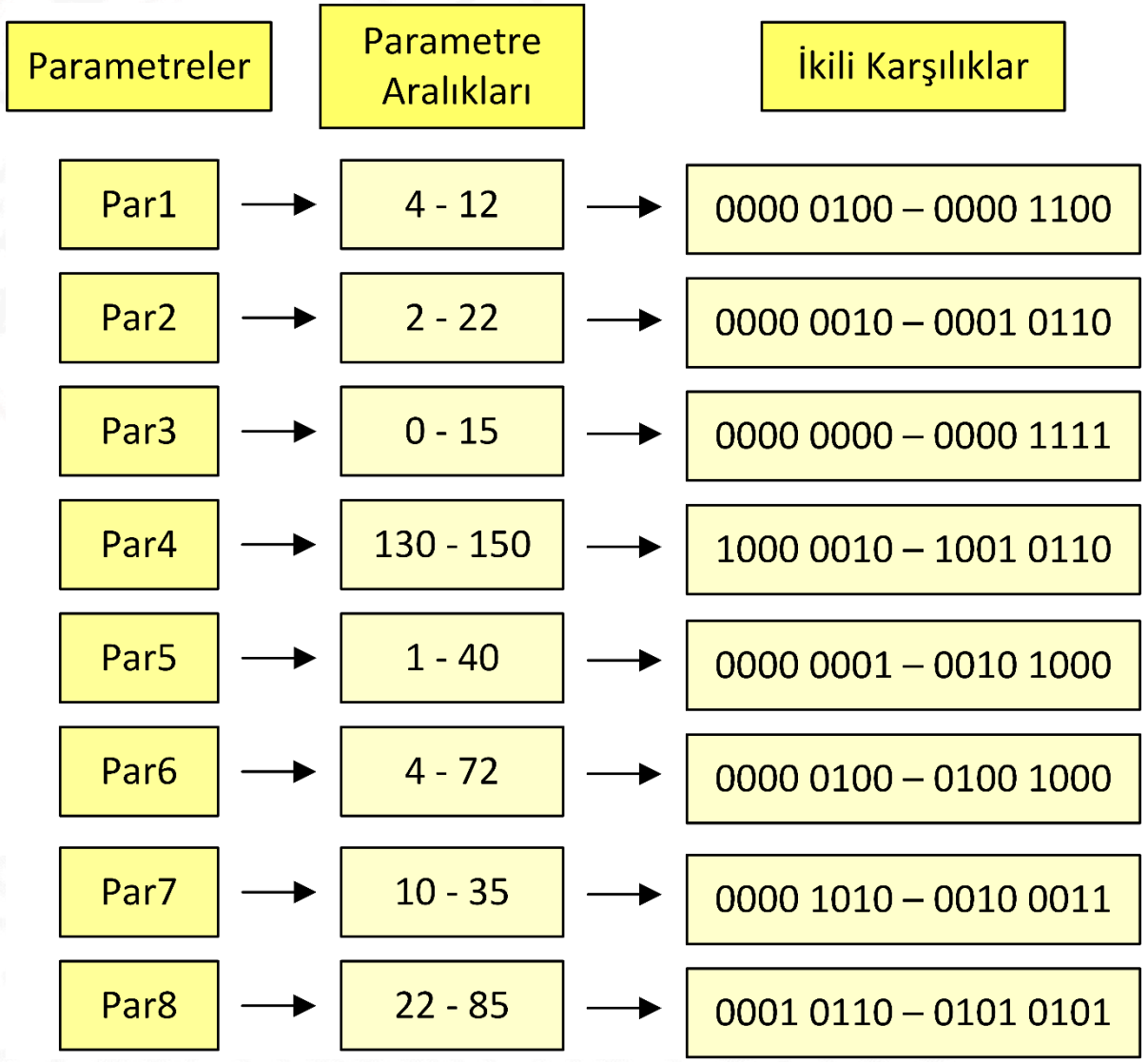
- Örneğin; alt ve üst sınır değerleri sabit olan 8 farklı parametreye dair sınır bilgisinin saklanması ve iletilmesi.



11.1 ROM (Okunabilir) Hafıza

Örnek 11.1: 8 adet parametrenin alt ve üst sınır değerleri şekilde görüldüğü gibidir. Buna göre ***alt ve üst değerler 8'er bit çıktı*** olmak üzere, bu ***sınır değerleri 16 bit formda saklayan*** ve **izin** girişine göre gerekli **adres**teki bilgiyi **alt** ve **ust** isimli çıkış portlarına **clk** tetiği ile aktaran sayısal devre için gerekli VHDL kodunu yazınız.

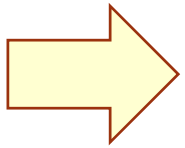
Not: **izin** girişi ***lojik-0*** iken veya **rst** girişi ***aktif*** iken, **alt** ve **ust** isimli çıkış portlarına ***gerekli bit sayısı kadar yüksek empedans ataması*** yapılacaktır.



11.1 ROM (Okunabilir) Hafıza


```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Use ieee.std_logic_unsigned.all;

Entity Uyg_rom is
Port (clk, rst, izin: in std_logic;
        adres: in std_logic_vector(2 downto 0);
        alt, ust: out std_logic_vector(7 downto 0));
End Uyg_rom;
```

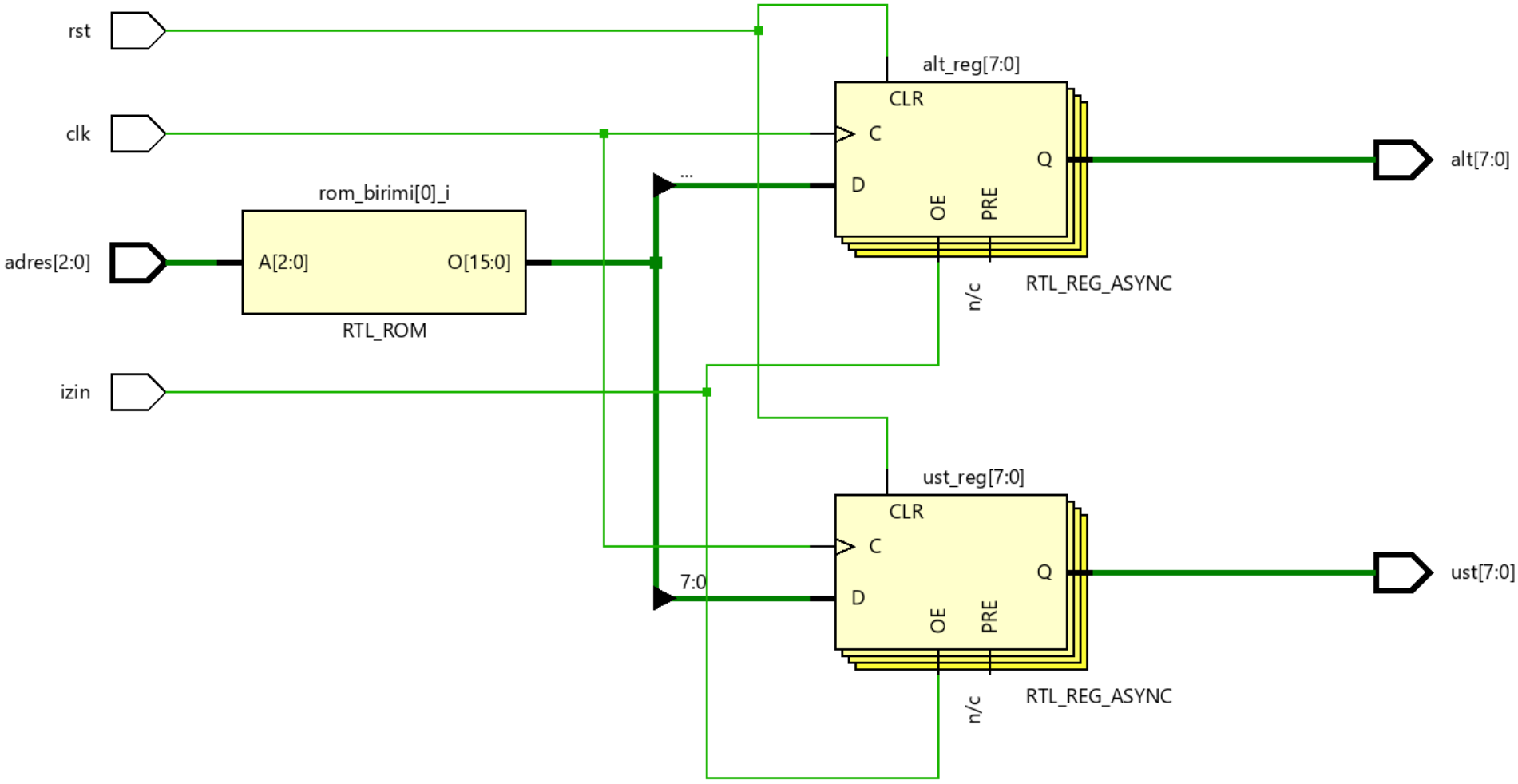
11.1 ROM (Okunabilir) Hafıza



```
Architecture davranis of Uyg_rom is
type bellek_tipi is array (0 to 7) of std_logic_vector(15 downto 0);
constant rom_birimi: bellek_tipi := (
   0 => "0000010000001100",
   1 => "0000001000010110",
   2 => "0000000000001111",
   3 => "1000001010010110",
   4 => "0000000100101000",
   5 => "0000010001001000",
   6 => "0000101000100011",
   7 => "0001011001010101");
Begin
   Process (clk,rst)
   Begin
      If (rst= '1') then
         alt <= (others=>'Z');
         ust <= (others=>'Z');
      Elsif rising_edge(clk) then
         If izin= '1' then
            alt <= rom_birimi(conv_integer(adres))(15 downto 8);
            ust <= rom_birimi(conv_integer(adres))(7 downto 0);
         Else
            alt <= (others=>'Z');
            ust <= (others=>'Z');
         End if;
      End if;
  End process;
End davranis;
```

11.1 ROM (Okunabilir) Hafıza

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



11.1 ROM (Okunabilir) Hafıza

Daha sade nasıl yazarız?

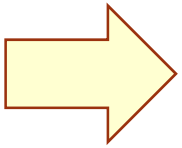
```
Architecture davranis of Uyg_rom is
type bellek_tipi is array (0 to 7) of std_logic_vector(15 downto 0);
constant rom_birimi: bellek_tipi := (
    0 => "0000 0100 0000 1100",
    1 => "0000 0010 0001 0110",
    2 => "0000 0000 0000 1111",
    3 => "1000 0010 1001 0110",
    4 => "0000 0001 0010 1000",
    5 => "0000 0100 0100 1000",
    6 => "0000 1010 0010 0011",
    7 => "0001 0110 0101 0101");
Begin
```

- x"040C"
- x"0216"
- x"000F"
- x"8296"
- x"0128"
- x"0448"
- x"0A23"
- x"1655"

Hexadecimal format ile

11.1 ROM (Okunabilir) Hafıza

Tasarım-2



Önceki örnekteki **Library** ve **Entity** bölümleri aynı kaldı.

Yine aynı RTL şeması elde edilir.

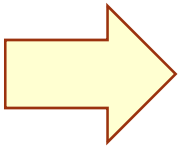
```
Architecture davranis of Uyg_rom is
type bellek_tipi is array (0 to 7) of std_logic_vector(15 downto 0);
constant rom_birimi: bellek_tipi := (0 => x"040C", 1 => x"0216",
2 => x"000F", 3 => x"8296", 4 => x"0128", 5 => x"0448", 6 =>
x"0A23", 7 => x"1655");
Begin
   Process (clk,rst)
   Begin
      If (rst= '1') then
         alt <= (others=>'Z');
         ust <= (others=>'Z');
      Elsif rising_edge(clk) then
         If izin= '1' then
            alt <= rom_birimi(conv_integer(adres))(15 downto 8);
            ust <= rom_birimi(conv_integer(adres))(7 downto 0);
         Else
            alt <= (others=>'Z');
            ust <= (others=>'Z');
         End if;
      End if;
   End process;
End davranis;
```

11.1 ROM (Okunabilir) Hafıza


```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity Uyg_rom is
Port (clk, rst, izin: in std_logic;
        adres: in std_logic_vector(2 downto 0);
        alt, ust: out std_logic_vector(7 downto 0));
End Uyg_rom;
```

11.1 ROM (Okunabilir) Hafıza

Tasarım-3



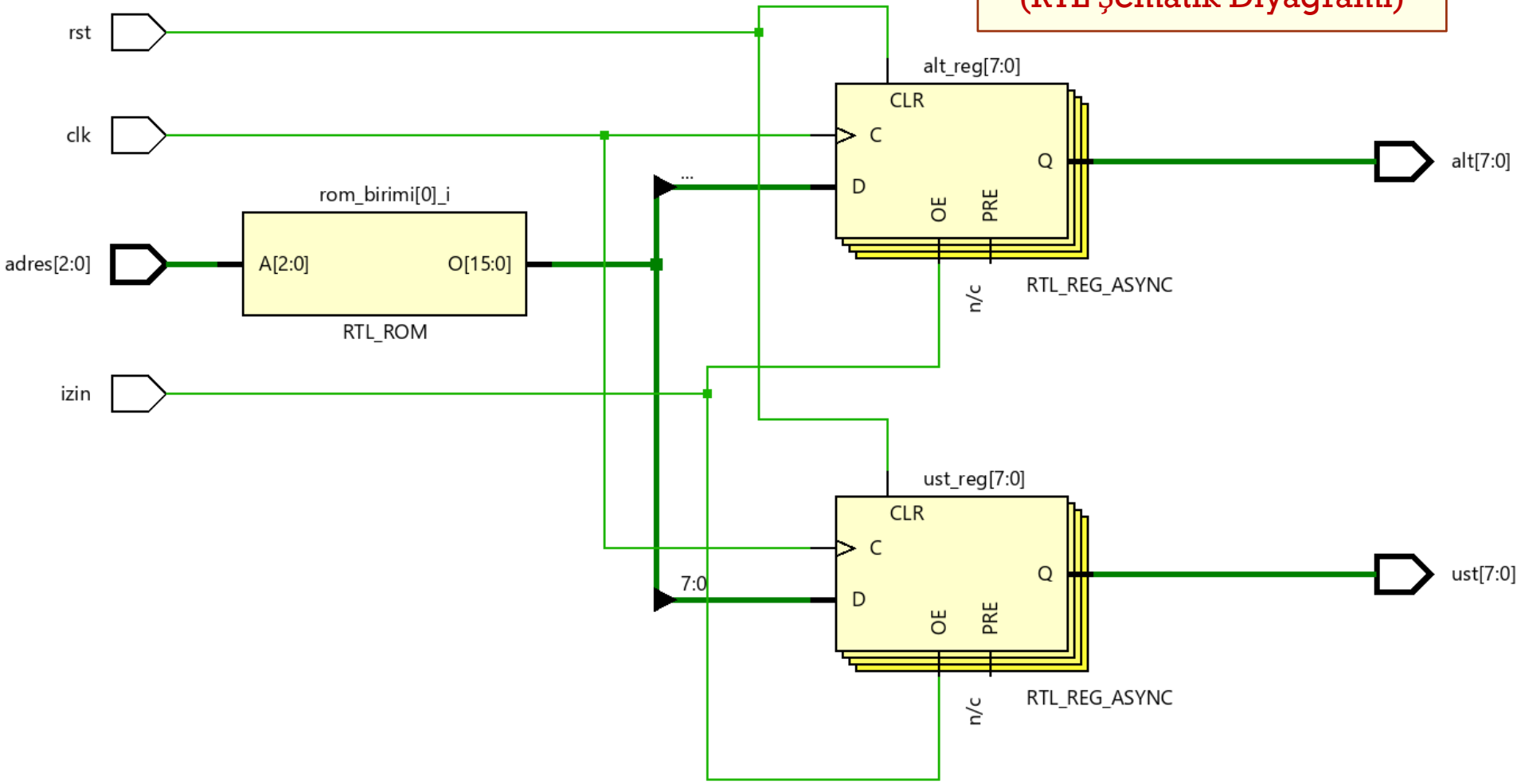
```
Architecture davranis of Uyg_rom is
type bellek_tipi is array (0 to 7) of std_logic_vector(15 downto 0);
constant rom_birimi: bellek_tipi := (0 => x"040C", 1 => x"0216", 2 =>
x"000F", 3 => x"8296", 4 => x"0128", 5 => x"0448", 6 => x"0A23",
7 => x"1655");
Begin
   Process (clk,rst)
   Begin
      If (rst= '1') then
         alt <= (others=>'Z');
         ust <= (others=>'Z');
      Elsif rising_edge(clk) then
         If izin= '1' then
            alt <= rom_birimi(to_integer(unsigned(adres)))(15 downto 8);
            ust <= rom_birimi(to_integer(unsigned(adres)))(7 downto 0);
         Else
            alt <= (others=>'Z');
            ust <= (others=>'Z');
         End if;
      End if;
  End process;
End davranis;
```

11.1 ROM (Okunabilir) Hafıza

Tasarım-3

Tasarım-1 ve Tasarım-2'deki
RTL şemasının aynısı elde edildi.

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



11.2 RAM (Rasgele Erişimli) Hafıza

- RAM birimi, hem okuma hem de yazma yapılabilen hafıza yapısıdır.

- İçerisinde **bilgi saklama hücreleri ile bu hücrelere veri aktarmak** veya **bu hücrelerden verileri okumak** amacıyla tasarlanmış devrelerden oluşur.

- Temel olarak **bir hafıza** ve **okuma / yazma seçimi sağlayan kontrol devresi *RAM yapısını teşkil eder.***

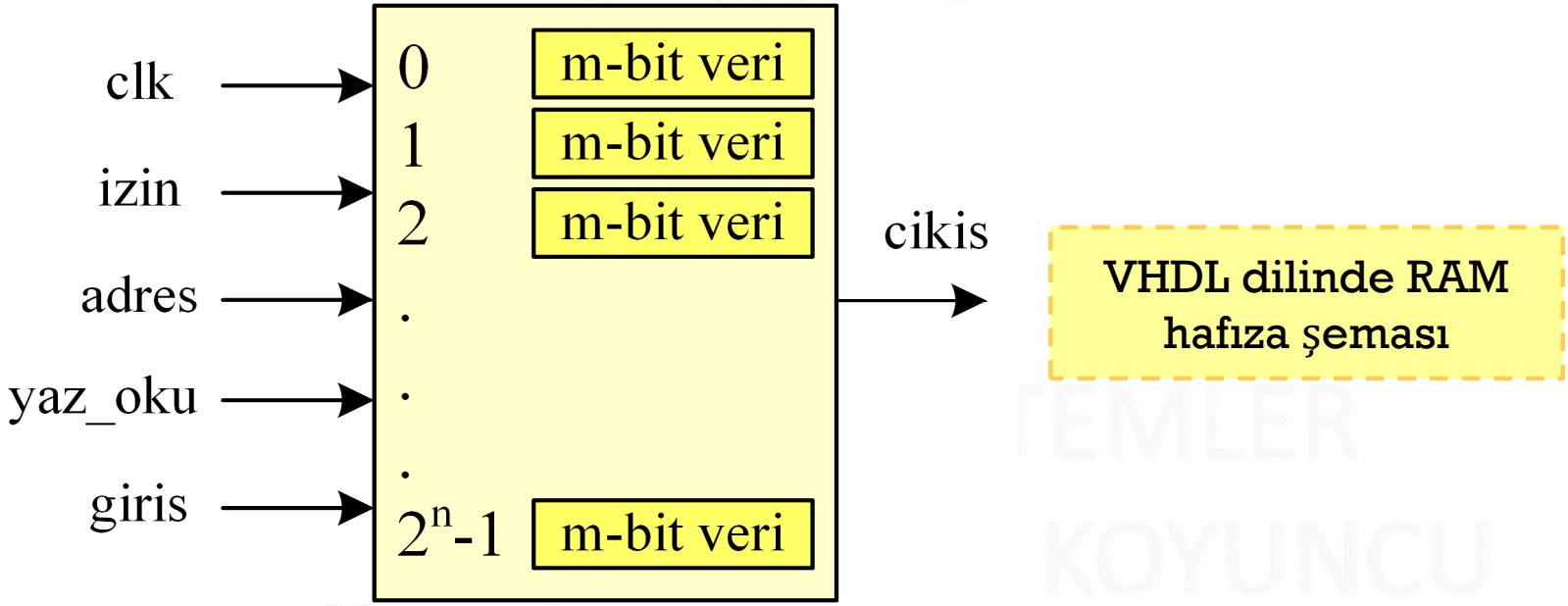
- Bilgiyi geçici olarak depolamaktadır.

- Açılımındaki rasgele erişim ifadesi ise ***sıra gözetmeksizin istenilen adreste okuma / yazma yapabilmesinden kaynaklıdır.***

11.2 RAM (Rasgele Erişimli) Hafıza

- VHDL üzerinden RAM blok şeması aşağıda görüldüğü gibidir.

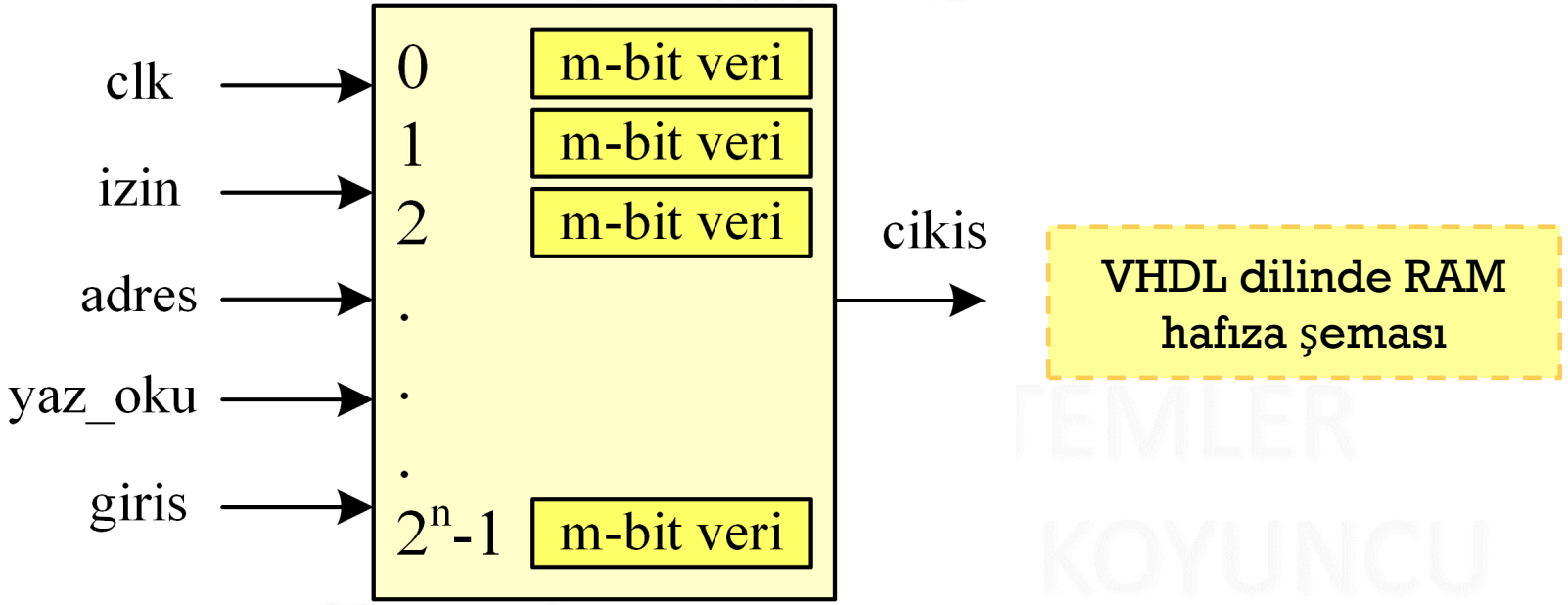
- Yapıya dair blok şemada görüldüğü üzere her "***clk***" tetiğinde "***izin***" girişi lojik-1 ise "***adres***" ve "***yaz_oku***" bilgileri okunmaktadır.



11.2 RAM (Rasgele Erişimli) Hafıza

- Bu bilgilere göre gerekli adresteki m-bitlik veri **okunabilmekte** (sistem çıkışına aktarılmakta)

- veya sistem girdisi gerekli adresteki m-bitlik veri yerine **aktarılmaktadır** (**yazılmaktadır**).



11.2 RAM (Rasgele Erişimli) Hafıza

Örnek 11.2: 8 adet 4 bitlik verilerin bulunacağı **(2^3x4) RAM bloğu** tasarlanacaktır.

- rst isimli giriş portu aktif olduğunda bütün RAM hafızası sıfırlanacaktır.

- clk isimli giriş port sinyalinin yükselen kenarında **izin** girişi kontrol edilecek;

* **lojik-0** ise **cikis** isimli çıkış portuna gerekli bit sayısı kadar yüksek empedans ataması yapılacaktır.

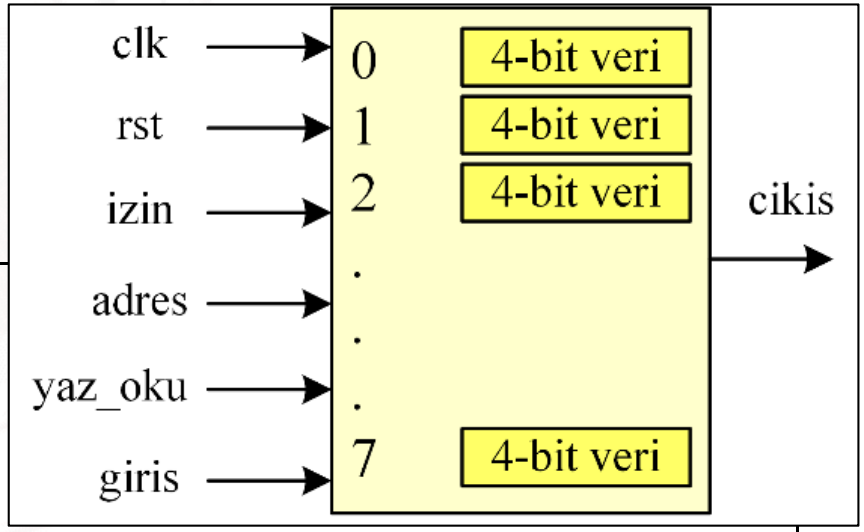
* **lojik-1** ise **yaz_oku** girdisine göre yazma veya okuma işlemi gerçekleştirilecektir.

- Okuma için gerekli adresteki bilgiyi **cikis** isimli çıkış portuna, yazma için ise **giris_dat** isimli giriş portundaki veriyi gerekli adrese yazan devre için gerekli VHDL kodunu yazınız.

(**Not: yaz_oku** portunda giriş '0' ise okuma, '1' ise yazma işlemi yapılacaktır)

11.2 RAM (Rasgele Erişimli) Hafıza

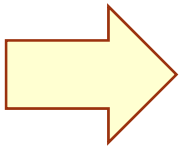
Tasarım-1




```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.std_logic_arith.all;
Use ieee.std_logic_unsigned.all;

Entity Uyg_ram is
Port (clk, rst, izin, yaz_oku: in std_logic;
        adres: in std_logic_vector(2 downto 0);
        giris_dat: in std_logic_vector(3 downto 0);
        cikis: out std_logic_vector(3 downto 0));
End Uyg_ram;
```

11.2 RAM (Rasgele Erişimli) Hafıza

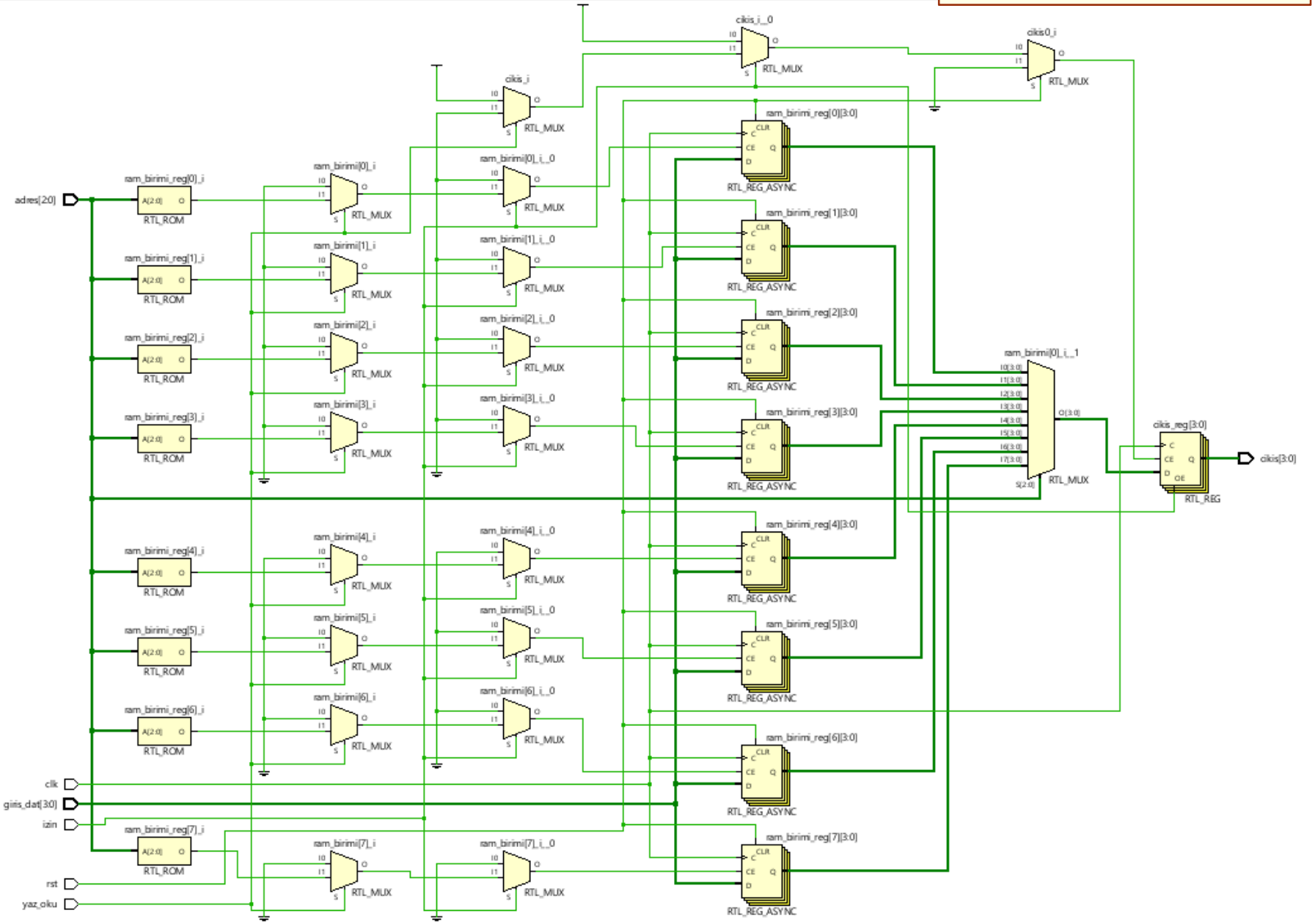
Tasarım-1



```
Architecture davranis of Uyg_ram is
type bellek_tipi is array (0 to 7) of std_logic_vector(3 downto 0);
signal ram_birimi: bellek_tipi;
Begin
   Process (clk, rst)
   Begin
      If (rst= '1') then
         ram_birimi <= (others=> (others=> '0') );
      Elsif rising_edge(clk) then
         If izin= '0' then
            cikis <= (others=>'Z');
         Elsif (izin= '1') then
            If yaz_oku = '0' then
               cikis <= ram_birimi(conv_integer(adres));
            Elsif yaz_oku = '1' then
               ram_birimi(conv_integer(adres)) <= giris_dat;
            End if;
         End if;
      End if;
  End process;
End davranis;
```

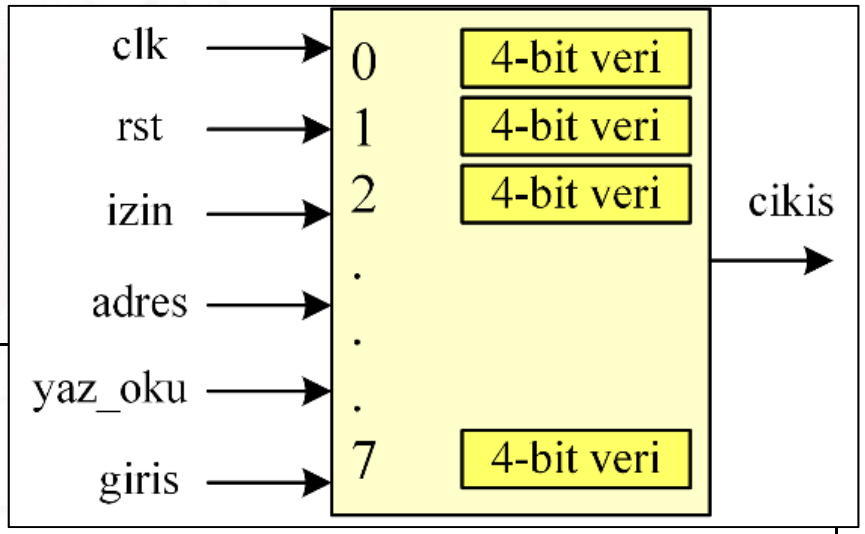
11.2 RAM (Rasgele Erişimli) Hafıza

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



11.2 RAM (Rasgele Erişimli) Hafıza

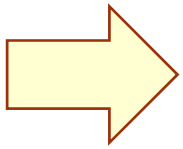
Tasarım-2




```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity Uyg_ram is
Port (clk, rst, izin, yaz_oku: in std_logic;
        adres: in std_logic_vector(2 downto 0);
        giris_dat: in std_logic_vector(3 downto 0);
        cikis: out std_logic_vector(3 downto 0));
End Uyg_ram;
```

11.2 RAM (Rasgele Erişimli) Hafıza

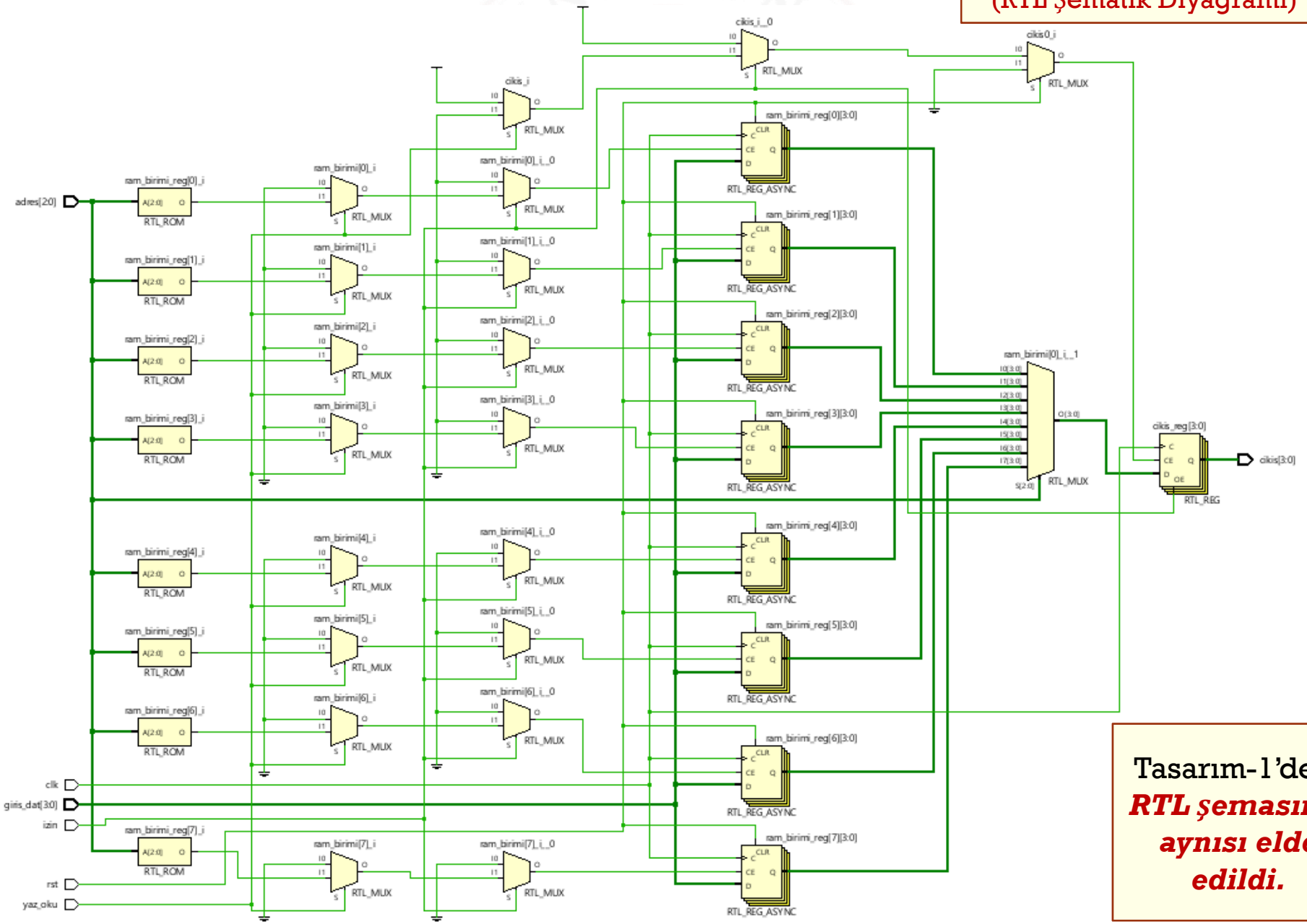
Tasarım-2



```
Architecture davranis of Uyg_ram is
type bellek_tipi is array (0 to 7) of std_logic_vector(3 downto 0);
signal ram_birimi: bellek_tipi;
Begin
   Process (clk, rst)
   Begin
      If (rst= '1') then
         ram_birimi <= (others=> (others=> '0') );
      Elsif rising_edge(clk) then
         If izin= '0' then
            cikis <= (others=>'Z');
         Elsif (izin= '1') then
            If yaz_oku = '0' then
               cikis <= ram_birimi(to_integer(unsigned(adres)));
            Elsif yaz_oku = '1' then
               ram_birimi(to_integer(unsigned(adres))) <= giris_dat;
            End if;
         End if;
      End if;
  End process;
End davranis;
```

11.2 RAM (Rasgele Erişimli) Hafıza

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)



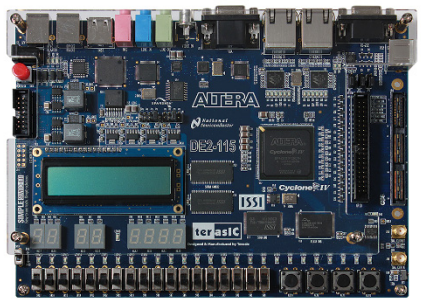
Tasarım-1'deki
RTL şemasının aynısı elde edildi.

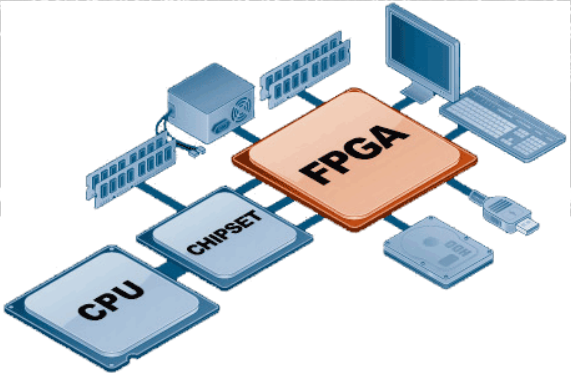
SONRAKİ DERS KONUSU

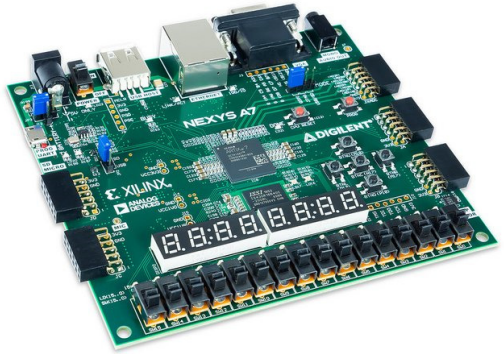
12- SONLU DURUM MAKİNELERİ

İLERİ SAYISAL SİSTEMLER

12- SONLU DURUM MAKİNELERİ







Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

12. SONLU DURUM MAKİNELERİ

- Sonlu durum makinesi (SDM); **sonlu (sınırlı) sayıda durumdan**, **durumlar arası geçişlerden** ve **eylemlerden** meydana gelen otomatik sistem modelidir.

- Durum makineleri, VHDL kapsamında genel olarak ***ardışıl sistem davranışlarını modellemek için*** kullanılır.

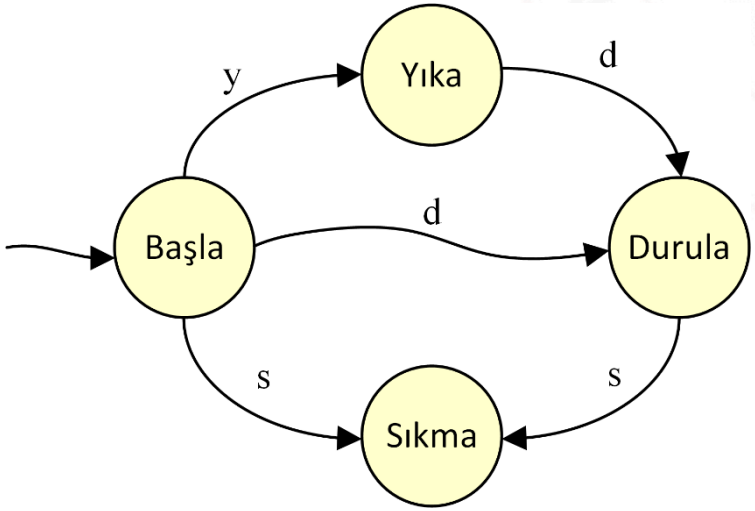
- SDM tipi tasarımlarda modellenecek ardışıl devre **durumlara ayrılır** ve bu **durumlar arası bağlantılar amaçlara uygun tanımlanır**.

- Sayısal tasarım kapsamında farklı SDM modelleri mevcuttur.

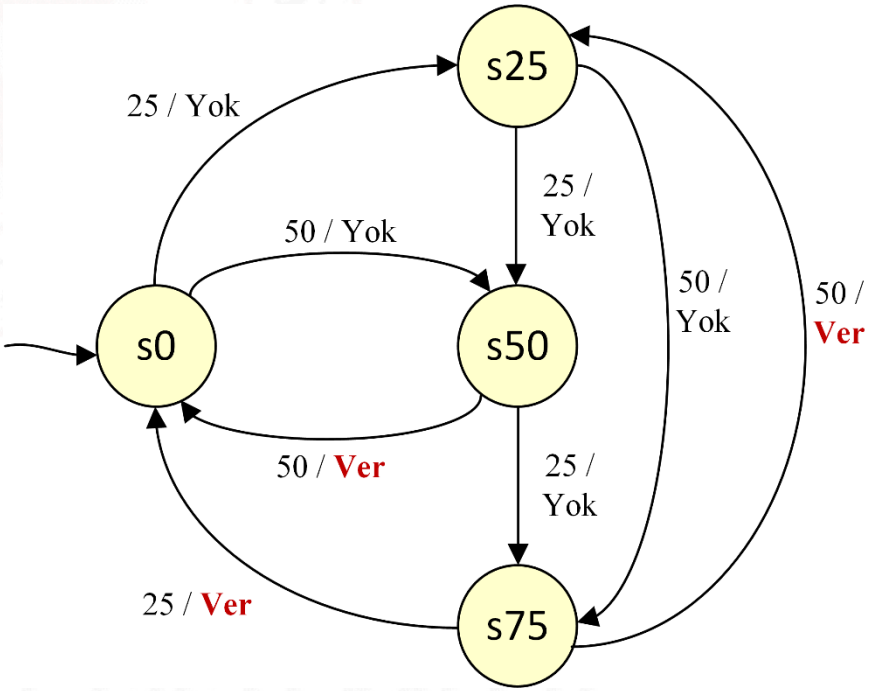
- Bu modellerden iki tanesi sıklıkla kullanılır ve bu iki model birbirine dönüştürülebilmektedir: **1-Moore SDM**, **2- Mealy SDM**.

12. SONLU DURUM MAKİNELERİ

- SDM yapıları oluşturulurken graflar kullanılır.



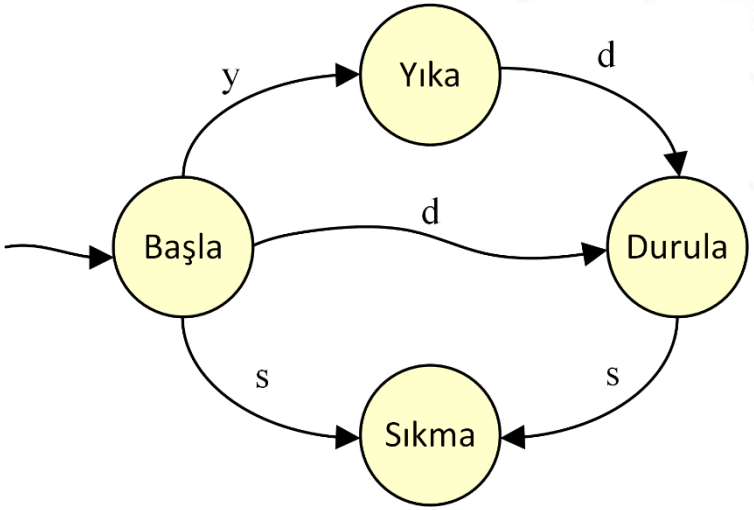
Moore makinesi temelli tasarlanan çamaşır makinesi kontrol birimi şeması



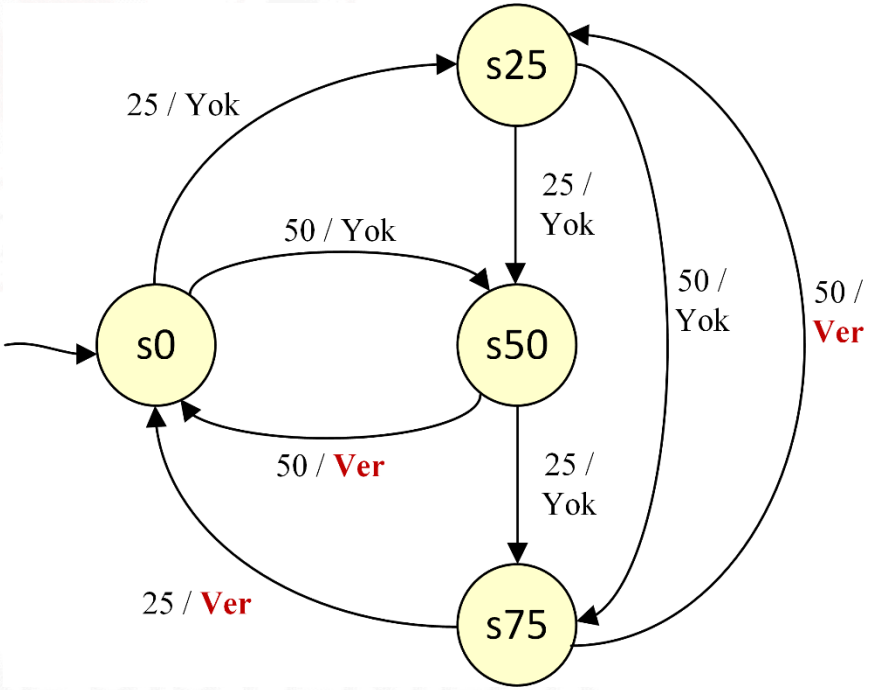
Mealy makinesine göre tasarlanan satış otomatı kontrol birimi şeması

12. SONLU DURUM MAKİNELERİ

- Moore tipi makinelerde çıkış değerleri durumlara / düğümlere yazılırken, okunan semboller kenarlar / geçişler üzerinde gösterilir.



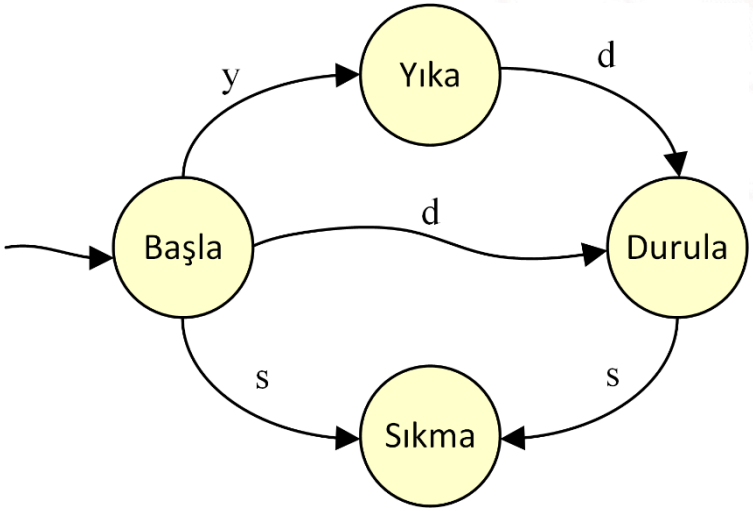
Moore makinesi temelli tasarlanan çamaşır makinesi kontrol birimi şeması



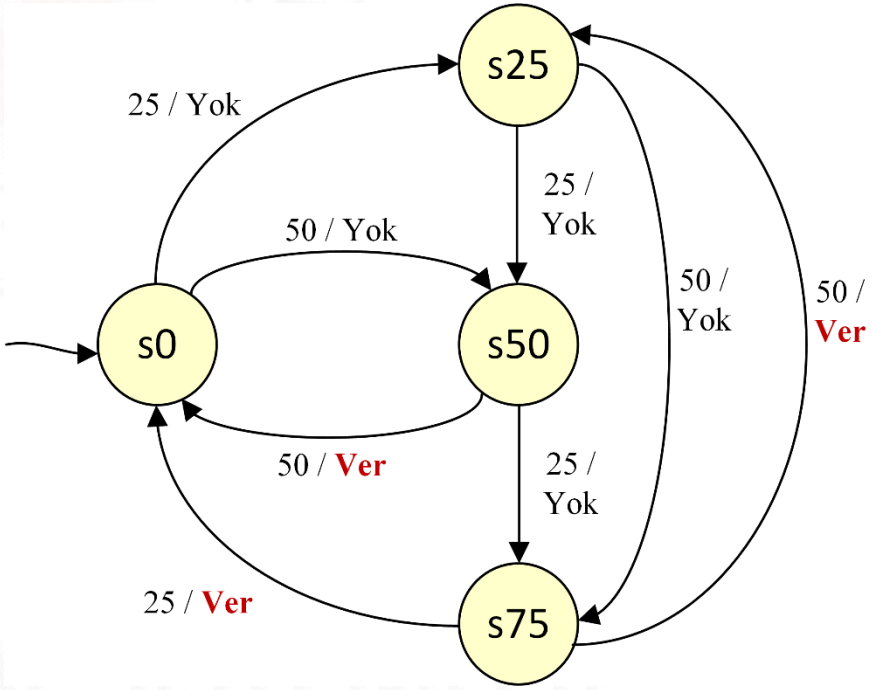
Mealy makinesine göre tasarlanan satış otomatı kontrol birimi şeması

12. SONLU DURUM MAKİNELERİ

- Mealy tipi makinelerde ise giriş / çıkış değerleri kenarlar üzerinde ifade edilir.



Moore makinesi temelli tasarlanan çamaşır makinesi kontrol birimi şeması

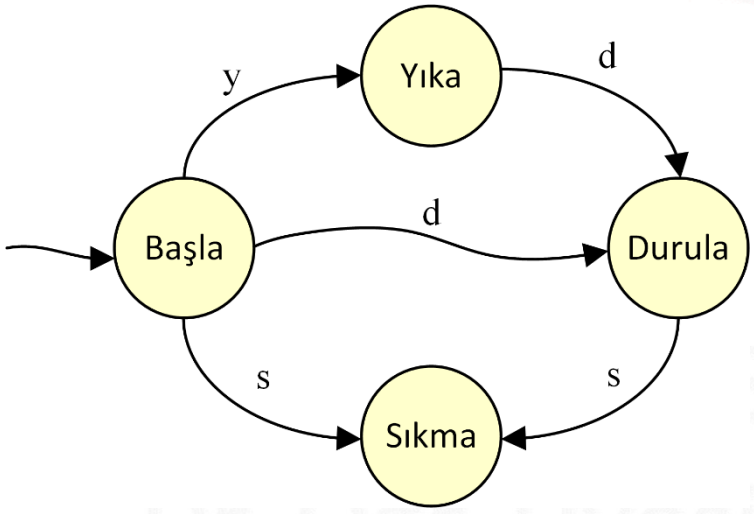


Mealy makinesine göre tasarlanan satış otomatı kontrol birimi şeması

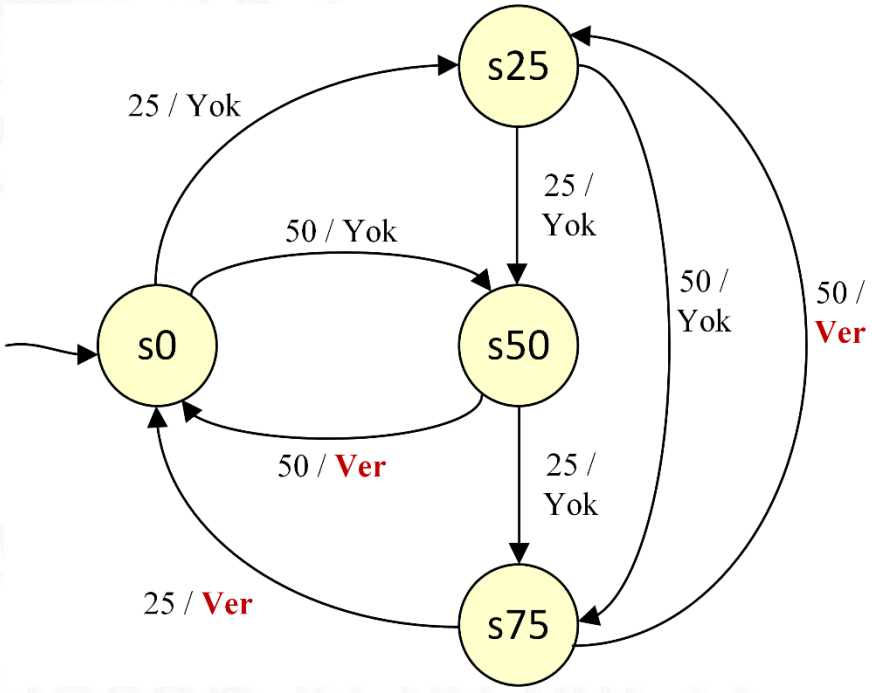
12. SONLU DURUM MAKİNELERİ

Moore ve Mealy makinelerinin farklılıkları şunlardır:

- Moore makineleri daha kolay tasarlanır.



Moore SDM

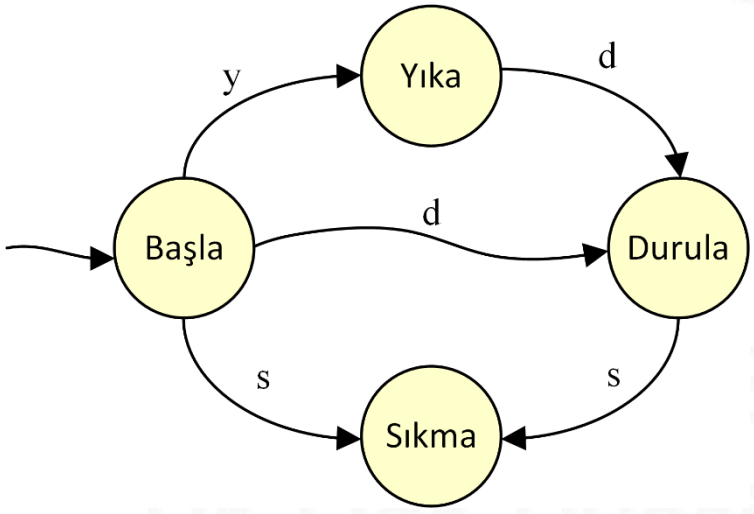


Mealy SDM

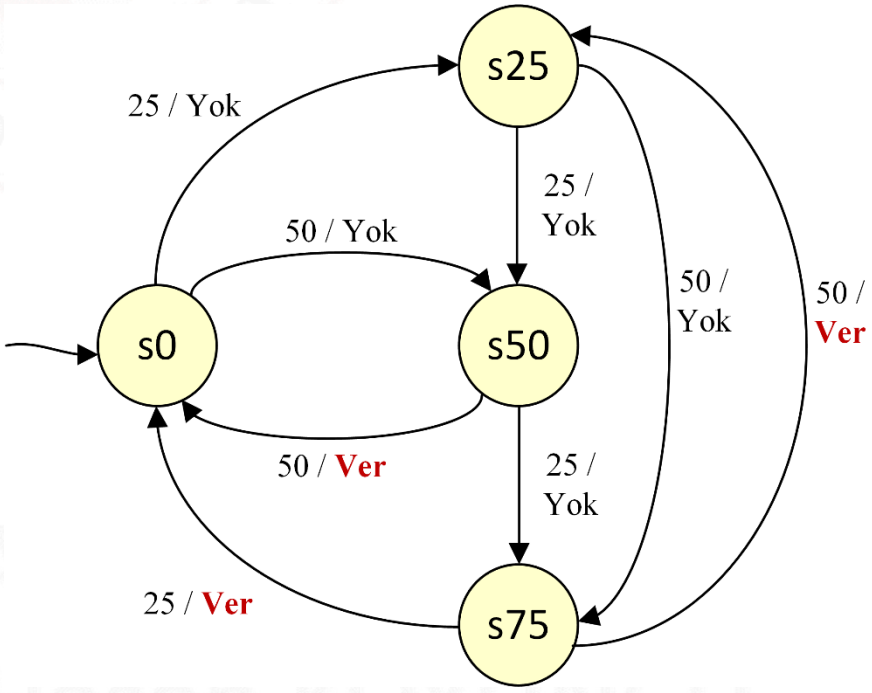
12. SONLU DURUM MAKİNELERİ

Moore ve Mealy makinelerinin farklılıkları şunlardır:

- Mealy makineleri genelde daha az durum kümesi üzerinden tasarlanır.



Moore SDM

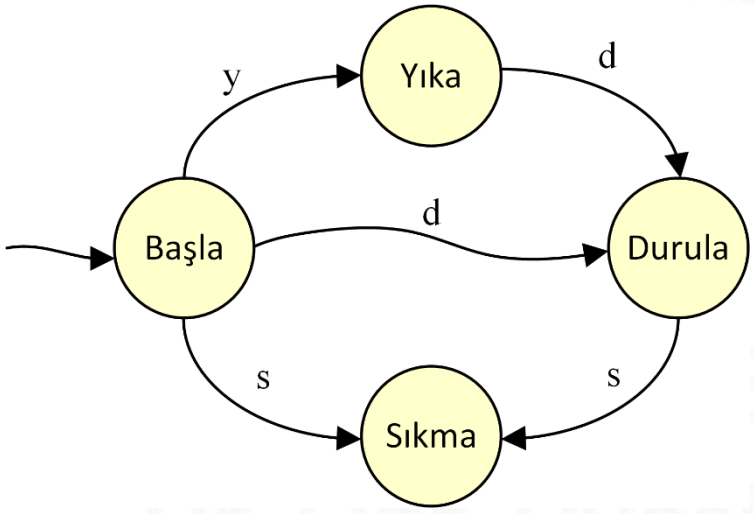


Mealy SDM

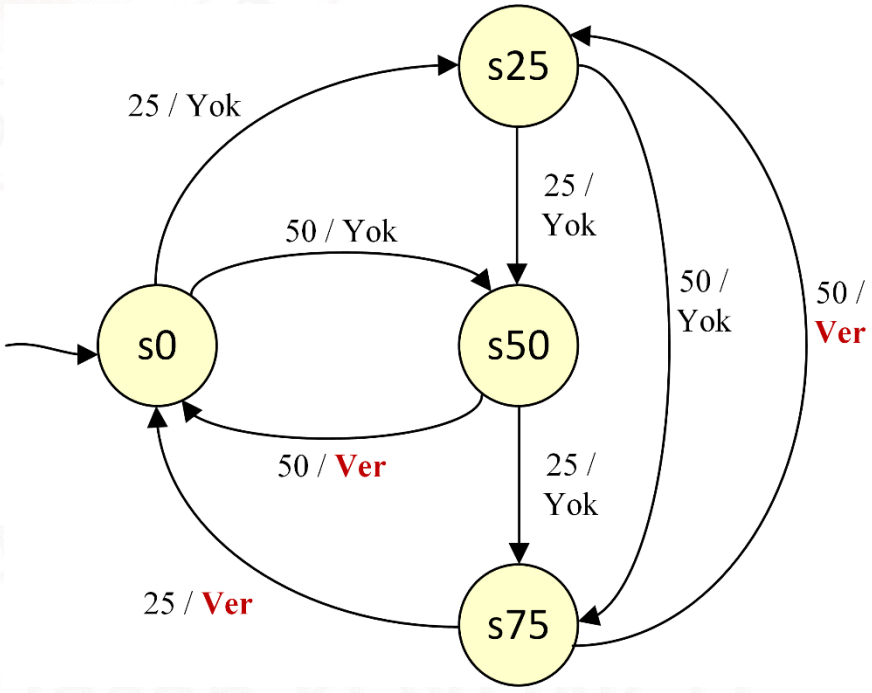
12. SONLU DURUM MAKİNELERİ

Moore ve Mealy makinelerinin farklılıkları şunlardır:

- Mealy makinesinde çıkışlar geçişlerde belirtilir.



Moore SDM

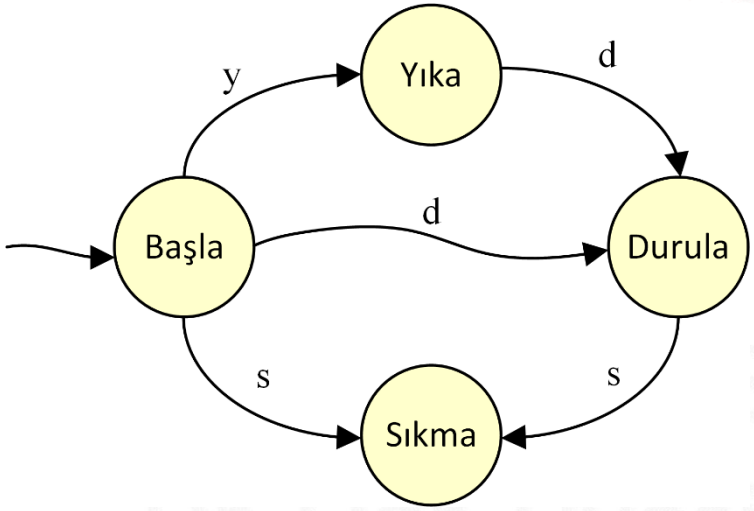


Mealy SDM

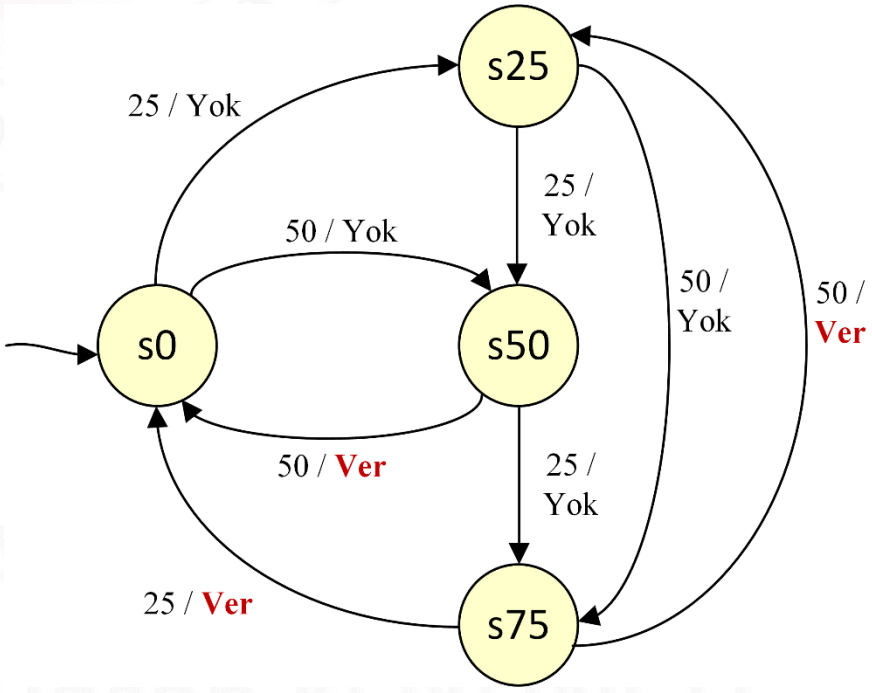
12. SONLU DURUM MAKİNELERİ

Moore ve Mealy makinelerinin farklılıkları şunlardır:

- Moore makinesinde çıkışları görmek için makinenin anlık durumuna bakmak gerekir.



Moore SDM

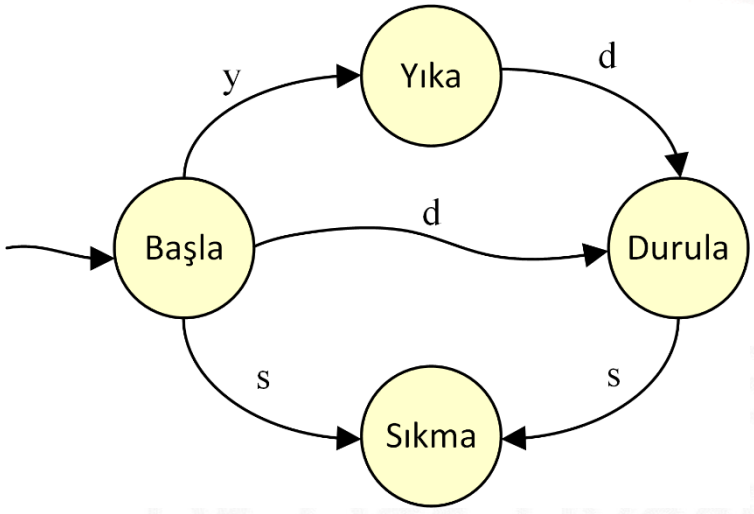


Mealy SDM

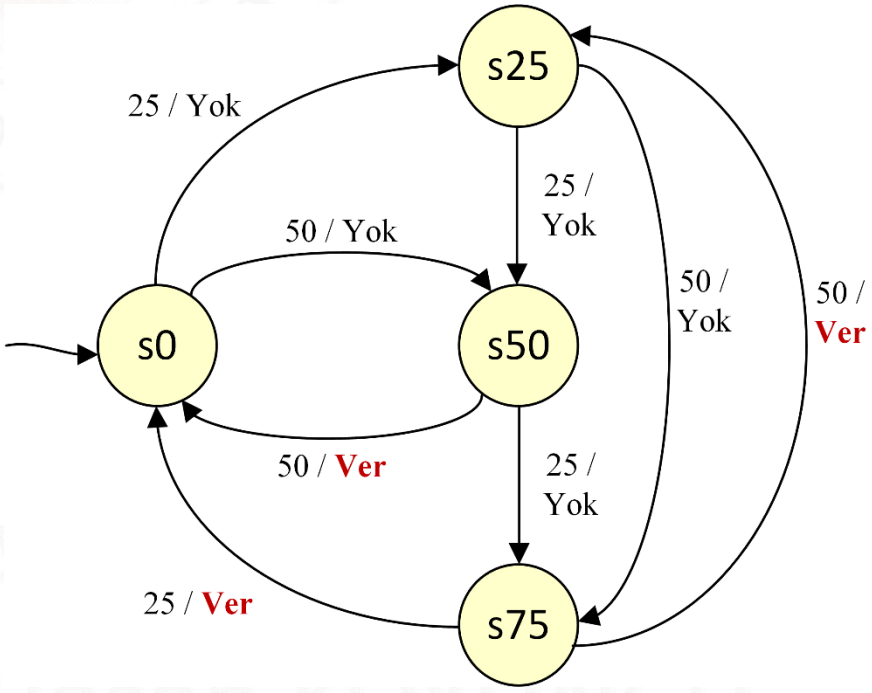
12. SONLU DURUM MAKİNELERİ

Moore ve Mealy makinelerinin farklılıkları şunlardır:

- Mealy makinesinde birden fazla giriş / çıkış bağlantısı kolaylıkla tanımlanabildiğinden çok karmaşık makine tasarımlarında kullanılırlar.



Moore SDM



Mealy SDM

12.1 Durumların Tanımlanması

- Komplike ardışıl sistemlerde herbir tasarım evresi, **SDM kapsamında ayrı bir durum** olarak ifade edilebilir.

- Ardışıl devreler, ***şimdiki durum bilgisini kullanarak bir sonraki durumda hangi işlemi yapacağını bilir.***

- Tasarımcı tarafından bütün durumlar tespit edildikten sonra, ***durumları birbirinden ayırt etmek için farklı ifadeler kullanılır.***

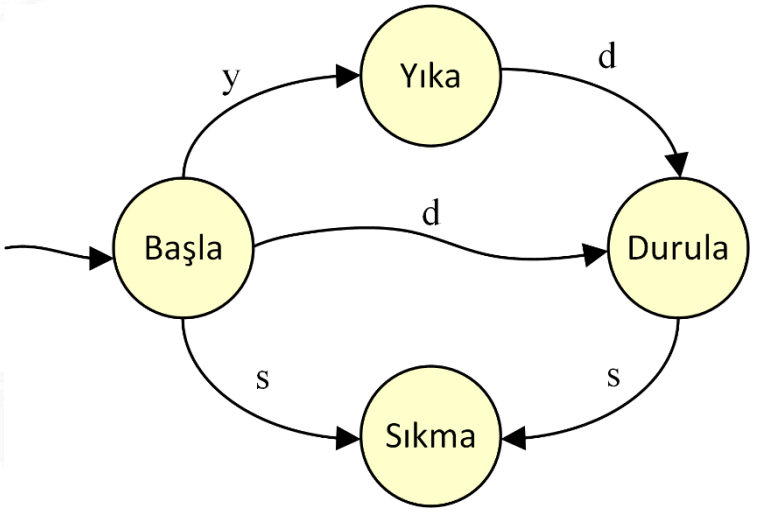
- VHDL' de farklı durumları ifade etmek için sıralı tür tanımlamasının kullanılması gerekir.

12.1 Durumların Tanımlanması

▪VHDL' de farklı durumları ifade etmek için sıralı tür tanımlamasının kullanılması gerekir.

▪ Şekildeki çamaşır makinesi kontrol devresinde gerekli durum için aşağıdaki kod tanımlaması yapılabilir.

Type durumlar **is** (basla, yika, durula, skm);



Moore SDM

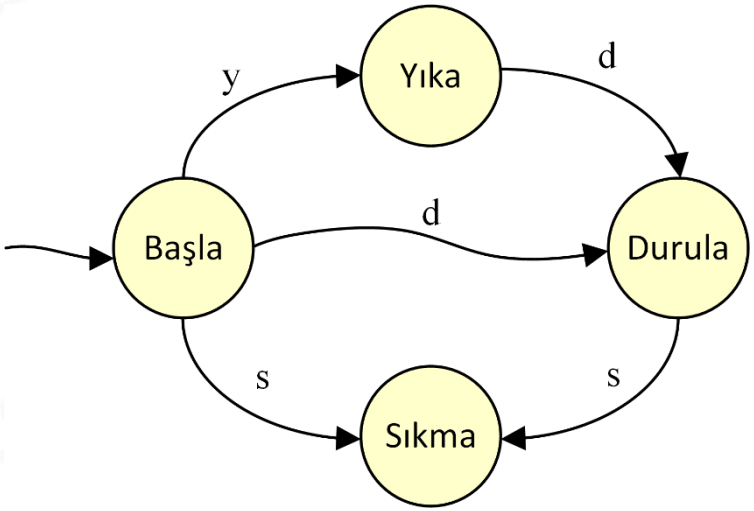
12.2 Moore ve Mealy Makineleri

- Sentez aşamasında sıralı türdeki herbir duruma karşılık uygun sayılar verilir.

- Örneğin aşağıdaki şekilde, Moore SDM ile çamaşır makinesi kontrolünde ifade edilen durumlara karşılık basla → "00", yika → "01", durula → "10", skm → "11" şeklinde atama yapılabilir.

- SDM analizinde şimdiki durumu muhafaza etmek için yukarıda verilen sıralı tür tipinde sinyal tanımlaması yapılır.

Type durumlar **is** (basla, yika, durula, skm);



Moore SDM

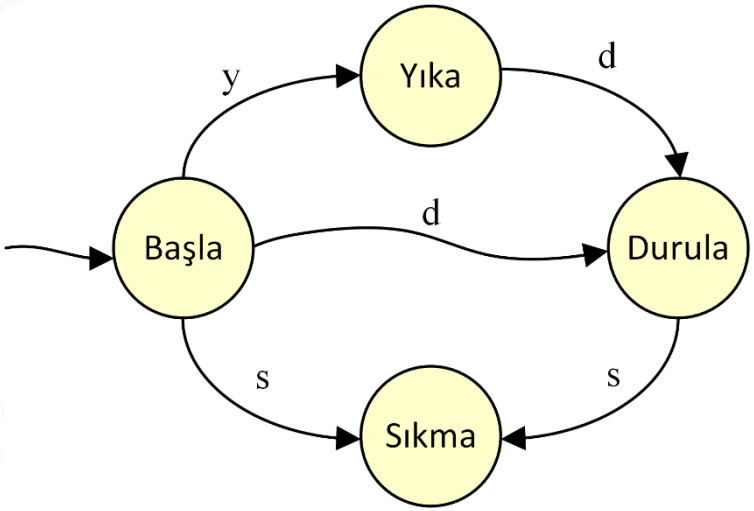
12.2 Moore ve Mealy Makineleri

- Sentez aşamasında sıralı türdeki herbir duruma karşılık uygun sayılar verilir.

- Örneğin aşağıdaki şekilde, Moore SDM ile çamaşır makinesi kontrolünde ifade edilen durumlara karşılık basla → "00", yika → "01", durula → "10", skm → "11" şeklinde atama yapılabilir.

- Durumlar içinde başlangıç durumu (makinenin ilk durumu), sinyale ilk değer ataması olarak aktarılabilir.

Signal simdi_d: *durumlar* := basla;



Moore SDM

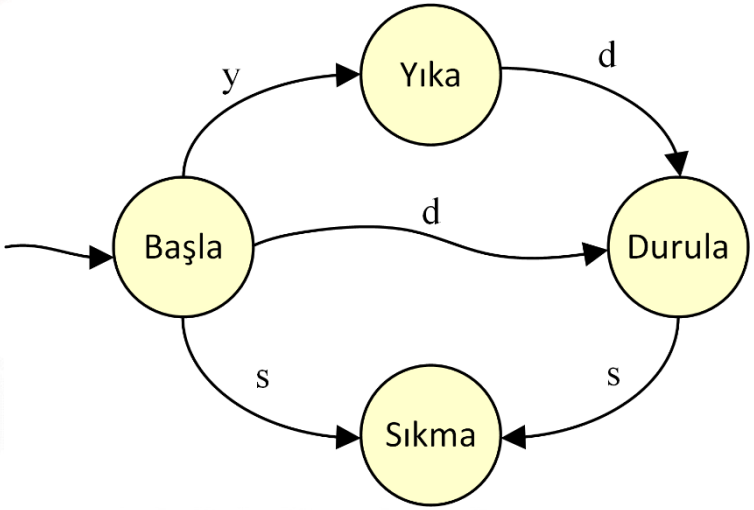
12.2 Moore ve Mealy Makineleri

- Şekil incelendiğinde, çamaşır makinesine dair 4 farklı çalışma durumu ve bu durumlar arası geçişleri gösteren 5 farklı ok bulunduğu görülür.

- İlk durum olan "başla" durumu, boşluktan gelen bir ok ile yetkilendirilmiştir.

- "Sıkma" durumu makine için son durumu göstermektedir.

- Geçişler üzerindeki harfler (*y,d,s*), bir durumdan diğerine geçme için işletilen komutlardır.



Moore SDM

12.2 Moore ve Mealy Makineleri

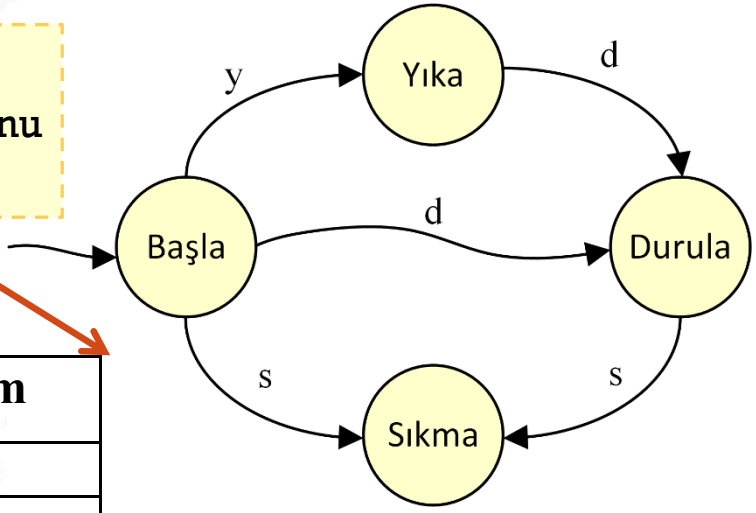
- Durumları ve geçiş komutları bilinen bir SDM analizinde durum-geçiş tablosu' nun oluşturulması hem diyagramın analizinde hem de kod aktarımında kolaylık sağlamaktadır.

- Aşağıdaki tablo, şekildeki çamaşır makinesi kontrol diyagramı için üretilen durum-geçiş tablosunu göstermektedir.

Gerekli ise **Çıkışlar** sütunu eklenebilir.

Çamaşır makinesi kontrolü için durum-geçiş tablosu

Şimdiki Durum	Geçiş Komutu	Yeni Durum
Başla	***y*** **– "00"**	Yıka
Başla	***d*** **– "01"**	Durula
Başla	***s*** **– "10"**	Sıkma
Yıka	***d*** **– "01"**	Durula
Durula	***s*** **– "10"**	Sıkma



Moore SDM

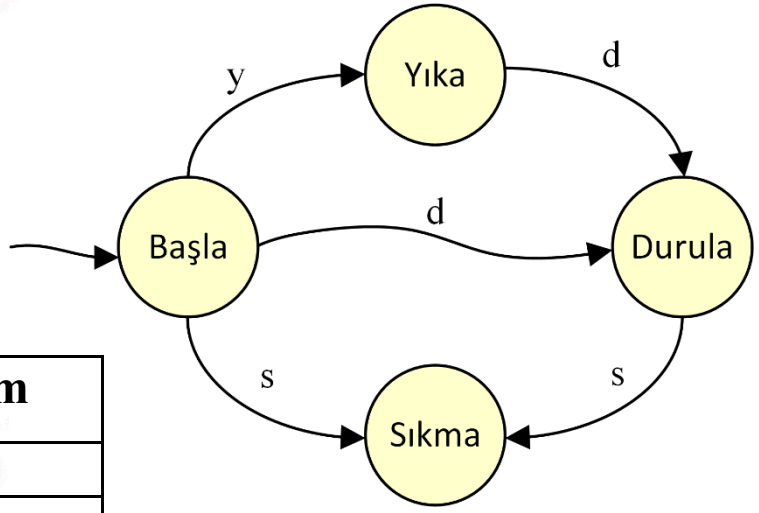
12.2 Moore ve Mealy Makineleri

- Tablo incelenirse tasarlanan SDM, "***yds***", "***ds***" ve "***s***" kontrol kelimelerini kavrayabilmektedir.

- VHDL'de bulunulan durumları ifade etmek için saklayıcı kullanılır.

Çamaşır makinesi kontrolü için durum-geçiş tablosu

Şimdiki Durum	Geçiş Komutu	Yeni Durum
Başla	***y*** **– "00"**	Yıka
Başla	***d*** **– "01"**	Durula
Başla	***s*** **– "10"**	Sıkma
Yıka	***d*** **– "01"**	Durula
Durula	***s*** **– "10"**	Sıkma



Moore SDM

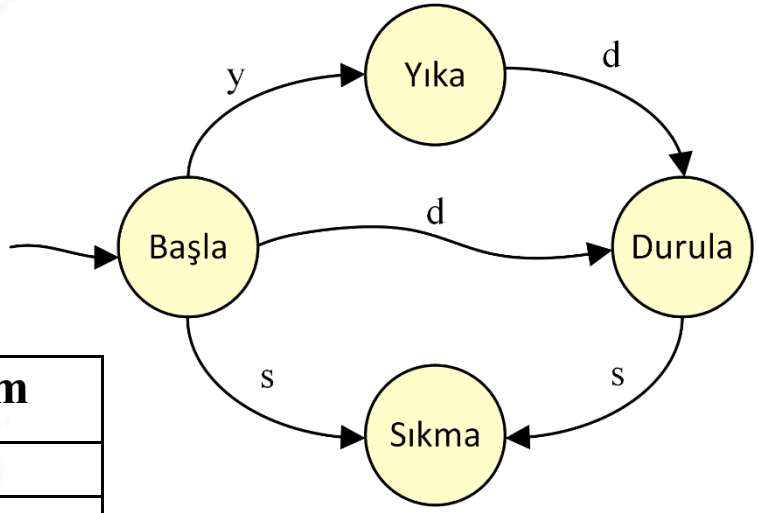
12.2 Moore ve Mealy Makineleri

- Burada "**n**" gerekli hafıza birimi sayısı olmak üzere, "**$2^n \geq$ durum sayısı**" kuralına göre işlem yapılır.

- Yine main VHDL kodunda, ardışıl kontrollerin sağlanabilmesi için en az bir adet **process** bloğu kullanılması gerekir.

Çamaşır makinesi kontrolü için durum-geçiş tablosu

Şimdiki Durum	Geçiş Komutu	Yeni Durum
Başla	***y*** **– "00"**	Yıka
Başla	***d*** **– "01"**	Durula
Başla	***s*** **– "10"**	Sıkma
Yıka	***d*** **– "01"**	Durula
Durula	***s*** **– "10"**	Sıkma



Moore SDM

12.2 Moore ve Mealy Makineleri

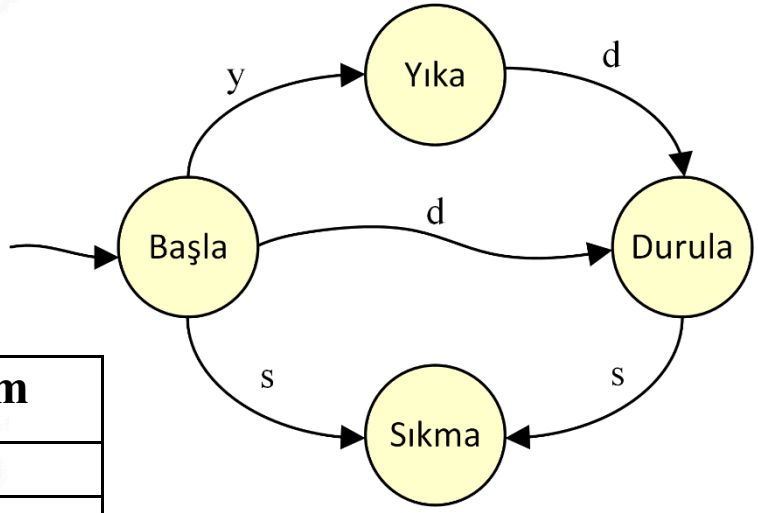
- Şekil için gerekli VHDL kodunu yazalım. Bu aşamada herbir durum alttaki komut ile tanımlanır. İkinci komut ile de şimdiki duruma ilk değer ataması yapılır.

1- Type durumlar **is** (basla, yika, durula, skm);

2- Signal simdi_d: *durumlar* := basla;

Çamaşır makinesi kontrolü için durum-geçiş tablosu

Şimdiki Durum	Geçiş Komutu	Yeni Durum
Başla	***y*** **– "00"**	Yıka
Başla	***d*** **– "01"**	Durula
Başla	***s*** **– "10"**	Sıkma
Yıka	***d*** **– "01"**	Durula
Durula	***s*** **– "10"**	Sıkma



Moore SDM

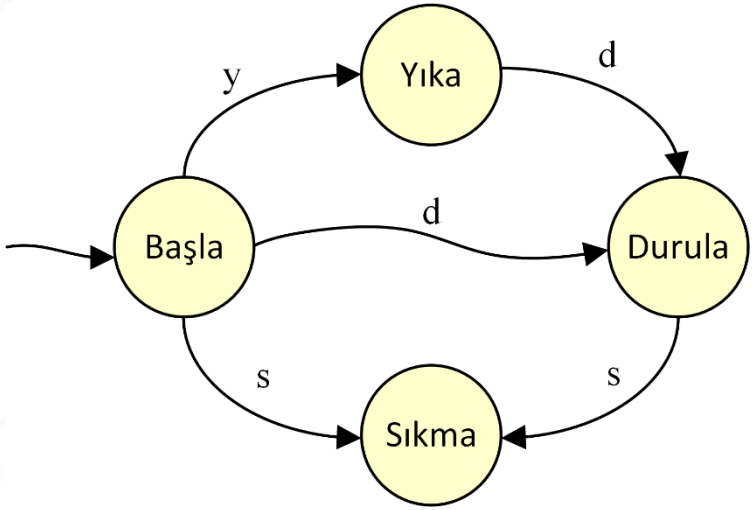
12.2 Moore ve Mealy Makineleri

- Bu örnekte durumların herbirisi ayrı bir çıkış olarak görülebilir ve bu olayda Moore makinesi kullanmak kolaylık sağlar.

- Durumlar arası geçiş için "***y,d,s***" kontrol kelimelerinden birinin kullanılması gereklidir.

- Bu kelimelerin toplam sayısı 3 olduğundan iki bit giriş üzerinden kodlamaları yapılabilir.

- Buna göre; "***y***" için "**00**", "***d***" için "**01**" ve "***s***" için "**10**" ikili değerlerine göre kod yazılabilir.



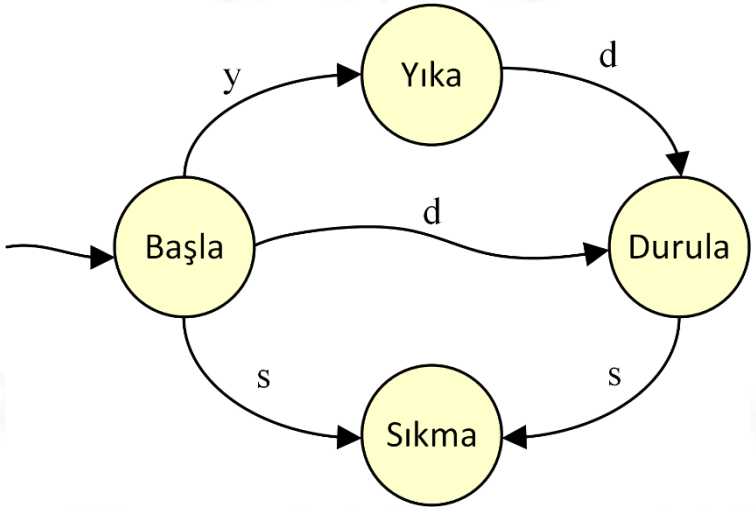
Moore SDM

$2^n \geq$ giriş sayısı

12.2 Moore ve Mealy Makineleri

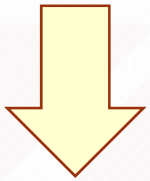
- **Clk** sinyalinin yükselen kenarında iki bitlik "**giris**" isimli giriş port bilgisi okunacak, sonrasında şimdiki durum sinyaline yeni durum ataması yapılacaktır.

- Sistem çıkışı ise '0' (başla, yıka, durula) veya '1' (sıkma) seviyelerine sahip olacaktır.



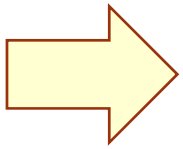
Moore SDM

12.2 Moore ve Mealy Makineleri



```
Library ieee;
Use ieee.std_logic_1164.all;

Entity camasir_sdm is
Port (clk : in std_logic;
        giris : in std_logic_vector(1 downto 0);
        cikis : out std_logic);
End camasir_sdm;

Architecture davranis of  camasir_sdm is
Type durumlar is (basla, yika, durula, skm);
Signal simdiki_d: durumlar := basla;
Begin
```

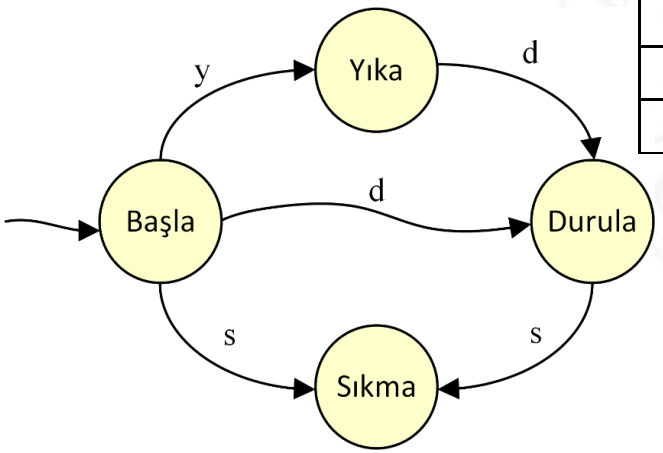
12.2 Moore ve Mealy Makineleri



```
Process (clk)
Begin
   If rising_edge(clk) then
      Case simdiki_d is  -- "y" için "00", "d" için "01" ve "s" için "10"
         When basla =>
            If (giris="00") then
               cikis <= '0';
               simdiki_d <= yika;
            Elsif (giris="01") then
               cikis <= '0';
               simdiki_d <= durula;
            Elsif (giris="10") then
               cikis <= '0';
               simdiki_d <= skm;
            End if;
         When yika =>
            If (giris="01") then
               cikis <= '0';
               simdiki_d <= durula;
            End if;
         When durula =>
            If (giris="10") then
               cikis <= '0';
               simdiki_d <= skm;
            End if;
         When skm =>
               cikis <= '1';
      End case;
   End if;
End process;
End davranis;
```

12.2 Moore ve Mealy Makineleri

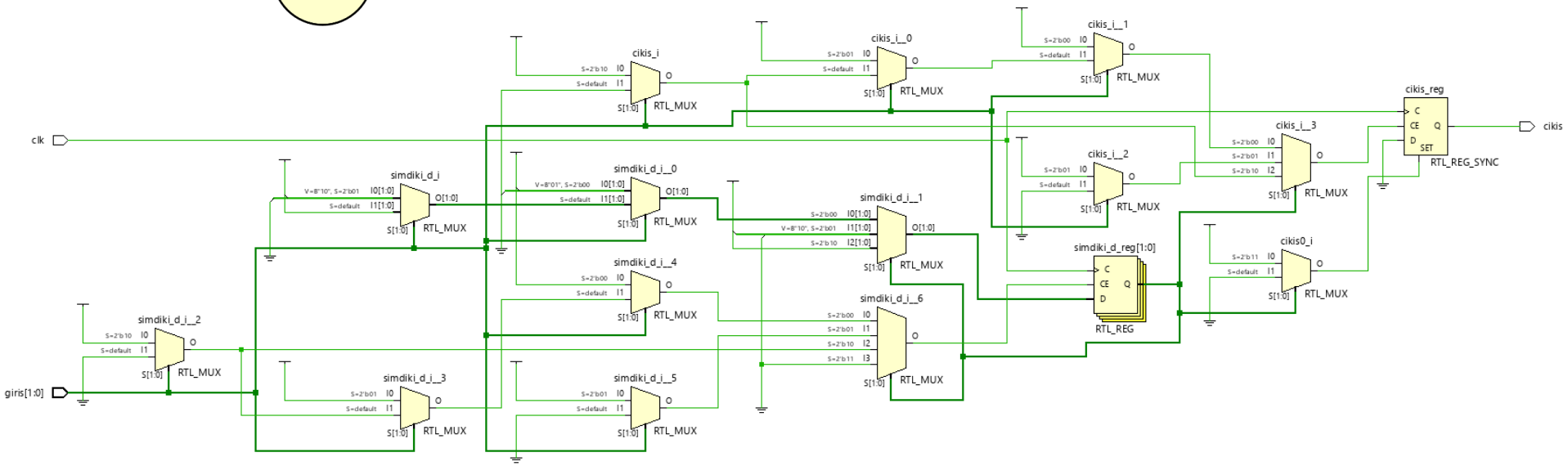
Moore SDM



Çamaşır makinesi kontrolü için durum-geçiş tablosu

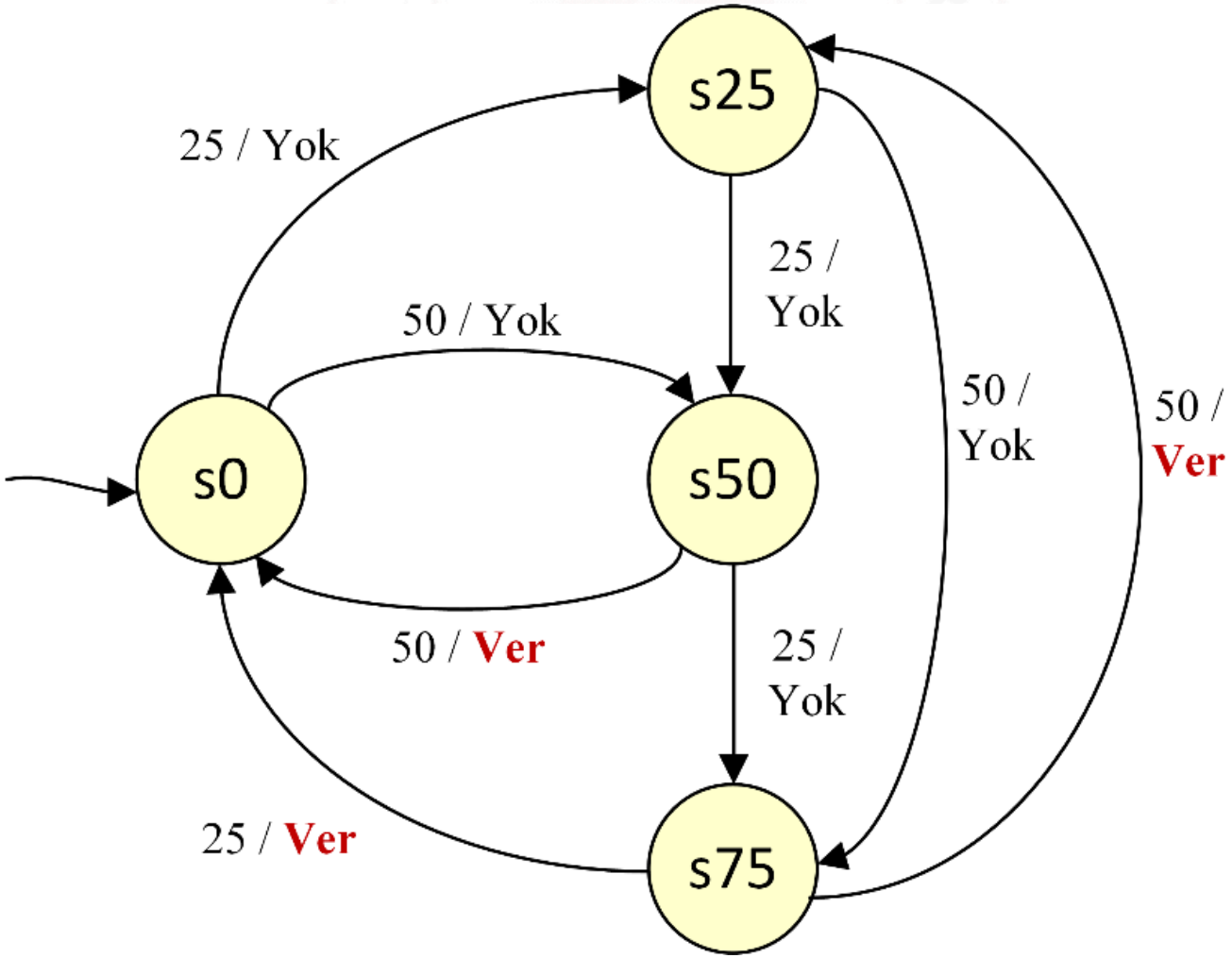
Şimdiki Durum	Geçiş Komutu	Yeni Durum
Başla	***y*** **– "00"**	Yıka
Başla	***d*** **– "01"**	Durula
Başla	***s*** **– "10"**	Sıkma
Yıka	***d*** **– "01"**	Durula
Durula	***s*** **– "10"**	Sıkma

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

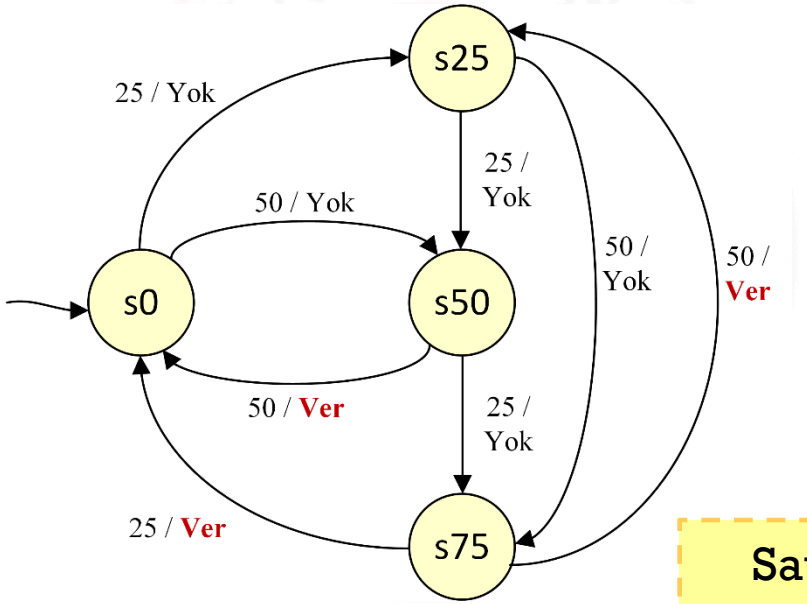


12.2 Moore ve Mealy Makineleri

- Şekildeki SDM yapısına dair gerekli VHDL kodunu Mealy SDM ile yazalım.



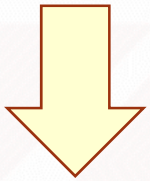
12.2 Moore ve Mealy Makineleri



Satış otomat kontrolü için durum-geçiş tablosu

Şimdiki Durum	Geçiş Komutu / Giriş	Yeni Durum	Çıkış
s0	***25kuruş*** **– '0'**	s25	0
s0	***50kuruş*** **– '1'**	s50	0
s25	***25kuruş*** **– '0'**	s50	0
s25	***50kuruş*** **– '1'**	s75	0
s50	***25kuruş*** **– '0'**	s75	0
s50	***50kuruş*** **– '1'**	s0	1
s75	***25kuruş*** **– '0'**	s0	1
s75	***50kuruş*** **– '1'**	s25	1

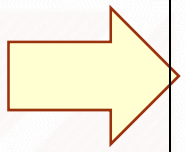
12.2 Moore ve Mealy Makineleri



```
Library ieee;
Use ieee.std_logic_1164.all;

Entity  satis_sdm is
Port (clk : in std_logic;
        giris : in std_logic;
        cikis : out std_logic);
End  satis_sdm;

Architecture davranis of satis_sdm is
Type durumlar is (s0, s25, s50, s75);
Signal simdiki_d: durumlar := s0;
Begin
```

12.2 Moore ve Mealy Makineleri

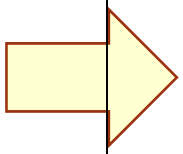
```
Process (clk)
Begin
   If  rising_edge(clk) then
      Case simdiki_d is
         When s0 =>
            If (giris='0') then
               cikis <= '0';
               simdiki_d <= s25;
            Elsif (giris='1') then
               cikis <= '0';
               simdiki_d <= s50;
            End if;
         When s25 =>
            If (giris='0') then
               cikis <= '0';
               simdiki_d <= s50;
            Elsif (giris='1') then
               cikis <= '0';
               simdiki_d <= s75;
            End if;
```



```
Process (clk)
Begin
   If  rising_edge(clk) then
      Case simdiki_d is  -- "25 Kuruş" için '0', "50 Kuruş" için '1'
         When s0 =>
            If (giris='0') then
               cikis <= '0';
               simdiki_d <= s25;
            Elsif (giris='1') then
               cikis <= '0';
               simdiki_d <= s50;
            End if;
         When s25 =>
            If (giris='0') then
               cikis <= '0';
               simdiki_d <= s50;
            Elsif (giris='1') then
               cikis <= '0';
               simdiki_d <= s75;
            End if;
         When s50 =>
            If (giris='0') then
               cikis <= '0';
               simdiki_d <= s75;
            Elsif (giris='1') then
               cikis <= '1';
               simdiki_d <= s0;
            End if;
         When s75 =>
            If (giris='0') then
               cikis <= '1';
               simdiki_d <= s0;
            Elsif (giris='1') then
               cikis <= '1';
               simdiki_d <= s25;
            End if;
      End case;
   End if;
End process;
End davranis;
```

12.2 Moore ve Mealy Makineleri

```
            When s50 =>
               If (giris='0') then
                  cikis <= '0';
                  simdiki_d <= s75;
               Elsif (giris='1') then
                  cikis <= '1';
                  simdiki_d <= s0;
               End if;
            When s75 =>
               If (giris='0') then
                  cikis <= '1';
                  simdiki_d <= s0;
               Elsif (giris='1') then
                  cikis <= '1';
                  simdiki_d <= s25;
               End if;
         End case;
      End if;
   End process;
End davranis;
```



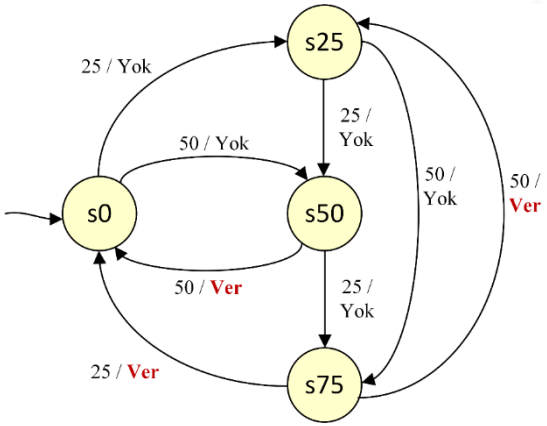
```
Process (clk)
Begin
   If  rising_edge(clk) then
      Case simdiki_d is  -- "25 Kuruş" için '0', "50 Kuruş" için '1'
         When s0 =>
            If (giris='0') then
               cikis <= '0';
               simdiki_d <= s25;
            Elsif (giris='1') then
               cikis <= '0';
               simdiki_d <= s50;
            End if;
         When s25 =>
            If (giris='0') then
               cikis <= '0';
               simdiki_d <= s50;
            Elsif (giris='1') then
               cikis <= '0';
               simdiki_d <= s75;
            End if;
         When s50 =>
            If (giris='0') then
               cikis <= '0';
               simdiki_d <= s75;
            Elsif (giris='1') then
               cikis <= '1';
               simdiki_d <= s0;
            End if;
         When s75 =>
            If (giris='0') then
               cikis <= '1';
               simdiki_d <= s0;
            Elsif (giris='1') then
               cikis <= '1';
               simdiki_d <= s25;
            End if;
      End case;
   End if;
End process;
End davranis;
```

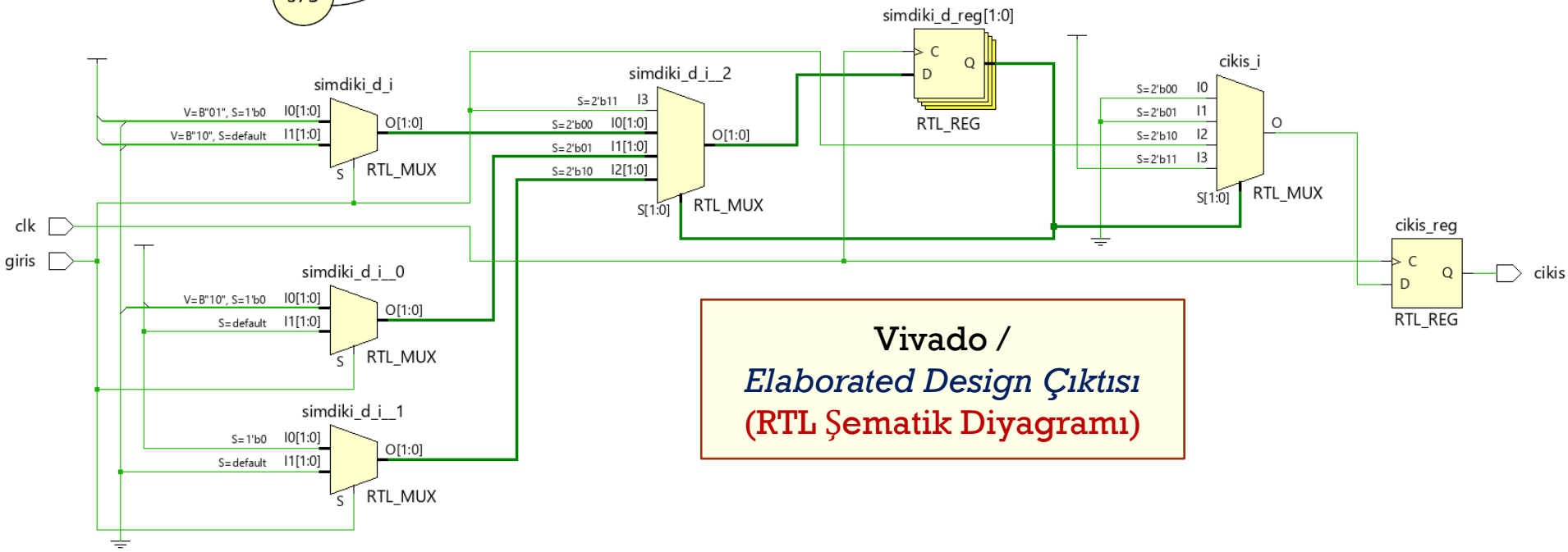
12.2 Moore ve Mealy Makineleri

Satış otomat kontrolü için durum-geçiş tablosu

Şimdiki Durum	Geçiş Komutu / Giriş	Yeni Durum	Çıkış
s0	***25kuruş – '0'***	s25	0
s0	***50kuruş – '1'***	s50	0
s25	***25kuruş – '0'***	s50	0
s25	***50kuruş – '1'***	s75	0
s50	***25kuruş – '0'***	s75	0
s50	***50kuruş – '1'***	s0	1
s75	***25kuruş – '0'***	s0	1
s75	***50kuruş – '1'***	s25	1

Mealy SDM





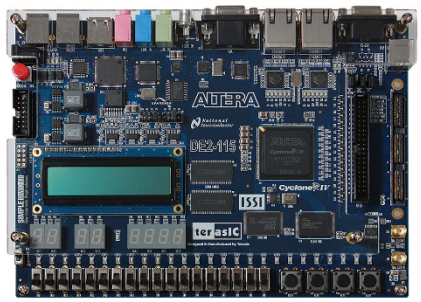
Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

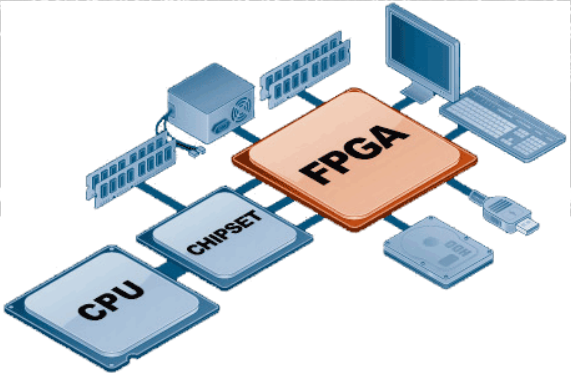
SONRAKİ DERS KONUSU

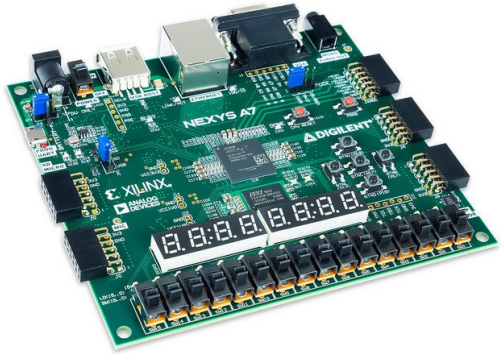
13- ÖRNEK UYGULAMALAR (EK)

İLERİ SAYISAL SİSTEMLER

13- ÖRNEK UYGULAMALAR (EK)







Ders sorumlusu:

Doç. Dr. Hasan KOYUNCU

13. ÖRNEK UYGULAMALAR (EK)

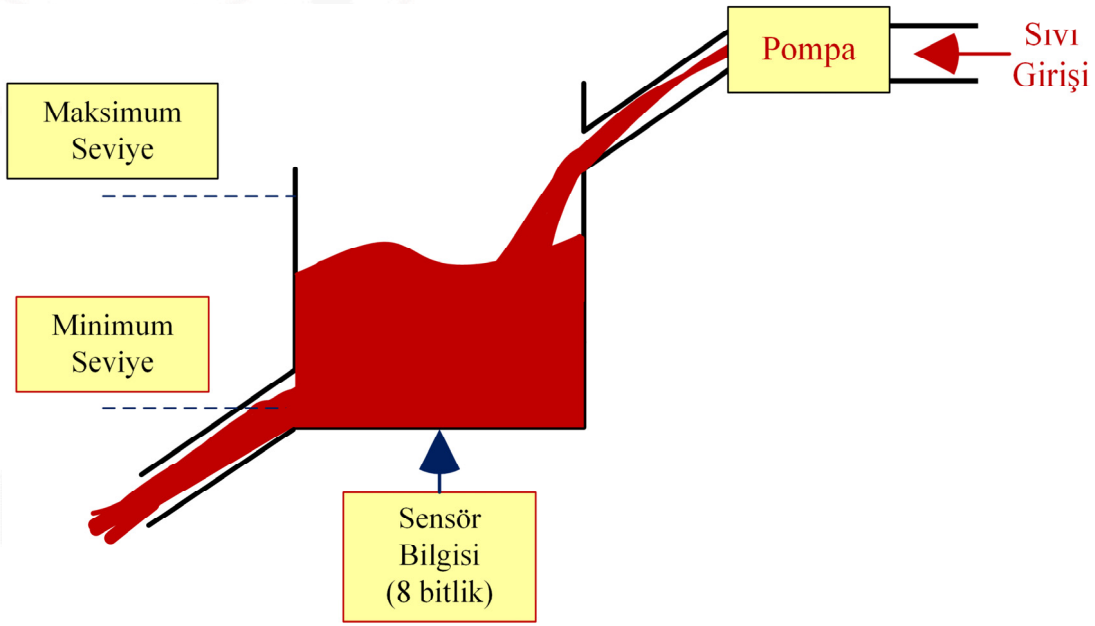
Örnek 13.1: Sıvı seviye kontrolü için;

- **8 bitlik girdi** isimli giriş port sinyali işlenerek, **pompa** isimli çıkış portuna pompanın **çalışması** için **'1'**, **durması** için **'0'** seviyeleri gönderilecektir.

- Sıvı seviyesinin, maksimum seviye (250) ve minimum seviye (20) arası değerler alabileceği bilinmektedir.

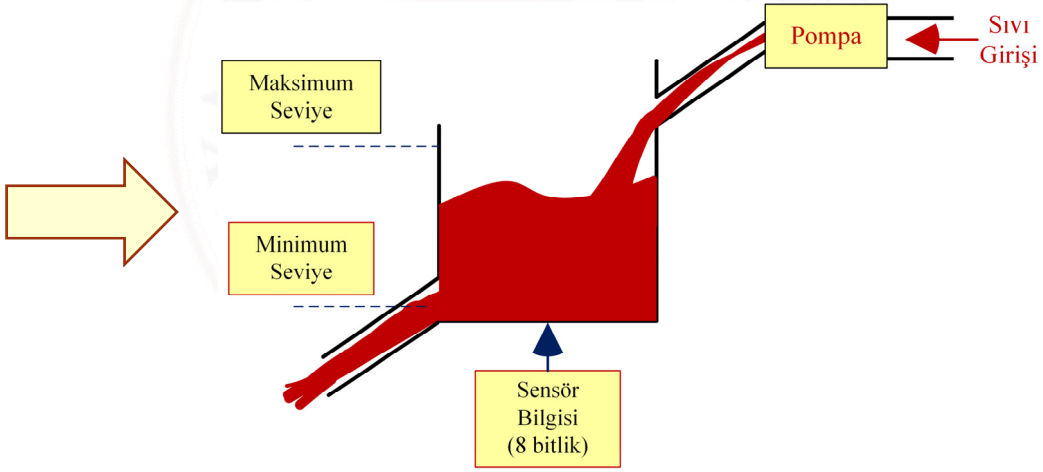
- **rst** giriş port sinyali **'1'** olduğunda pompa durdurulacak, **clk** giriş port sinyalinin ***yükselen kenarında*** bilgiler işlenerek pompaya gerekli çıkış iletilecektir.

Tasarım için gerekli VHDL kodunu yazınız.



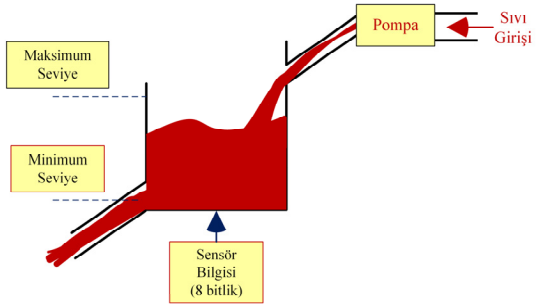
13. ÖRNEK UYGULAMALAR (EK)

Örnek 13.1: Sıvı seviye kontrolü için gerekli VHDL kodunu yazınız.




```
Library ieee;
Use ieee.std_logic_1164.all;

Entity seviye_kontrol is
Port (clk, rst : in std_logic;
        seviye : in std_logic_vector (7 downto 0);
        pompa: out std_logic);
End  seviye_kontrol;
```

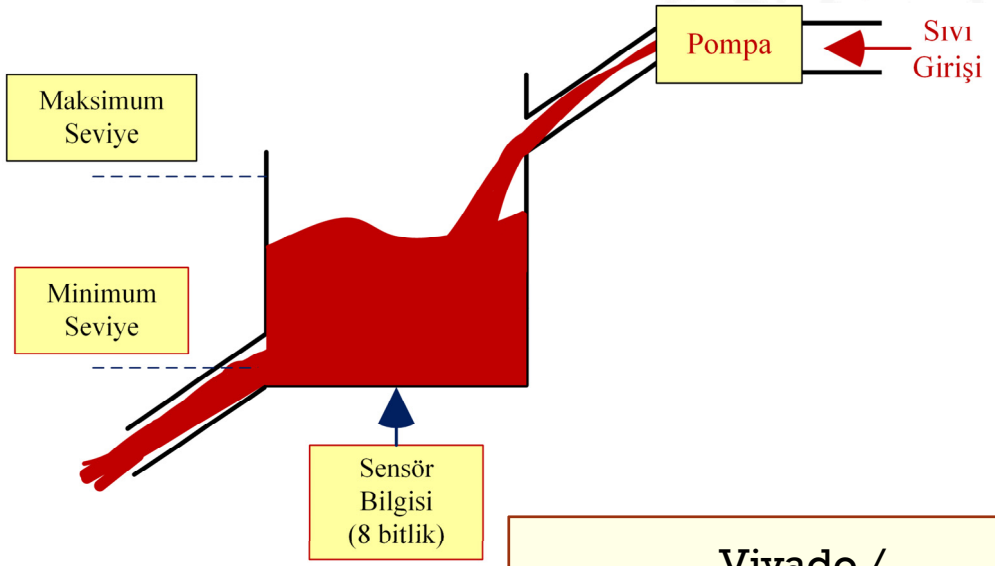
13. ÖRNEK UYGULAMALAR (EK)



Örnek 13.1: Sıvı seviye kontrolü için gerekli VHDL kodunu yazınız.

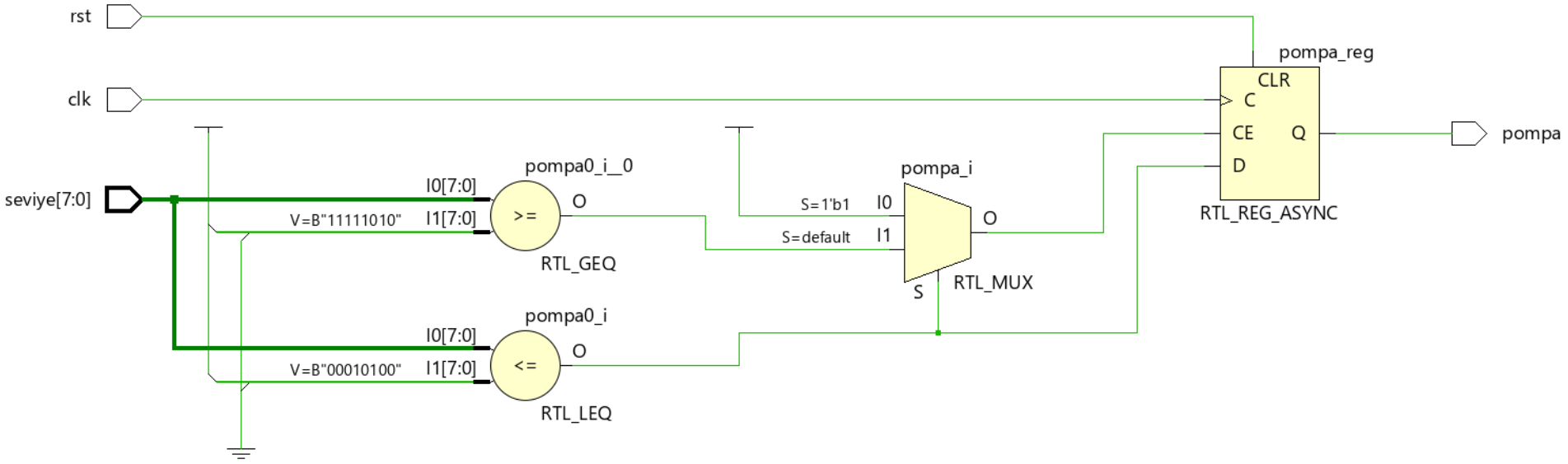
```
Architecture davranis of  seviye_kontrol is
Constant low_level:  std_logic_vector := "00010100"; --değer karşılığı 20
Constant high_level:  std_logic_vector := "11111010"; --değer karşılığı 250
Begin
   Process (rst, clk)
   Begin
      If rst='1' then
         pompa <= '0';
      Elsif rising_edge(clk) then
         If ( seviye <= low_level ) then
            pompa <= '1';
         Elsif (  seviye  >= high_level ) then
            pompa <= '0';
         End if;
      End if;
   End process;
End davranis;
```

13. ÖRNEK UYGULAMALAR (EK)

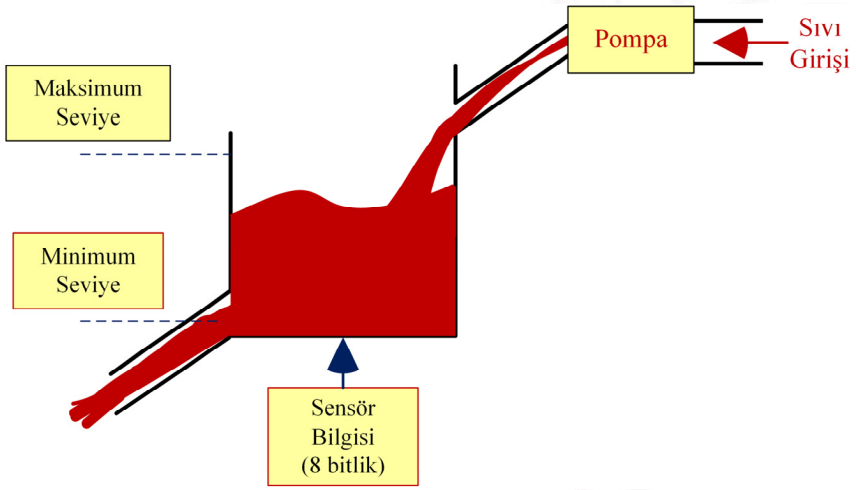



```
Architecture davranis of  seviye_kontrol is
Constant low_level:  std_logic_vector := "00010100";
Constant high_level:  std_logic_vector := "11111010";
Begin
   Process (rst, clk)
   Begin
      If rst='1' then
         pompa <= '0';
      Elsif rising_edge(clk) then
         If ( seviye <= low_level ) then
            pompa <= '1';
         Elsif (  seviye  >= high_level ) then
            pompa <= '0';
         End if;
      End if;
   End process;
End davranis;
```

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

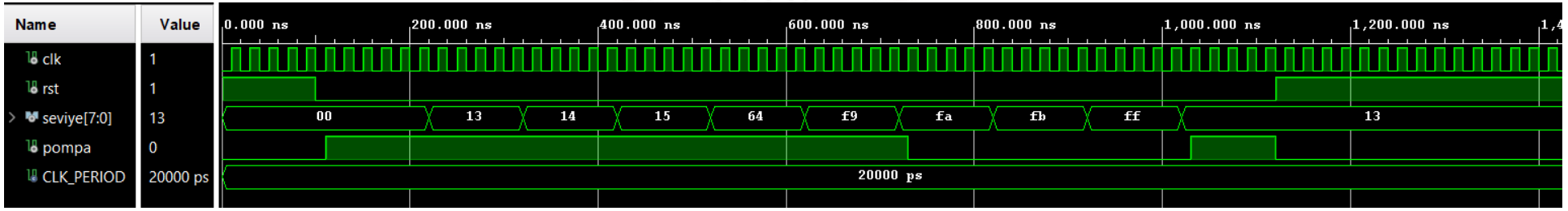


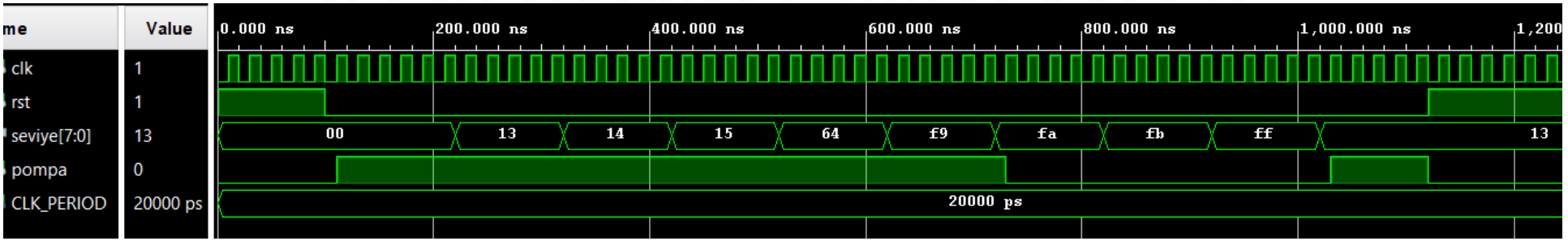
13. ÖRNEK UYGULAMALAR (EK)




```
Library ieee;
Use ieee.std_logic_1164.all;

Entity seviye_kontrol is
Port (clk, rst : in std_logic;
      seviye : in std_logic_vector (7 downto 0);
      pompa: out std_logic);
End  seviye_kontrol;
```

Vivado / *Simulation Çıktısı*





13. ÖRNEK UYGULAMALAR (EK)

Örnek 13.2: Bir otoparkta boş park yeri sayısını belirtmek için;

- **1'er bitlik giris_sen** ve **cikis_sen** isimli sensör girişleri **(giriş port sinyalleri)** sırasıyla giren araç ve çıkan araç için işlenerek, **yer_bilgisi** isimli çıkış portuna otopark içi araç sayısı **(ikili)** aktarılacaktır.

- **rst** giriş port sinyali **'1'** olduğunda araç sayı bilgisi birlenecek (otopark tamamen boş, 255 boş yer var), **clk** giriş port sinyalinin ***yükselen kenarında*** bilgiler işlenerek gerekli sayı çıkışı üretilecektir.

Tasarım için gerekli VHDL kodunu yazınız.

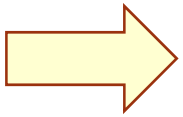
13. ÖRNEK UYGULAMALAR (EK)

Örnek 13.2: Park yeri kontrolü için gerekli VHDL kodunu yazınız.


```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity park_yeri is
Port (clk, rst, giris_sen, cikis_sen : in std_logic;
      yer_bilgisi: out std_logic_vector (7 downto 0));
End park_yeri;
```

13. ÖRNEK UYGULAMALAR (EK)

Örnek 13.2: Park yeri kontrolü için gerekli VHDL kodunu yazınız.



```
Architecture davranis of park_yeri is
Signal yer_sayici:  unsigned (7 downto 0) := (others =>'1');
Signal giris_vek, cikis_vek:  unsigned (3 downto 0) := (others =>'0');
Constant max_deg: unsigned (7 downto 0) := (others =>'1');  -- 255 araç için
Begin
   Process (rst, clk)
   Begin
      If rst='1' then
         yer_sayici <=  (others =>'1');
      Elsif rising_edge(clk) then
         giris_vek <= giris_vek(2 downto 0) & giris_sen;
         cikis_vek <= cikis_vek(2 downto 0) & cikis_sen;
         -- Araç hem girer hem de çıkarsa boş yer sayısı değişmez
         If giris_vek = "1100" and cikis_vek = "1100" then
            yer_sayici <=  yer_sayici;
         -- Araç girerse boş yer azalır
         Elsif giris_vek = "1100"  and yer_sayici > "00000000" then
            yer_sayici <=  yer_sayici - 1;
         -- Araç çıkarsa boş yer artar
         Elsif cikis_vek = "1100" and yer_sayici < max_deg then
            yer_sayici <= yer_sayici + 1;
         End if;
      End if;
   End process;
   yer_bilgisi <= std_logic_vector (yer_sayici);
End davranis;
```

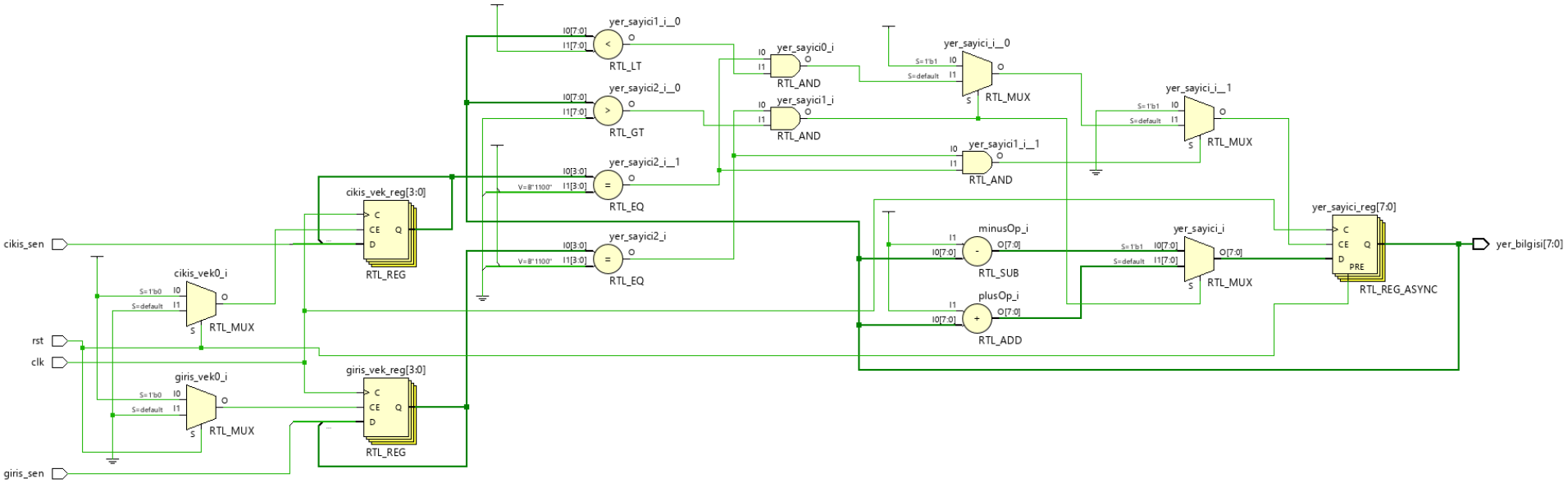
Aynı

to_unsigned(0,8)

13. ÖRNEK UYGULAMALAR (EK)


```
Architecture davranis of park_yeri is
Signal yer_sayici:  unsigned (7 downto 0) := (others =>'1');
Signal giris_vek, cikis_vek:  unsigned (3 downto 0) := (others =>'0');
Constant max_deg: unsigned (7 downto 0) := (others =>'1');  -- 255 araç için
Begin
   Process (rst, clk)
   Begin
      If rst='1' then
          yer_sayici <=  (others =>'1');
      Elsif rising_edge(clk) then
         giris_vek <= giris_vek(2 downto 0) & giris_sen;
         cikis_vek <= cikis_vek(2 downto 0) & cikis_sen;
         -- Araç hem girer hem de çıkarsa boş yer sayısı değişmez
         If giris_vek = "1100" and cikis_vek = "1100" then
            yer_sayici <=  yer_sayici;
         -- Araç girerse boş yer azalır
         Elsif giris_vek = "1100"  and yer_sayici > "00000000" then
            yer_sayici <=  yer_sayici - 1;
         -- Araç çıkarsa boş yer artar
         Elsif cikis_vek = "1100" and yer_sayici < max_deg then
            yer_sayici <= yer_sayici + 1;
         End if;
      End if;
   End process;
   yer_bilgisi <= std_logic_vector (yer_sayici);
End davranis;
```

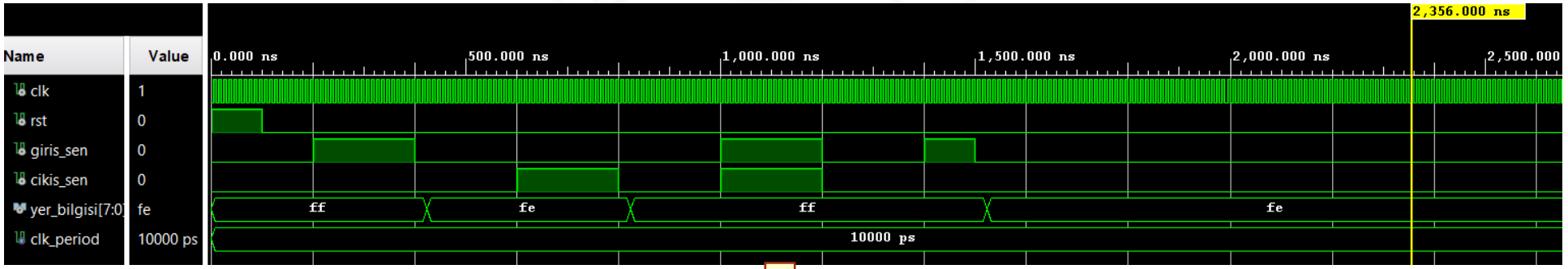
Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

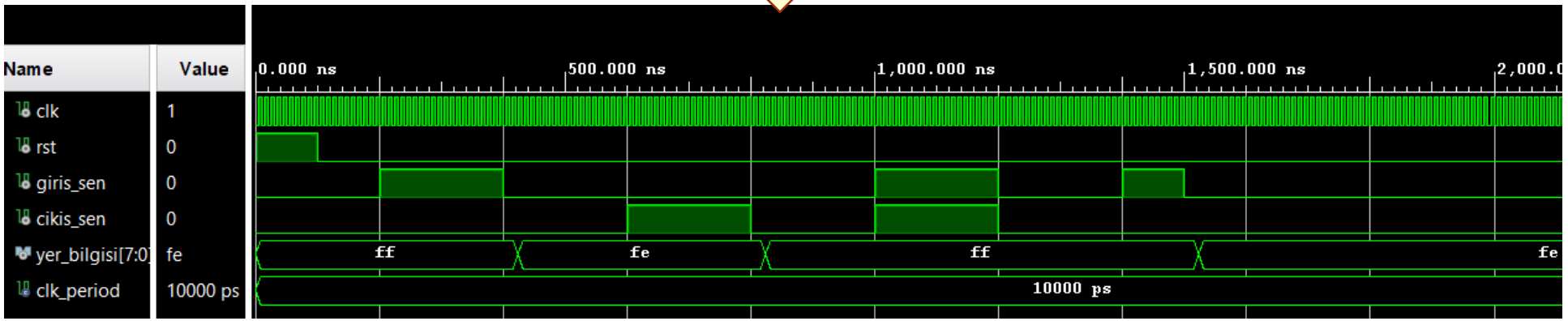


13. ÖRNEK UYGULAMALAR (EK)


```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity park_yeri is
Port (clk, rst, giris_sen, cikis_sen : in std_logic;
      yer_bilgisi: out std_logic_vector (7 downto 0));
End park_yeri;
```

Vivado / *Simulation Çıktısı*





13. ÖRNEK UYGULAMALAR (EK)

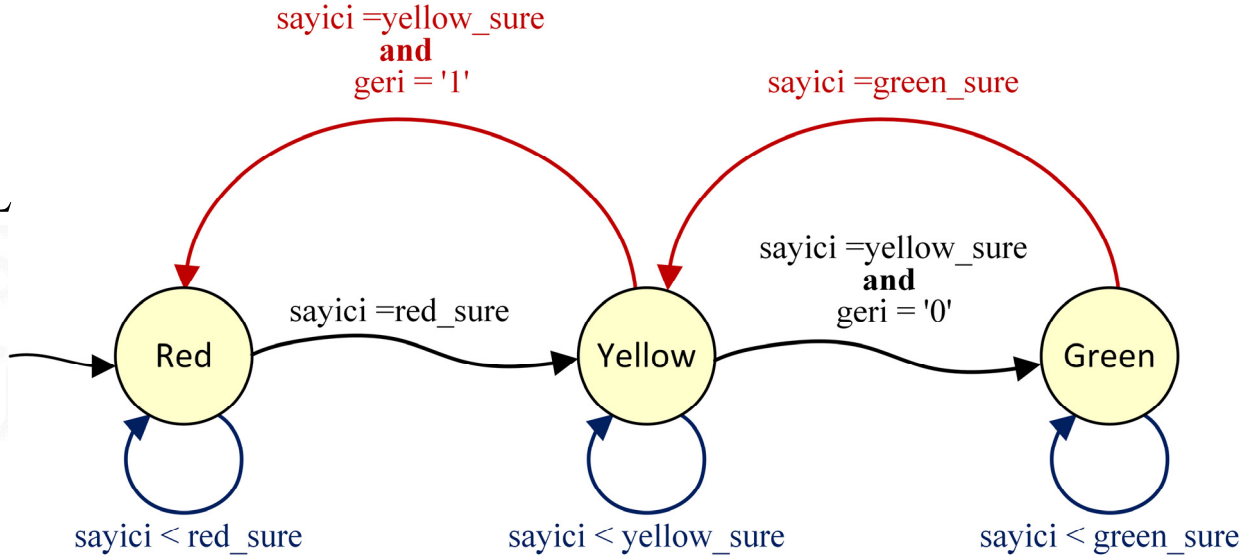
Örnek 13.3: Bir trafik lambasındaki renk değişimi için;

- 30 saniye **kırmızı**, 5 saniye **sarı** ve 30 saniye **yeşil** yanması sağlanacaktır.

- **rst** giriş port sinyali **'1'** olduğunda ışık kırmızıya dönmelidir.

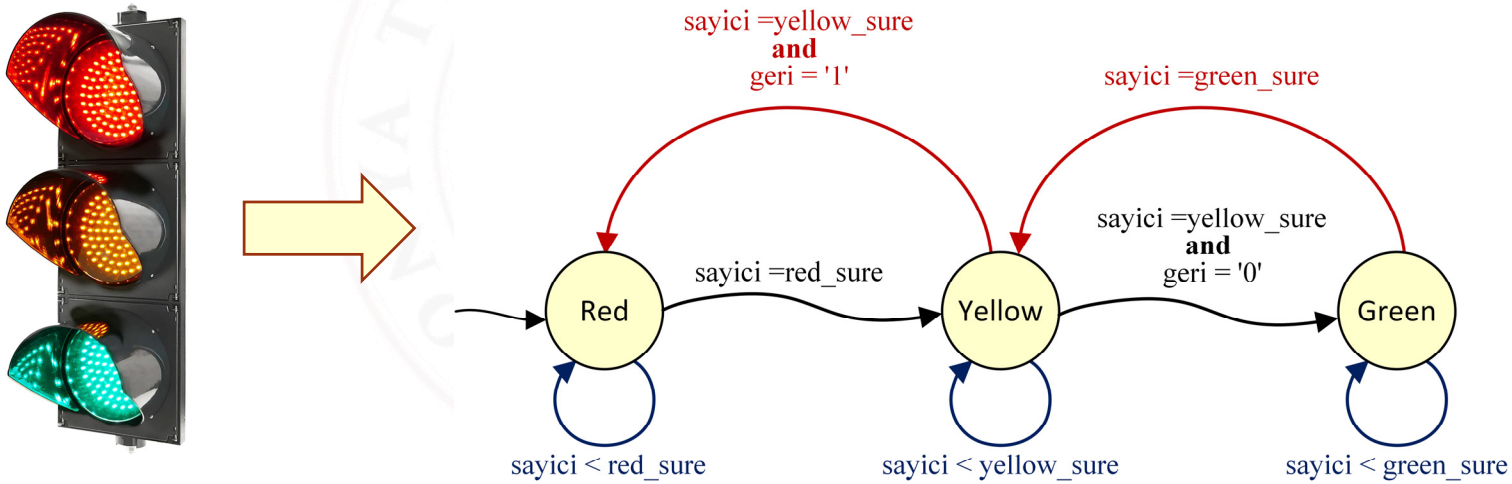
- **clk** giriş port sinyalinin ***yükselen kenarında*** ise gerekli işlemlerin yapılarak **red**, **yellow** ve **green** isimli **1'er bitlik** çıkış portlarına gerekli yetkilendirme sağlanacaktır.

Tasarım için gerekli VHDL kodunu yazınız.



13. ÖRNEK UYGULAMALAR (EK)

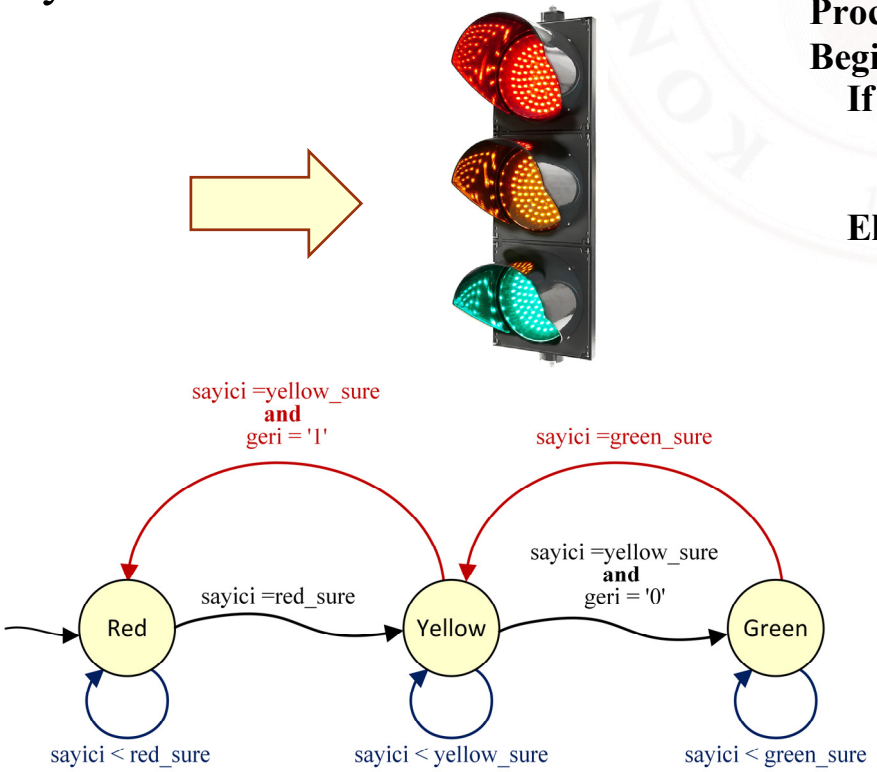
Örnek 13.3: Bir trafik lambasındaki renk değişimi için gerekli VHDL kodunu yazınız.




```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity  trafik_kontrol is
Port (clk, rst : in std_logic;
        red,  yellow, green: out std_logic);
End trafik_kontrol;
```

13. ÖRNEK UYGULAMALAR (EK)

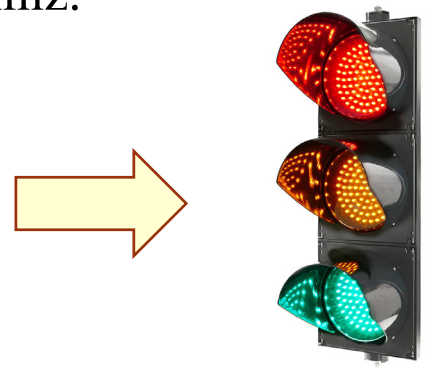
Örnek 13.3: Bir trafik lambasındaki renk değişimi için gerekli VHDL kodunu yazınız.

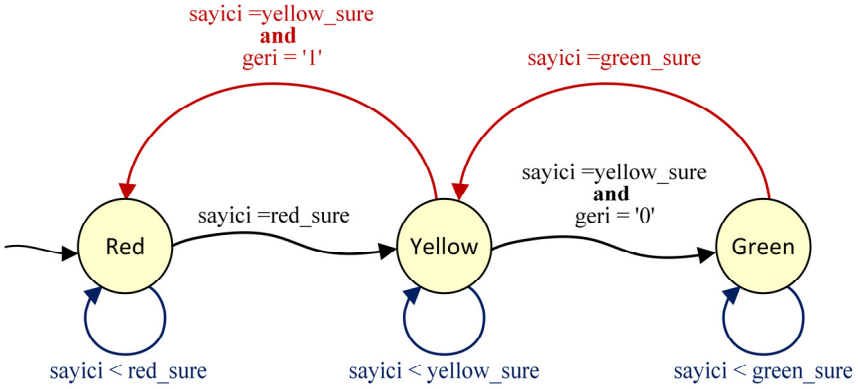



```
Architecture davranis of trafik_kontrol is
Type durumlar is (red_state, yellow_state, green_state);
Signal simdiki_d, gelecek_d: durumlar;
Signal sayici: integer range 0 to 31:= 0;
Signal geri: std_logic := '0';
-- süreler clk periyodu 1sn için ayarlı
Constant red_sure: integer:= 30;
Constant yellow_sure: integer:= 5;
Constant green_sure: integer:= 30;
Begin

   Process (rst, clk)
   Begin
      If  rst='1' then
         sayici <= 0;
         simdiki_d <= red_state;
      Elsif rising_edge(clk) then
         sayici <= sayici + 1;
         simdiki_d <= gelecek_d; -- gelecek durum diğer processten geliyor
         -- Durum geçişlerinde sayaç sıfırlama
         If (simdiki_d = red_state and sayici = red_sure) or
            (simdiki_d = yellow_state and sayici = yellow_sure) or
            (simdiki_d = green_state and sayici = green_sure) then
            sayici <= 0;
         End if;
         -- Geri sinyali güncelleme
         If (simdiki_d = green_state and sayici = green_sure) then
            geri <= '1';
         Elsif (simdiki_d = red_state and sayici = red_sure) then
            geri <= '0';
         End if;
      End if;
   End process;
```

13. ÖRNEK UYGULAMALAR (EK)

Örnek 13.3: Bir trafik lambasındaki renk değişimi için gerekli VHDL kodunu yazınız.



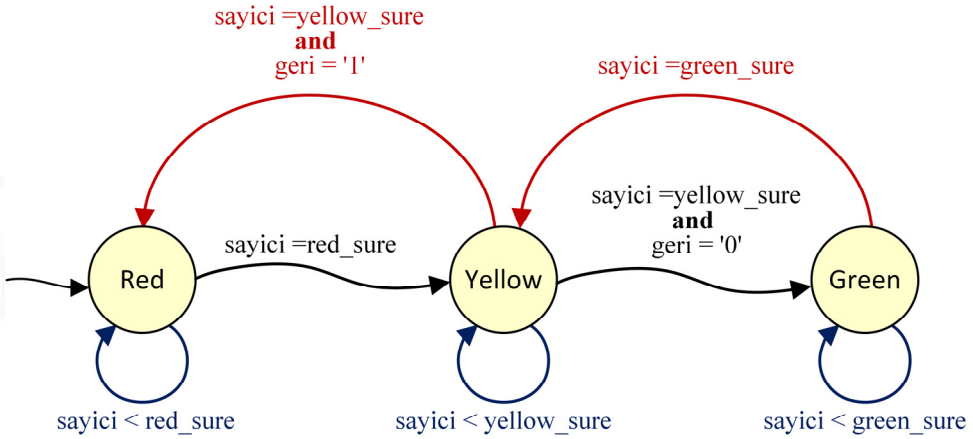



```
Process (simdiki_d, sayici)  -- ürettiği çıktı gelecek_d ve gerekli renk
Begin
      red <= '0'; yellow <= '0'; green <= '0';
      Case simdiki_d is
-- ////////////////////////////////////////////////////////// --
         When red_state =>
            red <= '1';
            If (sayici = red_sure) then
               gelecek_d <= yellow_state;
            Else
               gelecek_d <= red_state;
            End if;
-- ////////////////////////////////////////////////////////// --
         When yellow_state =>
            yellow <= '1';
            If (sayici = yellow_sure) then
               If (geri='0') then
                  gelecek_d <= green_state;
               Else
                  gelecek_d <= red_state;
               End if;
            Else
               gelecek_d <= yellow_state;
            End if;
-- ////////////////////////////////////////////////////////// --
         When green_state =>
            green <= '1';
            If (sayici = green_sure) then
               gelecek_d <= yellow_state;
            Else
               gelecek_d <= green_state;
            End if;
      End case;
End process;
End davranis;
```

13. ÖRNEK UYGULAMALAR (EK)

Örnek 13.3: Bir trafik lambasındaki renk değişimi için gerekli VHDL kodunu yazınız.

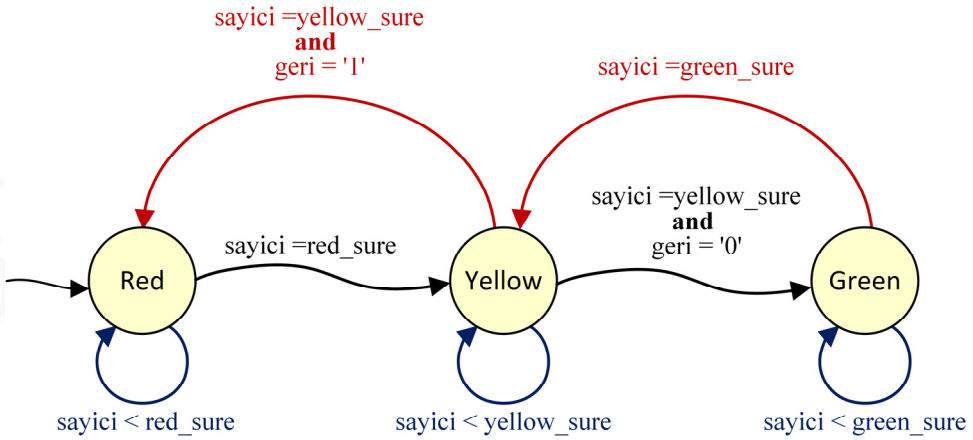
```
Process (simdiki_d, sayici)  -- ürettiği çıktı gelecek_d ve gerekli renk
Begin
        red <= '0'; yellow <= '0'; green <= '0';
        Case simdiki_d is
-- ////////////////////////////////////////////////////////////////////// --
            When red_state =>
              red <= '1';
              If (sayici = red_sure) then
                 gelecek_d <= yellow_state;
              Else
                 gelecek_d <= red_state;
              End if;
```



13. ÖRNEK UYGULAMALAR (EK)

Örnek 13.3: Bir trafik lambasındaki renk değişimi için gerekli VHDL kodunu yazınız.

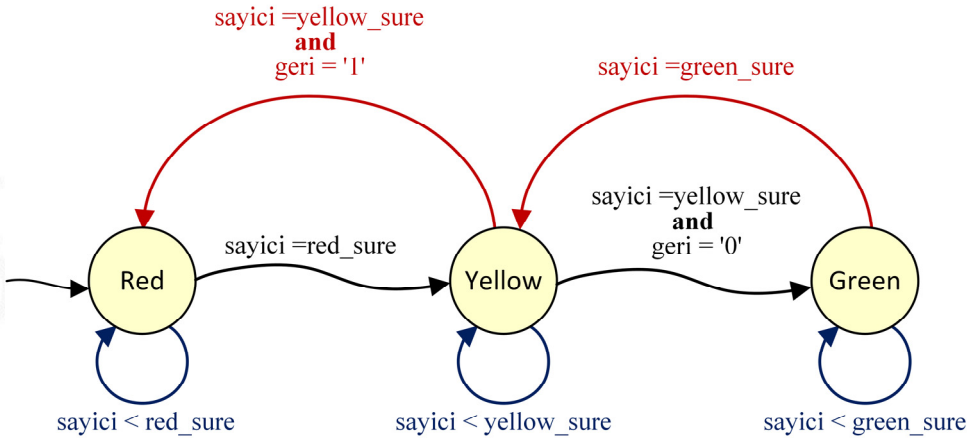
```
-- ////////////////////////////////////////////////////////////// --
        When yellow_state =>
            yellow <= '1';
            If (sayici = yellow_sure) then
                If (geri='0') then
                    gelecek_d <= green_state;
                Else
                    gelecek_d <= red_state;
                End if;
            Else
                gelecek_d <= yellow_state;
            End if;
```



13. ÖRNEK UYGULAMALAR (EK)

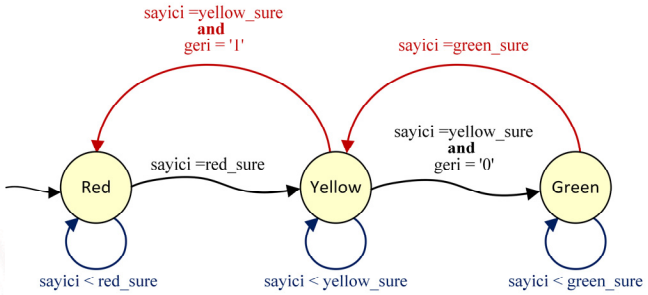
Örnek 13.3: Bir trafik lambasındaki renk değişimi için gerekli VHDL kodunu yazınız.

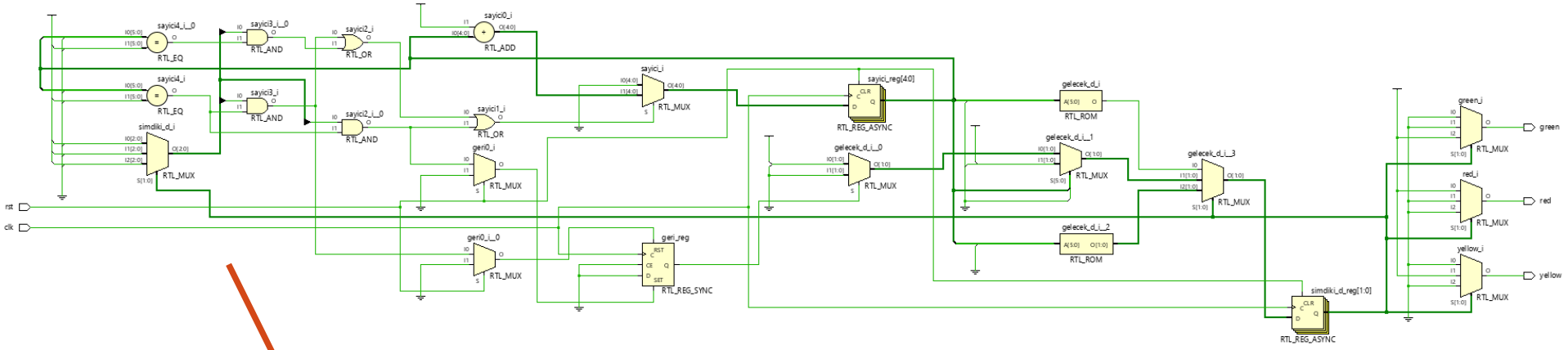
```
-- ////////////////////////////////////////////////////////////////// --
            When green_state =>
                green <= '1';
                If (sayici = green_sure) then
                    gelecek_d <= yellow_state;
                Else
                    gelecek_d <= green_state;
                End if;
            End case;
        End process;
    End davranis;
```

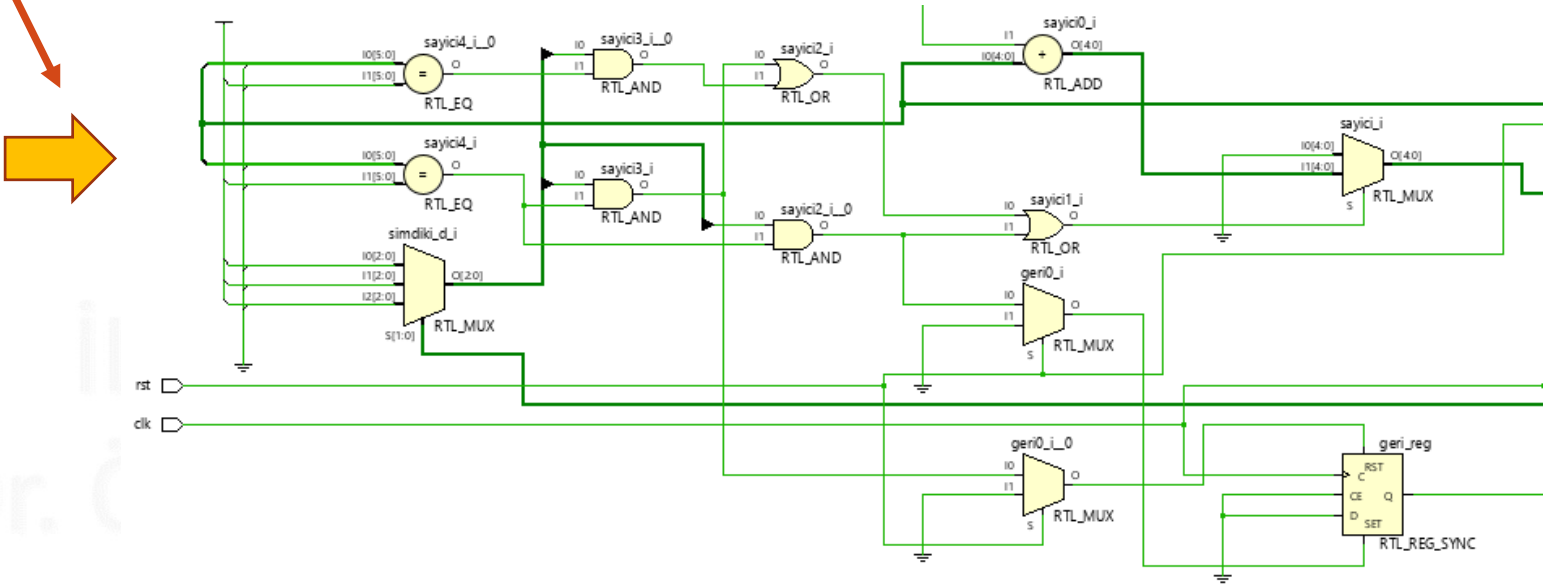


13. ÖRNEK UYGULAMALAR (EK)

Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)

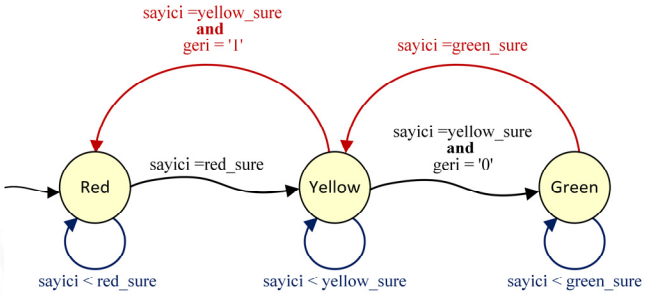


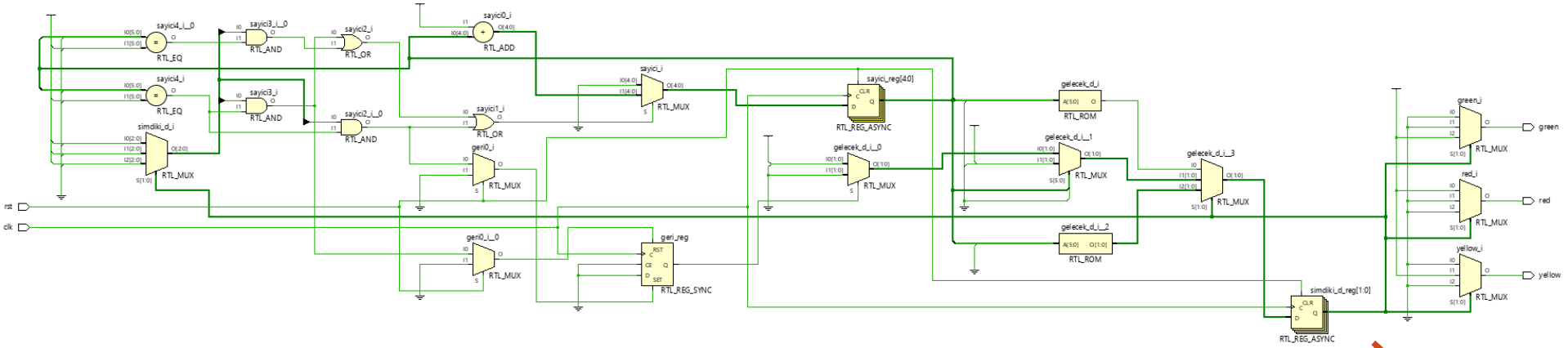


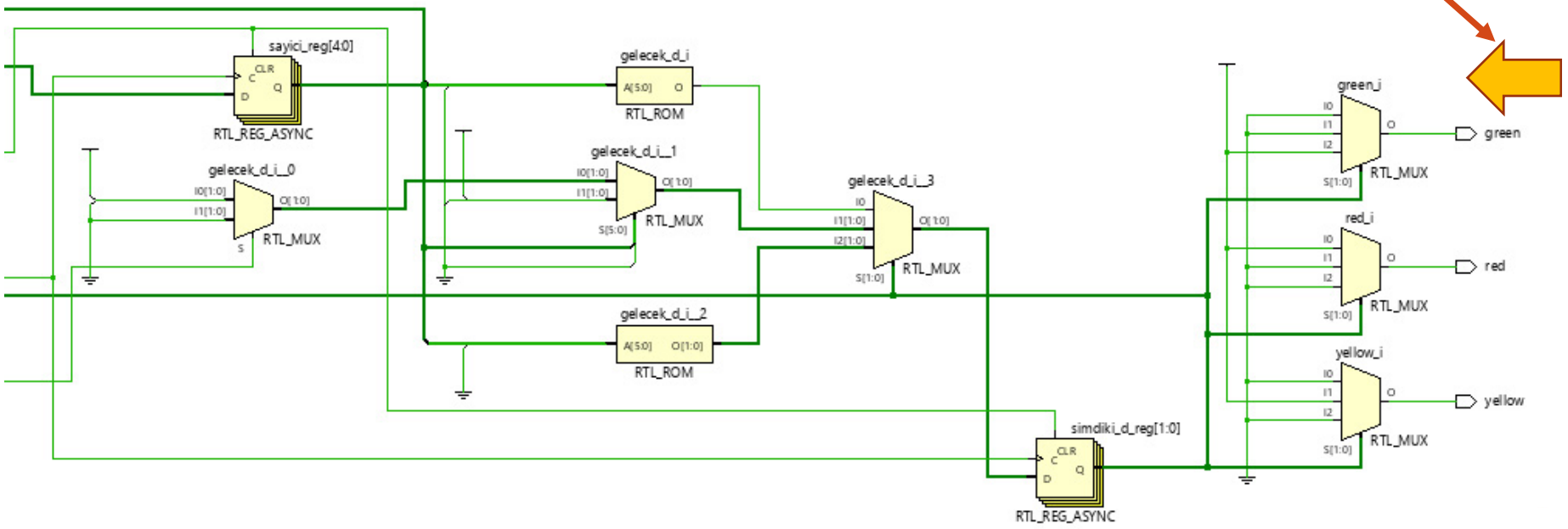


13. ÖRNEK UYGULAMALAR (EK)

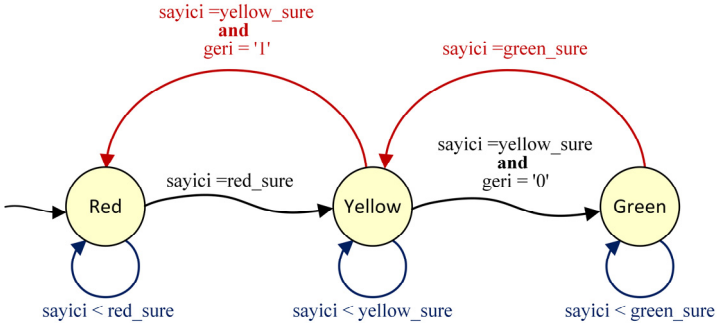
Vivado /
Elaborated Design Çıktısı
(RTL Şematik Diyagramı)





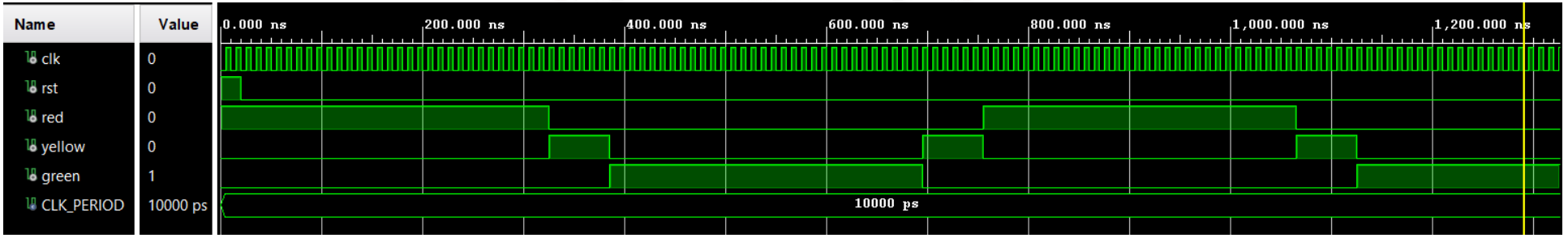


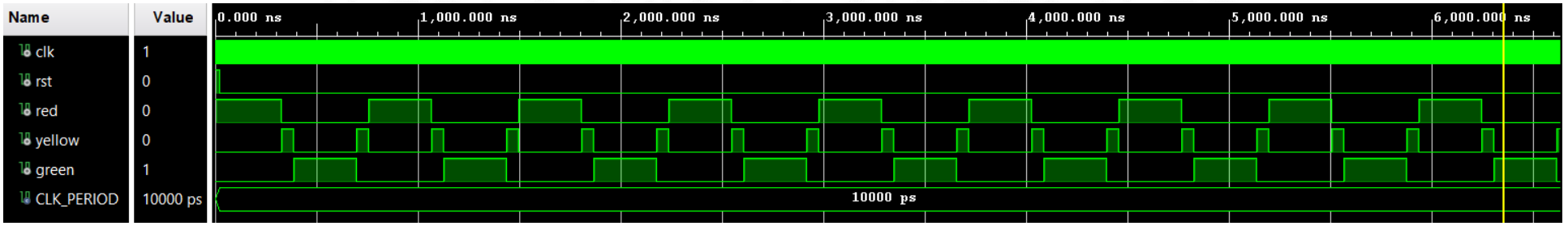
13. ÖRNEK UYGULAMALAR (EK)



```
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity  trafik_kontrol is
Port (clk, rst : in std_logic;
        red,  yellow, green: out std_logic);
End trafik_kontrol;
```

Vivado /
Simulation Çıktıları





FINISH

ESİNLENİLEN VE TAVSİYE EDİLEN KAYNAKLAR

- Digital Fundamentals, 2015 (Thomas L. Floyd, 11. Baskı, Global Edition)
- Her Yönüyle FPGA ve VHDL, 2015, (Engin Sarıtaş - Sedat Karataş, 3. Baskı, Palme Yayınları)
- Donanım Tanımlama Dili VHDL ve FPGA Uygulamaları, 2017, (Dr. İbrahim Savran, 1. Baskı, Papatya Bilim Yayınları)
- FPGA Prototyping by VHDL Examples: Xilinx Microblaze MCS Soc, 2017, (Pong P Chu, 2. Baskı, Wiley Press)
- Digital Design: With and Introduction to the Verilog HDL, VHDL, and System Verilog, 2018 (M. Morris Mano & Michael D. Ciletti, 6. Baskı, Global Edition)
- VHDL ve Verilog ile Sayısal Tasarım, 2019, (Doç. Dr. Burak Kelleci, 2. Baskı, Seçkin Yayınları)
- FPGA ile Gömülü Sistemler ve Sayısal Devre Tasarımı, 2020, (Dr. Serkan Dereli, Nobel Yayınları)
- Digital Design and Computer Architecture, RISC-V Edition, 2021 (Sarah Harris & David Harris, 1. Baskı, Morgan Kaufm)
- Getting Started with FPGAs: Digital Circuit Design, Verilog, and VHDL for Beginners, 2023, (Russell Merrick, No Starch Press)
- Fundamentals of VHDL for FPGA Programming Using Vivado, 2025, (Majid Pakdel, Wiley-IEEE Press)