FPGA REHBERİ

FPGA Temelleri

FPGA ve VHDL'in temellerini sıfırdan öğrenmek için hazırlanmış kapsamlı bir rehber

Ertan Suluağaç

Şubat 2026

İçindekiler

01 **FPGA ve VHDL Temelleri**
FPGA nedir, CPU/MCU farkı, yapı taşları, VHDL'e giriş

02 **HDL Nedir ve Nasıl Çalışır?**
Sematik tasarımdan HDL'lere geçiş, VHDL ve Verilog'un doğuşu

03 **VHDL Dili**
Kütüphane, paket, veri tipleri, entity ve architecture

04 **FPGA Tasarım Akışı**
Vivado ile tasarım, sentez ve uygulama adımları

05 **Kombinasyonel Mantık Tasarımı**
Kapı seviyesi, hiyerarşik tasarım ve eş zamanlı atamalar

06 **Ardışıl Mantık Tasarımı**
Flip-Flop, reset, pipeline ve senkronizasyon

07 **Block RAM (BRAM) Tasarımı**
BRAM yapısı, tek/çift port, okuma/yazma modları

08 **Durum Makinesi (FSM) Tasarımı**
Sonlu durum makinesi, kodlama ve geçişler

09 **Sık Karşılaşılan Sorular ve Kaynaklar**
Temel sorular, cevaplar ve referanslar

Bölüm 1

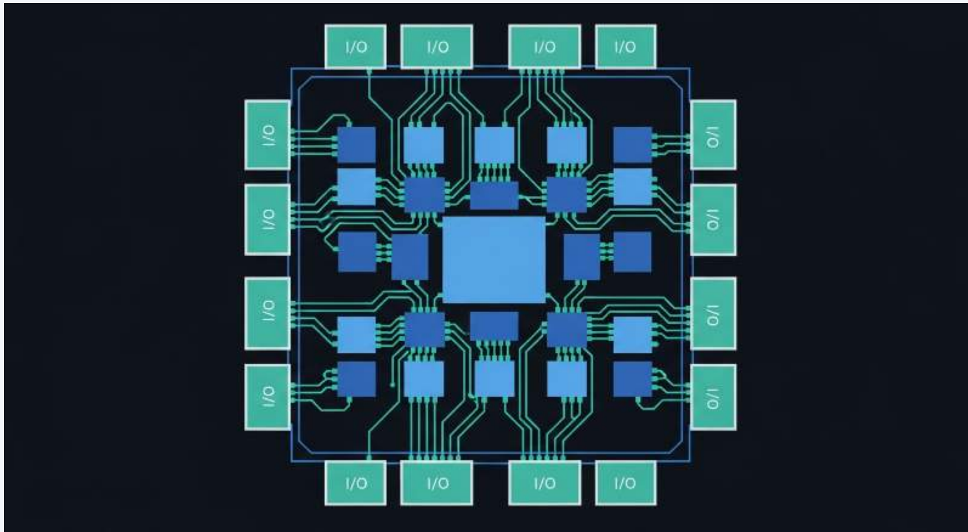
FPGA ve VHDL Temelleri

FPGA nedir, CPU/MCU farkı, yapı taşları, VHDL'e giriş

FPGA Nedir?

FPGA (Field Programmable Gate Array), üretimden sonra kullanıcı tarafından programlanabilen bir entegre devre türüdür. Türkçe karşılığı 'Alan Programlanabilir Kapı Dizisi' olarak ifade edilebilir.

En basit haliyle: FPGA, içinde binlerce küçük mantık blogu bulunan bir çiptir. Bu blokların bağlantılarını değiştirerek istediğiniz dijital devreyi oluşturabilirsiniz. Yani donanımı yazılım gibi programlıyorsunuz.



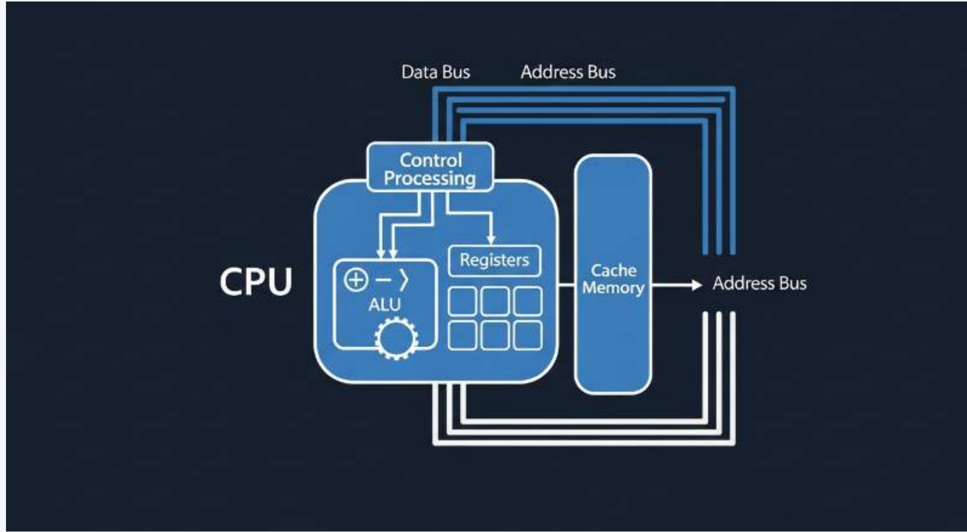
FPGA'nın iç yapısı: Mantık blokları, bağlantı yolları ve giriş/çıkış blokları

İPUCU

FPGA'yi bir LEGO seti gibi düşünebilirsiniz. Parçalar (mantık blokları) hazır, siz bunları istediğiniz şekilde birleştirerek kendi özel donanımınızı oluşturuyorsunuz.

CPU (Merkezi İşlem Birimi)

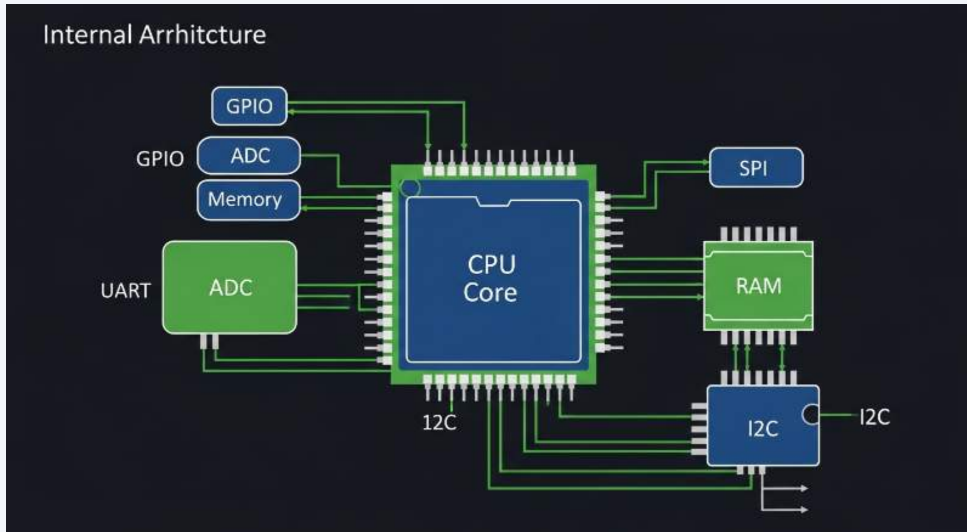
CPU (Central Processing Unit), bir bilgisayarın beynidir. İçinde ALU (aritmetik mantık birimi), yazmaçlar (register), kontrol birimi ve ön bellek (cache) bulunur. CPU, yazılım komutlarını tek tek sırayla çalıştırır.



CPU iç yapısı: ALU, yazmaçlar, kontrol birimi ve veri yolları

MCU (Mikrodenetleyici)

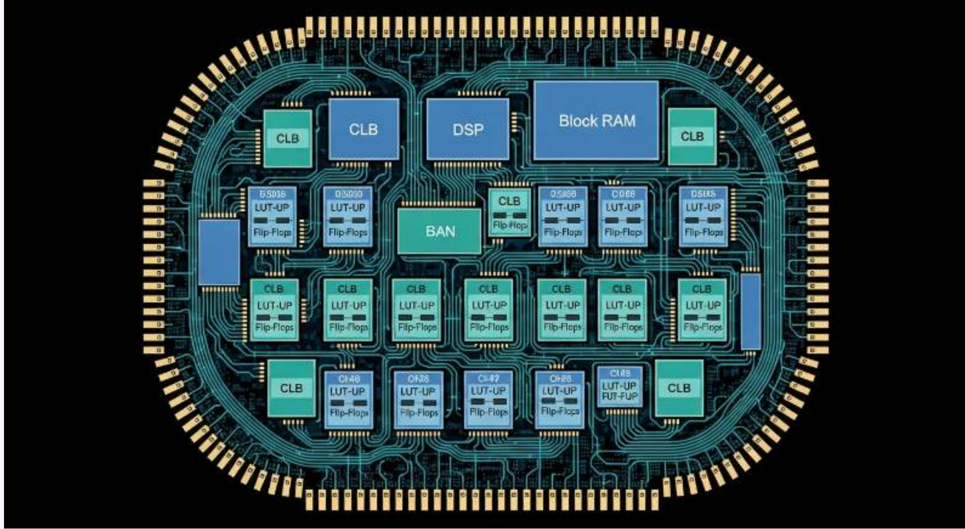
MCU (Microcontroller Unit), bir CPU çekirdeğinin yanına bellek (Flash, RAM), GPIO, ADC, UART, SPI, I2C gibi çevre birimlerini tek bir çip üzerine yerleştirmiştir. Gömülü sistemlerde (embedded systems) yaygın kullanılır.



MCU iç yapısı: CPU çekirdeği + bellek + çevre birimleri tek çipte

FPGA'nın İç Yapısı

FPGA ise tamamen farklı bir yapıyadır. İçinde sabit bir işlemci yoktur. Bunun yerine programlanabilir mantık blokları, bağlantı yolları ve giriş/çıkış birimleri vardır.

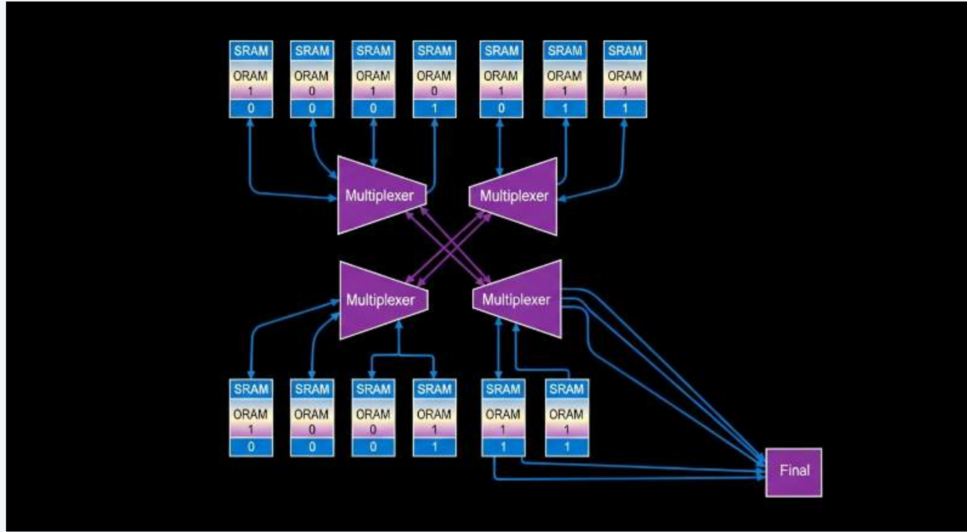


FPGA iç yapısı: CLB dizisi, Block RAM, DSP dilimi, IOB ve bağlantı yolları

- CLB (Karmaşık Mantık Blogu): LUT + FF içerir. Mantığın temel işlem birimi.
- LUT (Arama Tablosu): Gerçeklik tablosu gibi çalışır. N girişli LUT, 2^N SRAM hücresi içerir.
- FF (Flip-Flop): Tek bit veri saklayan bellek elemanı.
- Block RAM: Yerleşik bellek blokları, genellikle birkaç kB kapasitede.
- DSP Dilimi: Çarpma gibi aritmetik işlemleri hızlı yapan özel birimler.
- IOB (Giriş/Çıkış Blokları): Dış dünyayla iletişim. LVDS, LVCMOS, SSTL destekler.
- Bağlantı Yollari: CLB'ler arasi programlanabilir kablolama. FPGA alanının %60-70'i.

LUT Nasıl Çalışır?

LUT (Look-Up Table), FPGA'daki mantığın temel yapı taşıdır. Tüm olası giriş kombinasyonlari için çıkış değerlerini saklayarak kombinasyonel mantık fonksiyonlarını gerçekleştirir. İçinde küçük bir SRAM belleği vardır.



LUT yapısı: N girişli bir LUT, 2^N adet SRAM hücresi içerir. Örneğin LUT4, 16 SRAM hücresi gerektirir

Neden FPGA Kullanırız?

- Yeniden Yapılandırma: Donanım davranışı fiziksel değişiklik olmadan değiştirilebilir.
- Paralel Mimari: Birçok görevi donanım seviyesinde eş zamanlı yürütebilir.
- Geniş G/C Destegi: LVDS, LVCMOS, SSTL, HSTL gibi çeşitli standartları destekler.
- Özel Mantık: Sinyal işleme, protokol yönetimi gibi ihtiyaçlara özel tasarım.
- Belirli Zamanlama: Isletim sistemi yoktur, hassas zamanlama sağlar (radar, motor kontrol).

Karşılaştırma Tablosu

Özellik	CPU	MCU	FPGA
İşlem Modeli	Sırasal	Sırasal	Paralel (eş zama..
Programlama	C, Python	C/C++	VHDL / Verilog
Hiz	GHz (sirasal)	MHz	Çok yüksek (para..
Kullanım	Bilgisayar	IoT, gömülü	Sinyal işleme, t..

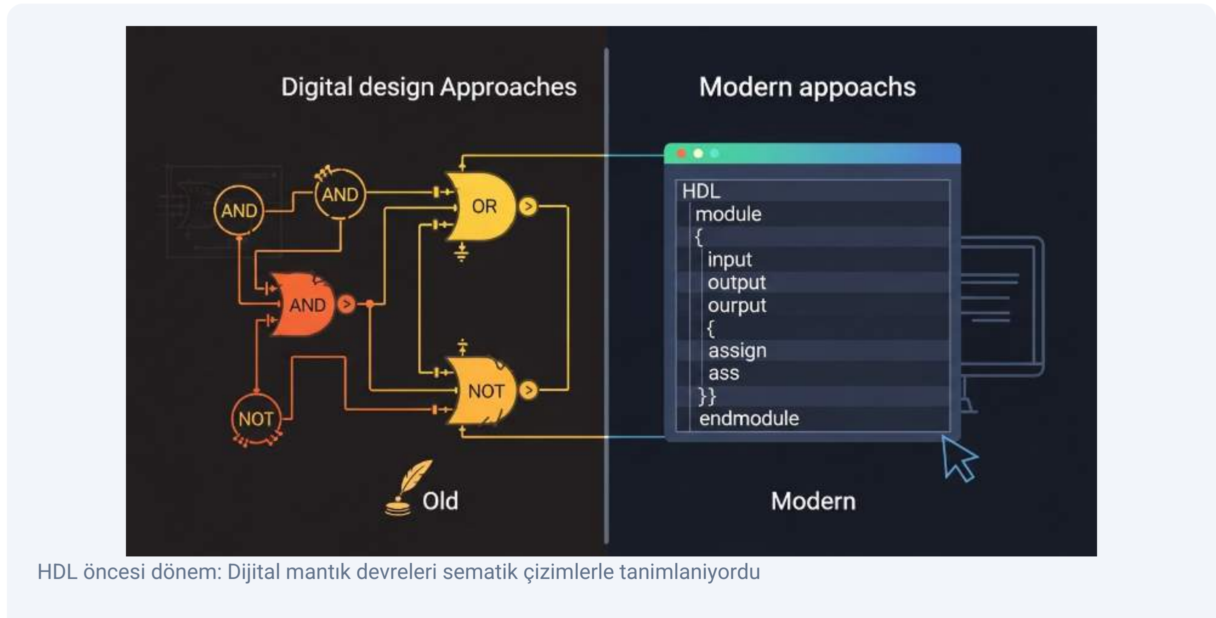
FPGA Üreticileri

Üretici	Eski Adı	Not
AMD	Xilinx	Pazar lideri, Vivado a..
Altera	Intel FPGA	Quartus Prime aracı

Microchip	Microsemi	Düşük güç FPGA'lar
Lattice	-	Küçük, düşük güçlü FPG..
Gowin	-	Uygun fiyatli
Efinix	-	Yeni nesil mimari
CologneChip	-	Açık kaynak arac desteği

HDL ile Tasarım

FPGA tasarımı için Donanım Tanımlama Dilleri (HDL) kullanılır. HDL'lerin ortaya çıkışından önce tasarımcılar mantık kapılarını elle sematik diyagramlar üzerinde çiziyordu.



HDL öncesi dönem: Dijital mantık devreleri sematik çizimlerle tanimlaniyordu

Dil	Özellik
VHDL	Katı, detaylı, IEEE standardı, ADA..
Verilog	Sade, C benzeri söz dizimi
SystemVerilog	Verilog'un genişletilmiş hali

Yüksek seviye sentez araçları da vardır: Vivado HLS (C/C++), Mathworks HDL Coder (MATLAB), Siemens Catapult (C++), Intel HLS. Ayrica Chisel, SpinalHDL, Amaranth gibi yazılım odakli alternatifler de mevcuttur.

DİKKAT

HDL dilleri yazılım dilleri gibi görünse de donanımı tanimlarlar. Bir 'for' döngüsü yazdiginizda bu bir döngü değil, donanım kopyalamasi anlamına gelir!

FPGA Uygulama Alanlari

FPGA'lar, paralel işlem yetenegi ve düşük gecikme süreleri sayesinde pek çok endustride kritik roller üstlenmektedir.

- Telekomunikasyon: 5G baz istasyonları, optik ag anahtarlama, protokol işlemcileri.
- Savunma ve Havacılık: Radar sinyal işleme, güvenli haberleşme, uydu sistemleri.
- Otomotiv: ADAS (Ileri Sürücü Destek Sistemleri), lidar veri işleme, arac içi ag gecitleri.
- Veri Merkezleri: Ag hızlandırma (SmartNIC), yapay zeka çıkarımları, video transcoding.
- Medikal: MRI/CT görüntü işleme, ultrason cihazları, hasta izleme sistemleri.
- Finans: Yüksek frekanli ticaret (HFT), risk analizi, piyasa verisi işleme.
- Endustriyel Otomasyon: Motor kontrol, PLC yerine gecebilen özel kontrol mantığı.
- Prototipleme: ASIC tasarımlarının FPGA üzerinde test edilmesi ve doğrulanması.

BİLGİ

FPGA'lar özellikle düşük gecikme (latency) gereken uygulamalarda one çıkar. Örneğin yüksek frekanli ticarette mikrosaniye altında işlem süresi gerekir ve bu ancak FPGA ile mümkündür.

SoC FPGA Nedir?

SoC FPGA (System on Chip FPGA), aynı çip üzerinde hem programlanabilir mantık (FPGA dokusunu) hem de sert işlemci cekirdeklerini (ARM Cortex-A, Cortex-R gibi) barındırır. Bu sayede yazılım ve donanım aynı çip üzerinde birlikte çalışabilir.

- AMD Zynq-7000: Çift çekirdekli ARM Cortex-A9 + Artix-7 FPGA dokusu.
- AMD Zynq UltraScale+ MPSoC: Dort çekirdekli ARM Cortex-A53 + Cortex-R5 + UltraScale+ FPGA.
- Intel Agilex: Dort çekirdekli ARM Cortex-A53 + Agilex FPGA dokusu.
- Microchip PolarFire SoC: RISC-V işlemci + PolarFire FPGA.

İPUCU

SoC FPGA ile Linux işletim sistemi işlemci tarafinda çalışırken, zaman kritik gorevler (motor kontrol, sinyal işleme) FPGA tarafinda donanım olarak yürütülür. AXI veriyolu ile iletişim kurulur.

FPGA Konfigurasyon Teknolojileri

Teknoloji	Kalicilik	Ornekler	Özellik
SRAM	Uçucu (güç kesil..	AMD Artix/Kintex..	Her açılışta yen..

Flash	Kalıcı (NVM)	Microchip PolarF..	Aninda başlar, d..
Antifuse	Tek seferlik (OTP)	Microsemi RTAX	Uzay/savunma, ra..

SRAM tabanlı FPGA'lar en yaygın kullanılan türlerdir. Ancak her güç açılışında konfigurasyon bitstream'inin harici bir Flash bellekten veya işlemciden yüklenmesi gerekir. Bu işlem genellikle milisaniyeler içerisinde tamamlanır.

FPGA Geliştirme Kartlari

FPGA öğrenmek için uygun fiyatli geliştirme kartları mevcuttur. Bazi popüler seçenekler:

Kart	FPGA	Fiyat Araligi	Hedef Kitle
Digilent Basys 3	Artix-7 (XC7A35T)	~150 USD	Başlangıç, unive..
Digilent Nexys A7	Artix-7 (XC7A100T)	~270 USD	Orta seviye, proje
Terasic DE10-Nano	Cyclone V SoC	~170 USD	SoC FPGA öğrenimi
Lattice iCEstick	iCE40HX1K	~30 USD	Açık kaynak arac..
Sipeed Tang Nano..	Gowin GW1NR-9	~15 USD	Düşük maliyetli ..

Bölüm 2

HDL Nedir ve Nasıl Çalışır?

Sematik tasarımdan HDL'lere geçiş, VHDL ve Verilog'un doğuşu

HDL Nedir?

HDL (Hardware Description Language - Donanım Tanımlama Dili), dijital devreleri metin tabanlı olarak tanımlamamızı sağlayan özel bir programlama dili türüdür. Bugün FPGA tasarımı denince akla ilk gelen VHDL ve Verilog, HDL ailesinin en yaygın iki üyesidir.

Peki HDL'ler neden ortaya çıktı? Bunu anlamak için önce HDL'lerden önceki döneme bakmamız gerekiyor.

HDL'lerden Önce: Sematik Tasarım Dönemi

VHDL ve Verilog gibi HDL'ler ortaya çıkmadan önce, dijital devreler sematik diyagramlar üzerinden tasarlanıyordu. Tasarımcılar kapı (gate) seviyesinde elle çizim yapıyordu: AND, OR, NOT kapıları, flip-flop'lar, bağlantı yolları... Hepsi tek tek kağıt üzerinde veya CAD araçlarında çiziliyordu.

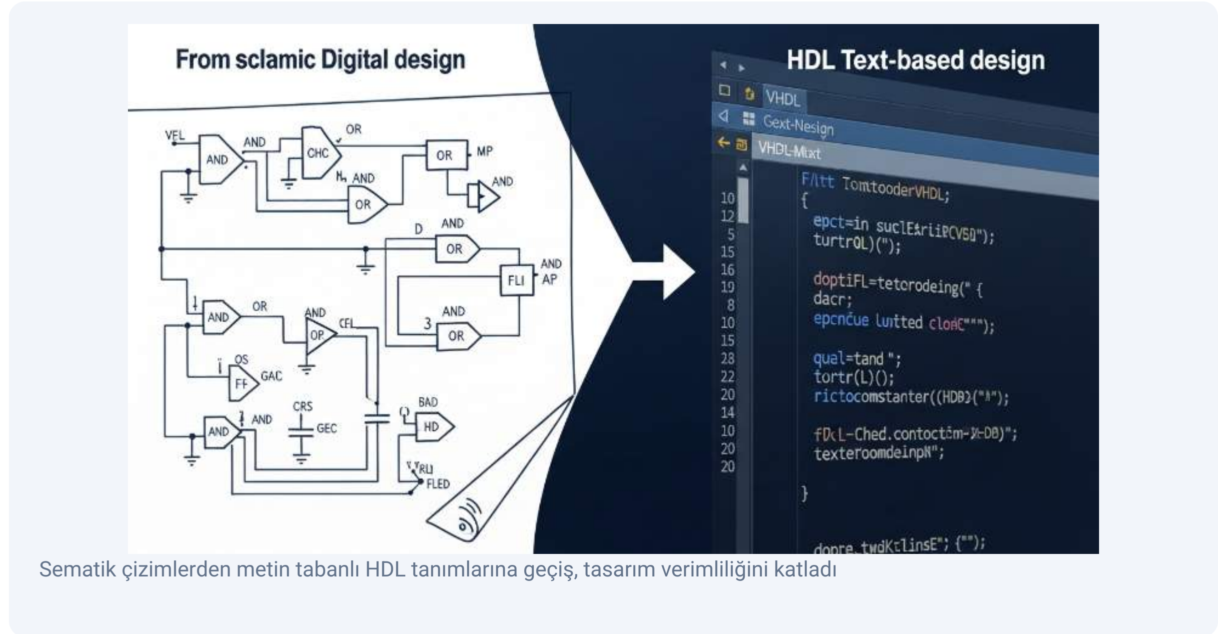
> **BİLGİ**
>
> Intel 4004 (1971) - dünyanın ilk ticari mikroişlemcisi - yaklaşık 2.300 transistörden oluşuyordu ve tamamen elle sematik çizimlerle tasarlandı. Tasarımcılar her bir transistörün yerini ve bağlantılarını tek tek belirlediler.

Küçük devreler için bu yöntem ise yarıyordu. Örneğin bir tam toplayıcı (full adder) devresini düşünelim: 3 girişi (A, B, Carry-in) ve 2 çıkışı (Sum, Carry-out) var. Bunu birkaç AND, OR, XOR kapısıyla çizebilirsiniz. Doğruluk tablosu da gayet anlaşılır:

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0

1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Ama devre büyüdükçe işler karmaşıklaştı. Binlerce, on binlerce, yuz binlerce kapı... Sematik diyagramlar sayfalarca uzuyor, baglantilari takip etmek imkansız hale geliyordu. Bir değişiklik yapmak istediğinizde tüm çizimi gözden geçirmeniz gerekiyordu. Hata riski çok yüksekti.



Sematik çizimlerden metin tabanlı HDL tanımlarına geçiş, tasarım verimliliğini katladı

Neden HDL'lere Geçildi?

1980'lerin sonlarına doğru entegre devrelerdeki transistör sayısı hızla arttı. Artık elle sematik çizmek pratik değildi. Tasarımcılar daha soyut, daha verimli bir yonteme ihtiyaç duyuyordu. İşte HDL'ler tam bu noktada devreye girdi.

- Karmaşıklık sorunu: Yuz binlerce kapıyı sematik olarak çizmek ve yönetmek imkansızlaştı
- Hata riski: Elle çizimde bağlantı hataları, eksik sinyaller çok sık yaşanıyordu
- Yeniden kullanılabilirlik: Sematik çizimler projeye özeldi, başka projede kullanmak zordu
- Simülasyon ihtiyacı: Devreyi üretmeden önce doğrulamak kritik hale geldi
- Takım çalışması: Büyük projelerde birden fazla tasarımcının paralel çalışması gerekiyordu

VHDL ve Verilog'un Doğuşu

İki büyük HDL neredeyse aynı dönemde ortaya çıktı ama farklı köklere sahipler:

VHDL - Very High Speed Hardware Description Language

VHDL, IEEE 1076 standardıdır. İlk sürümü 1987'de yayınlandı. ABD Savunma Bakanlığı'nin VHSIC (Very High Speed Integrated Circuits) programı kapsamında geliştirildi ve Ada programlama dilinden esinlenmiştir. Peki bugün Ada kullanan var mı? Çok az. Ama VHDL hala sapasağlam ayakta.

> **BİLGİ**
> VHDL case-insensitive (büyük/küçük harf duyarsız) bir dildir. Yani Led_Out, led_out ve LED_OUT hepsi aynı sinyaldir. Büyük-küçük harf farkı yoktur. Bu, okunabilirliği arttırır ama dikkatli olmazsanız karışıklığa da yol açabilir.

VHDL'in önemli revizyonları:

- IEEE 1076-1987: İlk standart. Temel yapılar tanımlandı
- IEEE 1076-1993: En yaygın kullanılan sürüm. Birçok iyileştirme ve düzeltme
- IEEE 1076-2008: Jenerik tipler, PSL (Property Specification Language) desteği
- IEEE 1076-2019: En güncel standart. Fixed-point ve floating-point paketleri, iyileştirilmiş sentez desteği

VHDL katı (strict) bir dildir. Bu ne demek?

- Birbirine baglanan sinyallerin tip, uzunluk ve yönleri uyuşmak zorunda
- Tüm nesneler (signal, variable, constant) kullanılmadan önce tanimlanmali
- Tip dönüşümü açıkça yapilmali - otomatik dönüşüm yok

> **İPUCU**
> VHDL'in bu katılığı bir dezavantaj gibi görünebilir ama aslinda en büyük avantajidir. Amaç: Hataları derleme (compile) aşamasında yakalamak, saatlerce süren simülasyondan sonra değil. Derleyici size 'bu sinyaller uyuşmuyor' dediginde, tasarımda bir mantık hatası var demektir.

VHDL'de Kütüphane ve Paket Yapısı

VHDL'de 'library' anahtar kelimesi, bir kütüphanedeki bileşenlere erişim sağlamak için kullanılır. Pratikte bir kütüphane bir klasör, paketler ise bu klasorun içindeki dosyalardır. C++'daki namespace kavramına benzer şekilde, ilgili tasarım bileşenleri gruplandirip yonetir.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

VHDL'de bazi temel kütüphaneler sunlardir:

- STANDARD: Temel veri tiplerini içerir (integer, boolean, bit vs.). Otomatik dahil edilir
- IEEE: En çok kullanılan kütüphane. STD_LOGIC_1164 ve NUMERIC_STD paketleri burada
- WORK: Kendi yazdiginiz tasarım dosyalarinin varsayilan kütüphanesi. Otomatik dahil edilir
- TEXTIO: Dosya okuma/yazma işlemleri için. Genellikle testbench'lerde kullanılır

- Kullanici tanımlı kütüphaneler: AMBA, UTIL gibi özel kütüphaneler de tanımlanabilir

Kutuphaneler 'paketlerden' oluşur. Paketlerin içinde tipler, alt tipler, sabitler, fonksiyonlar, prosedurler ve bileşen tanımları bulunur. Bir tasarımda kullanmak için önce 'library' ile kütüphaneyi, sonra 'use' ile ihtiyacınız olan paketi belirtirsiniz.

BİLGİ

En sık kullanılan iki paket: IEEE.STD_LOGIC_1164.ALL (std_logic ve std_logic_vector tipleri için) ve IEEE.NUMERIC_STD.ALL (signed, unsigned tipleri ve aritmetik işlemler için). Neredeyse her VHDL dosyasinin başında bu ikisini gorursunuz.

HDL, VHDL ve Verilog: Hangisi Ne?

Bu noktada sık sorulan bir soruyu netlestirelim: HDL bir dil mi, yoksa VHDL ve Verilog ayrı seyler mi?

HDL (Hardware Description Language) bir genel kavramdir, bir şemsiye terimdir. 'Donanım Tanımlama Dili' demektir. Dijital devreleri metin olarak tanımlayan dillerin tümüne verilen isimdir. VHDL ve Verilog ise bu şemsiyenin altindaki iki somut dildir.

Bunu şöyle düşünün: 'Programlama dili' bir genel kavramdir. C, Python, Java ise somut programlama dilleridir. Aynen oyle - HDL bir kategori, VHDL ve Verilog o kategorideki iki dildir.

```text
 HDL (Donanlm Tanlmlama Dili)
%%% VHDL (1987, IEEE 1076)
%%% Verilog (1984, IEEE 1364)
     %%% SystemVerilog (geni_letilmi_ hali)
```

Simdi bu hiyerarsideki her bir elemanı tek tek açıklayalım:

BİLGİ

HDL (Hardware Description Language): Dijital devreleri metin olarak tanımlayan dillerin genel adidir. Tek başına bir dil değildir, bir kategori ismidir. 'Programlama dili' nasıl C, Python, Java'yi kapsayan bir üst kavramsa, HDL de VHDL ve Verilog'u kapsayan bir üst kavramdir.

BİLGİ

VHDL (VHSIC Hardware Description Language): 1987'de IEEE 1076 olarak standartlaştırılan HDL dili. Ada programlama dilinden esinlenmiştir. Katı tip kontrolu vardır - sinyal tipleri, uzunluklari ve yönleri uyuşmak zorundadir. Hataları derleme aşamasında yakalamak için tasarlanmistir. Avrupa ve savunma sektöründe daha yaygındır. Daha ayrıntılı yazılır ama okunabilirliği yuksektir.

BİLGİ

Verilog: 1984'te oluşturulan, 1995'te IEEE 1364 olarak standartlaştırılan HDL dili. C programlama dilinden esinlenmiştir. VHDL'e göre daha kompakt ve hızlı yazılır. Daha az katı tip kontrolu vardır. ABD ve Asya'da daha yaygındır. Daha az satırla aynı tasarımı ifade edebilirsiniz.

BİLGİ

SystemVerilog: Verilog'un genişletilmiş halidir. IEEE 1800 standardıdır. Verilog'a nesne yonelimli programlama (OOP), gelişmiş doğrulama özellikleri (assertions, coverage, constrained random testing) ve daha güçlü veri tipleri eklenmiştir. Özellikle doğrulama (verification) tarafinda çok gucludur. Bugün endustrde tasarım için Verilog/SystemVerilog, doğrulama için SystemVerilog kullanımı yaygındır.

İkisi de aynı işi yapar: Dijital donanımı metin olarak tanımlamak. Ama farklı söz dizimleri ve farklı felsefeleri vardır:

- VHDL: 'Her seyi açıkça yaz, derleyici seni korusun' yaklasimi. Daha fazla kod ama daha az hata
- Verilog: 'Kisa yaz, hızlı ilerle' yaklasimi. Daha az kod ama dikkatli olmak gerekir

Surecteki yerlerine bakarsak:

- 1971 ve öncesi: Elle sematik çizim. Intel 4004, 2.300 transistör - elle yapılabildiği son dönem
- 1980'ler: Transistor sayısı yuz binleri asti. Elle çizim imkansızlaştı
- 1984-1987: HDL'ler doğdu. Artık devreler metin olarak tanimlaniyor
- Bugün: Milyarlarca transistorlü çipler var. HDL olmadan bunları tasarlamak düşünülemez

İPUCU

Kisaca: HDL bir devrim noktasidir - elle cizimden metin tabanlı tasarıma geçiş. VHDL ve Verilog ise bu devrimi gerçekleştiren iki aractir. Hangisini seçerseniz secin, ikisi de sizi aynı yere goturur.

Verilog ve SystemVerilog

Verilog, 1984'te Gateway Design Automation tarafından ticari bir urun olarak geliştirildi. C programlama diline benzer söz dizimi ile daha kompakt ve hızlı yazilabilir. 1995'te IEEE 1364 standardı olarak kabul edildi. Sonradan SystemVerilog ile genişletildi.

Her iki dil de aynı işi yapar: Dijital donanımı metin olarak tanımlamak. Hangisinin daha iyi olduğu tartismasi yıllardır sürer ama gerçek su ki ikisi de endustride yaygın kullanılır. Avrupa ve savunma sektöründe VHDL, ABD ve Asya'da Verilog daha yaygındır.

HDL Nasıl Çalışır?

HDL ile yazdigimiz kod, bilgisayar yazılımı gibi sırasıyla çalışan komutlar değildir. Biz aslinda bir donanım yapısı tanımlıyoruz. Örneğin 'bir AND kapısı koy, çıkışını su flip-flop'a bagla' diyoruz - ama bunu metin olarak yaziyoruz.

Surecin temeli şöyledir:

- 1. HDL Kodu Yazımı: VHDL veya Verilog ile tasarımınızı metin olarak tanımlıyorsunuz
- 2. Simülasyon: Yazdginiz kodu sanal ortamda test ediyorsunuz (ModelSim, Vivado Simulator vs.)

- 3. Sentez (Synthesis): Arac, yazdiginiz kodu gerçek mantık kapilarina (LUT, FF vs.) dönüştürüyor
- 4. Place & Route: Sentezlenen yapılar FPGA içerisine yerleştiriliyor ve birbirine bağlanıyor
- 5. Bitstream: FPGA'ya yüklenecek konfigurasyon dosyası oluşturuluyor

DİKKAT

Kritik fark: Yazilimda kod satırları sırayla çalışır. HDL'de tanimladiginiz yapılar eş zamanlı (concurrent) çalışır. İki ayrı process veya assign ifadesi aynı anda, paralel olarak gerceklesir. Bu, HDL'yi ogrenen yazilimcilarin en çok zorlandığı noktadir.

Sematik vs HDL: Bir Karşılaştırma

Özellik	Sematik Tasarım	HDL Tasarım
Tanımlama	Görsel çizim	Metin tabanlı kod
Karmaşık devreler	Çok zor, hata riski yü..	Yonetilebilir, modüler
Yeniden kullanım	Sınırlı	Component, module ile ..
Simülasyon	Sınırlı	Kapsamli testbench des..
Takım çalışması	Zor	Versiyon kontrolu ile ..
Soyutlama seviyesi	Kapı seviyesi	Davranışsal seviyeye k..
Dokumantasyon	Ayri çizim gerekli	Kod kendini açıklar
Degisiklik	Tüm çizimleri gözden g..	Ilgili modulu güncelleme

Basit Bir Örnek: AND Kapisi

Sematik tasarımda bir AND kapısını çizmek için CAD aracında kapinin sembolunu koyar, girişleri ve çıkışı baglardiniz. HDL'de ise aynı seyi birkaç satırla tanımlıyorsunuz:

VHDL

```
-- VHDL ile AND kapısı tanımlama
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity and_gate is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           Y : out STD_LOGIC);
end and_gate;

architecture Behavioral of and_gate is
begin
    Y <= A and B;
end Behavioral;
```

Bu kadar. Aynı işlem Verilog'da daha da kisa:

```verilog
// Verilog ile AND kapısı tanımlama
module and_gate (
    input  A,
    input  B,
    output Y
);
    assign Y = A & B;
endmodule
```

Simdi düşünün: Bu basit bir AND kapısı. Ama ya 32-bit bir ALU tasarliyorsaniz? Yuzlerce kapı, flip-flop, MUX... Sematik ile çizmek gunler alir. HDL ile modüler şekilde yazarsiniz, test edersiniz, tekrar kullanırsınız.

HDL'lerin Getirdigi Devrim

HDL'ler sadece bir yazım kolaylığı değildi. Tasarım felsefesini kökten degistirdiler:

- Davranışsal tanımlama: Devrenin ne yaptigini tanımlıyorsunuz, nasıl yapacagini sentez aracı buluyor
- IP Core kavrami: Hazir tasarım blokları satın alinabiliyor veya açık kaynak olarak kullanılabiliyor
- Otomatik sentez: Yazdiginiz yüksek seviyeli kod, otomatik olarak kapı seviyesine dönüştürülüyor
- Kapsamli doğrulama: Testbench'ler ile devrenizi üretmeden önce detaylı test edebiliyorsunuz
- Platform bagimsizligi: Aynı VHDL kodu farklı FPGA'larda veya ASIC'lerde kullanılabiliyor

> HDL'ler, dijital tasarımı demokratikleştirdi. Artık bir tasarımcı, transistör seviyesinde dusunmek zorunda kalmadan karmaşık sistemler olusturabiliyor.

Bugunun Dünyası: HLS ve Otesi

HDL'ler büyük bir devrimdi ama hikaye burada bitmiyor. Bugün High-Level Synthesis (HLS) araçları, C/C++ gibi yüksek seviyeli dillerden doğrudan donanım tasarımı üretebiliyor. AMD/Xilinx Vitis HLS, Intel HLS Compiler gibi araclar bu işi yapiyor.

Ancak HLS henuz HDL'lerin yerini almadi. Performans kritik tasarımlarda, zamanlama hassasiyeti gereken yerlerde hala VHDL ve Verilog tercih ediliyor. HLS daha çok hızlı prototipleme ve algoritma hızlandırma için kullaniliyor.

İPUCU

Tavsiye: FPGA öğrenmeye başlıyorsanız, önce VHDL veya Verilog'u iyi öğrenin. Temeli anlamadan HLS kullanmak, otomatik vites kullanip clutch'in ne olduğunu bilmemek gibidir. Temel sağlamsa, HLS'e geçiş çok daha kolay olur.

Bölüm 3

VHDL Dili

Kütüphane, paket, veri tipleri, entity ve architecture

VHDL Temelleri

VHDL (Very High Speed Integrated Circuit Hardware Description Language), IEEE standardı olan bir donanım tanımlama dilidir. IEEE-1076 numaralı standart altında tanımlanmıştır.

- IEEE standardıdır (IEEE-1076), ilk sürümü 1987'de yayınlanmıştır.
- Büyük/küçük harf duyarsizdir: My_signal = my_signal = MY_SIGNaL
- ADA programlama diline dayanır.
- Önemli revizyonlar: 1987, 1993, 2008, 2019
- Katı (strict) bir dildir - tip, uzunluk ve yonler uyuşmalıdır.
- Tüm nesneler kullanılmadan önce bildirilmelidir.
- Amaç: Hataları derleme zamanında yakalamak, saatlerce simülasyondan sonra değil.

Kütüphane ve Paketler

VHDL'de 'library' anahtar kelimesi kutuphanelere erişim sağlar. Pratikte bir kütüphane bir klasör, paketler o klasordeki dosyalardır. C++'daki namespace kavramına benzer.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
```

Kütüphane	İçeriği
STANDARD	Temel veri tipleri (boolean, integ..
IEEE	std_logic_1164, numeric_std paketl..
WORK	Kullanıcının kendi tasarım dosyaları
TEXTIO	Dosya okuma/yazma (simülasyon)

STANDARD Paketi Temel Tipler

```vhdl
-- Mantık tipleri:
type BOOLEAN is (FALSE, TRUE);
type BIT is ('0', '1');

-- Sayisal tipler:
type INTEGER is range -2147483648 to 2147483647;

-- Alt tipler:
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;

-- Dizi tipleri:
type STRING is array (POSITIVE range <>) of CHARACTER;
type BIT_VECTOR is array (NATURAL range <>) of BIT;
```

std_logic_1164 Paketi

Bu paket, std_logic ve std_logic_vector tiplerini tanımlar. std_logic 9 farklı değer alabilir:

Değer	Anlamı	Açıklama
'U'	Başlatılmamış	Henuz atama yapilmamis
'X'	Zorunlu Bilinmeyen	Catisma durumu
'0'	Zorunlu 0	Mantık 0 (LOW)
'1'	Zorunlu 1	Mantık 1 (HIGH)
'Z'	Yüksek Empedans	Bağlı değil (tri-state)
'W'	Zayif Bilinmeyen	Zayif sinyal çatışması
'L'	Zayif 0	Zayif pull-down
'H'	Zayif 1	Zayif pull-up
'-'	Farketmez	Optimizasyon için

BİLGİ

Neden sadece '0' ve '1' yetmez? I2C, bellek arayüzleri gibi uygulamalarda 'Z' (yüksek empedans) ve 'X' (bilinmeyen) durumlar da gereklidir.

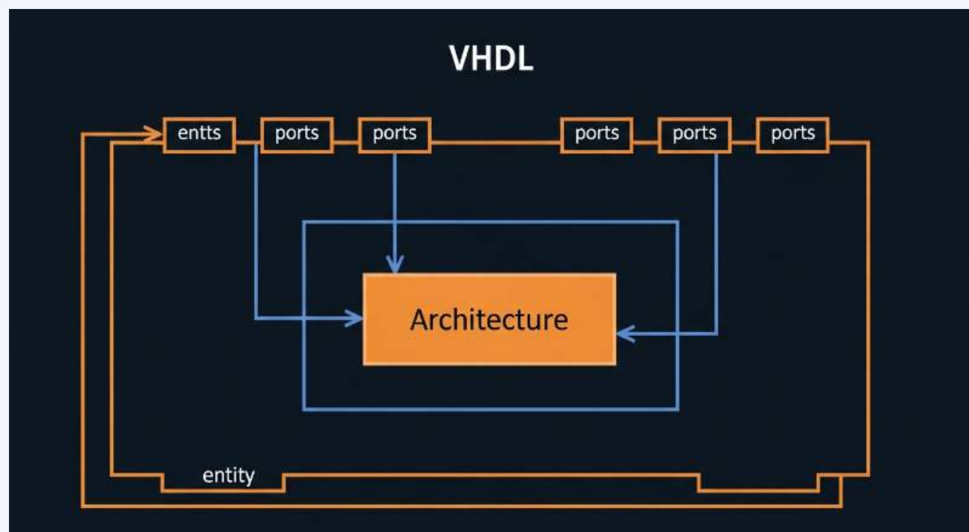
numeric_std Paketi

```vhdl
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
type SIGNED is array (NATURAL range <>) of STD_LOGIC;

function "+" (L, R: UNSIGNED) return UNSIGNED;
function TO_INTEGER (ARG: UNSIGNED) return NATURAL;
function TO_UNSIGNED (ARG, SIZE: NATURAL) return UNSIGNED;
```

> **DİKKAT**
>
> Synopsys kütüphaneleri (std_logic_arith, std_logic_unsigned) eski ve standart dışı kutuphanelerdir. Yeni tasarımlarda mutlaka IEEE numeric_std kullanın.

Entity (Varlik)

Entity, dijital bilesenin dış arayüzünü tanımlar. Generic parametreleri ve portlari belirtir.



Entity dış arayüzü, Architecture iç uygulamayı tanımlar

```vhdl
entity counter is
generic (
    N : integer := 8          -- Generic parametre
);
port (
        clk         :  in    std_logic;                              -- Saat
        rst         :  in    std_logic;                              -- Reset
        en_i        :  in    std_logic;                              -- Enabl
    mode_i  : in  std_logic;                         -- Mod seçimi
        cntr1_o     :  out   std_logic_vector(N-1 downto 0);         -- Sayıc
        cntr2_o     :  out   std_logic_vector(N-1 downto 0)          -- Sayıc
);
end counter;
```

İPUCU

Port isimlerinde _i son eki giriş, _o son eki çıkış için kullanılır. Bu isimlendirme kuralı kodu okumay kolaylaştırır.

Architecture (Mimari)

Architecture, entity'nin iç davranışını tanımlar. Sabitler, tipler, sinyaller bildirim kısmında (begin'den önce), davranış ise begin-end arasında yazılır.

```vhdl
architecture Behavioral of counter is
 -- Bildirim kısmı
constant c_single : integer := 1;
constant c_double : integer := 2;

type t_state is (S_IDLE, S_COUNT, S_PAUSE);
signal state : t_state := S_IDLE;

signal cntr1 : integer range 0 to 2**N-1 := 0;
signal cntr2 : unsigned(N-1 downto 0) := (others => '0');

begin
 -- Davranış kısmı: process blokları ve eş zamanlı atamalar
...
end Behavioral;
```

- 'constant' ile sabit değerler tanımlanır.
- 'type' ile özel tipler oluşturulur (genelde durum makinesi için).
- 'signal' ile iç yapılar bildirilir - kablo, FF veya BRAM olarak sentezlenebilir.
- Tip eşleşmesi zorunludur! Atamanin sag ve sol taraflari aynı tipte olmalıdır.

Tip Dönüştürme Fonksiyonlari

```vhdl
 -- numeric_std paketindeki dönüştürme fonksiyonları:
-- to_unsigned()       : integer -> unsigned
-- to_signed()         : integer -> signed
-- to_integer()        : unsigned/signed -> integer
-- std_logic_vector()  : unsigned/signed -> std_logic_vector
-- unsigned()          : std_logic_vector -> unsigned
-- signed()            : std_logic_vector -> signed

-- Örnek:
cntr1_o <= std_logic_vector(to_unsigned(cntr1, cntr1_o'length));
cntr2_o <= std_logic_vector(cntr2);
```

Değişken (Variable) ve Sinyal (Signal) Farkı

VHDL'de variable, bir process içinde tanimlanan yerel depolama elemanidir. Signal ise global bir yapıdır.

- variable ':=' ile atanır, signal '<=' ile atanır.
- variable process içinde yereldir, signal ise globaldir.
- variable değeri anında güncellenir, signal değeri process sonunda güncellenir.

```vhdl
process(a, b)
    variable temp : integer := 0;
begin
    temp := a + b;        -- anında güncellenir
    c <= temp;        -- process sonunda güncellenir
end process;
```

VHDL Operatorleri

Kategori	Operatorler	Örnek
Mantıksal	and, or, nand, nor, xo..	y <= a and b;
Karşılaştırma	=, /=, <, >, <=, >=	if (a = b) then
Aritmetik	+, -, *, /, mod, rem, ..	sum <= a + b;
Kaydırma	sll, srl, sla, sra, ro..	shifted <= data sll 2;
Birlestirme	&	full <= upper & lower;

DİKKAT

Çarpma (*) operatoru sentezlenebilir ve DSP dilimi kullanir. Bölme (/) operatoru ise genellikle sentezlenemez - sadece 2'nin kuvveti ile bölme işlemleri (sag kaydırma) sentezlenir.

Yararlı Attribute'lar

VHDL'de attribute'lar nesnelerin özelliklerini sorgulamak için kullanılır. Sentez ve simülasyonda sıklıkla kullanılan önemli attribute'lar:

Attribute	Kullanım	Açıklama
'length	signal'length	Vektorun bit sayısı
'range	signal'range	Vektorun aralik bilgisi

'left / 'right	signal'left	En sol / en sag indeks
'high / 'low	signal'high	En büyük / en küçük in..
'event	clk'event	Sinyalde değişiklik ol..
'rising_edge	rising_edge(clk)	Yükselen kenar algılama

VHDL

```
-- Attribute kullanım örnekleri
signal data : std_logic_vector(7 downto 0);

data'length   -- 8 döndürür
data'range    -- 7 downto 0 döndürür
data'left     -- 7 döndürür
data'right    -- 0 döndürür
data'high     -- 7 döndürür
data'low      -- 0 döndürür
```

Generate İfadesi

Generate ifadesi, tekrarlayan donanım yapilarini otomatik oluşturmak için kullanılır. for-generate ve if-generate olmak üzere iki türü vardır.

VHDL

```
-- for-generate: 8 adet FF olu_turma
GEN_FF : for i in 0 to 7 generate
    process (clk) begin
        if rising_edge(clk) then
            q(i) <= d(i);
        end if;
    end process;
end generate GEN_FF;

-- if-generate: Ko_ullu donanım olu_turma
GEN_PIPE : if PIPELINE_ENABLED generate
    process (clk) begin
        if rising_edge(clk) then
            pipe_reg <= data_in;
        end if;
    end process;
end generate GEN_PIPE;
```

BİLGİ

Generate ifadesi derleme zamanında değerlendirilir, çalışma zamanında değil. Yani for-generate bir döngü değil, donanım kopyalamasıdır. 8 iterasyonlu bir for-generate, 8 ayrı donanım blogu oluşturur.

Assert İfadesi (Simülasyon)

Assert ifadesi, simülasyon sırasında koşulları kontrol etmek için kullanılır. Sentezlenemez, sadece testbench'lerde kullanılır.

```vhdl
-- Assert örnekleri
assert (data_width > 0)
         r e p o r t   " V e r i   g e n i _ l i   i   s i f i r d a n   b ü y ü k   o l m a l 1 ! "
    severity failure;

assert (output = expected)
         r e p o r t   " Ç 1 k 1 _   b e k l e n e n   d e g e r l e   u y u _ m u y o r ! "
    severity error;

-- Severity seviyeleri: note, warning, error, failure
```

Testbench Temelleri

Testbench, tasarımı simülasyon ortamında test etmek için yazılan VHDL kodudur. Port listesi olmayan bir entity'dir ve test edilen bileşeni (DUT - Design Under Test) içinde örnekler.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity tb_counter is
-- Testbench entity'sinde port yoktur!
end tb_counter;

architecture Behavioral of tb_counter is
    signal clk : std_logic := '0';
    signal rst : std_logic := '1';
    signal count : std_logic_vector(7 downto 0);

    constant CLK_PERIOD : time := 10 ns;
begin
    -- Saat üreteci
    clk <= not clk after CLK_PERIOD / 2;

    -- DUT örnekleme
    DUT : entity work.counter
    port map (
        clk => clk,
        rst => rst,
        count_o => count
    );

    -- Test süreci
    STIM : process begin
        rst <= '1';
        wait for 100 ns;
        rst <= '0';
        wait for 1000 ns;
        assert false report "Simülasyon tamamlandi" severity note;
        wait;
    end process;
end Behavioral;
```

İPUCU

Testbench yazarken: 1) Önce reset uygulayip bilinen bir duruma getirin. 2) Farklı giriş kombinasyonlarini test edin. 3) Assert ile çıkışları otomatik doğrulayın. 4) Simülasyon sonunda 'wait;' ile durdurun.

Bölüm 4

FPGA Tasarım Akışı

Vivado ile tasarım, sentez ve uygulama adımları

EDA Araçları

EDA (Electronic Design Automation), entegre devre ve baskı devre kartı tasarımı için kullanılan yazılım araçlarıdır. FPGA tasarımı için en yaygın kullanılan arac AMD/Xilinx'in Vivado'sudur.

Vivado tasarım akisi: RTL'den bitstream'e kadar olan adımlar

Vivado Tasarım Adimlari

- 1. RTL Tasarımı: VHDL/Verilog kodunun yazılması.
- 2. Fonksiyonel Simülasyon: Kodun davranışsal doğruluğu test edilir.
- 3. RTL Analizi (Elaboration): Tasarimin yapisal olarak incelenmesi.
- 4. G/C Planlama (I/O Planning): Pin atamaları yapılır.
- 5. Sentez (Synthesis): HDL kodu, FPGA temel bileşenlerine (LUT, FF) dönüştürülür.
- 6. Zamanlama Kisitlari: Saat frekansı ve zamanlama gereksinimleri belirlenir.
- 7. Uygulama (Place & Route): Mantık bileşenleri FPGA üzerine yerleştirilir ve bağlanır.
- 8. Bitstream Üretimi: FPGA'ya yüklenecek yapılandırma dosyası oluşturulur.

Tasarım Giriş ve Cikislari

Giriş	Çıkış
RTL Kaynak Kodlari (.vhd/.v)	FPGA Yapılandırma (bitstream)
Test Bench (simülasyon için)	Donanım Tanimi (xsa, yazılım için)
Kısıtlamalar (zamanlama, G/C)	Raporlar (kullanım, zamanlama)
IP Cekirdekler (AMD veya 3. parti)	Flash Programlari (bin/mcs)
Blok Tasarım (CPU tabanlı sistem)	Netlist (sentez çıktısı)

Sentez Nedir?

Sentez, HDL kodunu hedef FPGA'nın temel bileşenlerine (LUT, FF vb.) dönüştüren işlemdir. Çıktısı 'teknoloji haritalanmış netlist' olarak adlandırılır. Basitçe: kodunuz donanım parçalarına çevriliyor.

BİLGİ

Vivado'da sadece 'Generate Bitstream' tuşuna basarak tüm akisi otomatik başlatabilirsiniz. Aracınız gerisini halleder!

Sentezlenemeyen Yapilar

Bazi VHDL yapıları sadece simülasyonda çalışır, sentezlenemez:

- 'wait' ifadeleri (wait until rising_edge haric): wait for 10 ns;
- 'after' ifadesi: signal_a <= '1' after 10 ns;
- Dosya G/C işlemleri (textio paketi): Sadece simülasyon için.
- 'real' tipi sinyaller (math_real paketi): Donanımda karşılığı yoktur.
- Bölme işlemi ('/'): Sentezlenebilir ama sonuç kabul edilemez. IP kullanın.

FPGA Kaynak Kullanımı

DİKKAT

FPGA kaynaklarinin %100'unu kullanmak pratikte mümkün ve istenir değil. Yüksek kullanım oranı yönlendirme zorluklarına, uzun kritik yollara ve zamanlama hatalarına yol açar. Genellikle %70-80 kullanım oranı hedeflenir.

Kısıtlama Dosyalari (XDC)

XDC (Xilinx Design Constraints) dosyaları, FPGA tasarımında fiziksel ve zamansal kısıtlamaları tanımlar. Pin atamaları, saat tanımları ve zamanlama gereksinimleri bu dosyada belirtilir.

```tcl
## Saat tanımlama (100 MHz)
create_clock -period 10.000 -name sys_clk [get_ports clk]

## Pin atamaları
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]

## LED pin atamaları
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]

## Buton giri_leri
set_property PACKAGE_PIN T18 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports rst]
```

İPUCU

XDC dosyasını yazarken geliştirme kartinin referans kilavuzundaki pin haritasini kullanın. Yanlis pin ataması FPGA'ya zarar verebilir!

IP Core Kullanımı

IP (Intellectual Property) Core, önceden tasarlanmış ve doğrulanmış dijital devre bloklaridir. Vivado'nun IP Catalog'undan hazır IP'ler kullanabilirsiniz.

- Saat Yonetimi: Clocking Wizard - PLL/MMCM ile saat çarpma, bölme, faz kaydırma.
- Bellek: Block Memory Generator - BRAM yapilandirmasi, ROM, RAM.
- Matematik: Floating Point, CORDIC - Kayar nokta işlemi, trigonometrik fonksiyonlar.
- İletişim: AXI UART Lite, AXI IIC - Seri haberleşme arayüzleri.
- Islemci: MicroBlaze - Yumusak işlemci çekirdeği (soft processor).
- DSP: FIR Compiler, FFT - Dijital sinyal işleme blokları.

BİLGİ

IP Core kullanımı geliştirme süresini önemli ölçüde kısaltır. Ancak her IP'nin kaynak tüketimi ve zamanlama etkisi vardır. Kullanım raporlarini dikkatli inceleyin.

Statik Zamanlama Analizi (STA)

STA, tasarımdaki tüm veri yollarinin zamanlama gereksinimlerini karşılayıp karsilamdigini kontrol eder.

Simülasyon yapmadan, tüm olası yolları analiz eder.

- Setup Analizi: Verinin saat kenarından yeterince önce kararlı olup olmadigini kontrol eder.
- Hold Analizi: Verinin saat kenarından sonra yeterince kararlı kalıp kalmadığını kontrol eder.
- WNS (Worst Negative Slack): En kotu setup zamanlama payı. Pozitif olmalı!
- WHS (Worst Hold Slack): En kotu hold zamanlama payı. Pozitif olmalı!
- TNS (Total Negative Slack): Tüm ihlallerin toplamı. Sıfır olmalı!

DİKKAT

WNS veya WHS negatif ise tasarımınız güvenilir değildir! Çözümler: saat frekansını düşürün, pipeline ekleyin, mantık karmaşıklığını azaltın veya fiziksel kısıtlamalar tanımlayın.

Tasarım Optimizasyon İpuçları

- Kayit (Register) dengeleme: Vivado'nun retiming özelliğini aktif edin.
- Kaynak paylaşımı: Aynı çarpıcıyı farklı zamanlarda kullanın.
- ROM için başlangıç değerleri: LUT yerine BRAM kullanarak alan tasarrufu sağlayın.
- Gereksiz mantığı kaldırın: Kullanilmayan sinyaller ve portlar kaynak tüketir.
- Hiyerarsiyi koruyun: out-of-context (OOC) sentez ile modülleri bağımsız optimize edin.
- Saat alanı sayisini minimumda tutun: Her ek saat alanı CDC karmaşıklığı ekler.

Bölüm 5

Kombinasyonel Mantık Tasarımı

Kapı seviyesi, hiyerarşik tasarım ve eş zamanlı atamalar

Kombinasyonel Mantık Tasarımı

Kombinasyonel mantikte çıkış sadece anlik girişlere bağlıdır. Bellegi yoktur, saat sinyali gerekmez. Çıkış, girişler değiştiğinde anında değişir.

Kapı Seviyesi (Yapisal) Tasarım

En temel tasarım yöntemi, mantık kapılarını doğrudan VHDL'de tanımlamaktır. std_logic_1164 paketinde and, or, not, nand, nor, xor, xnor fonksiyonları tanımlıdır.

Örnek: y = abc' + a'bc + a'b'c' fonksiyonunu VHDL ile yazalım:

VHDL

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity gate_level_comb is
port (
    a : in  std_logic;
    b : in  std_logic;
    c : in  std_logic;
    y : out std_logic
);
end gate_level_comb;

architecture Behavioral of gate_level_comb is
begin
    y <= (a and b and (not c)) or
         ((not a) and b and c) or
         ((not a) and (not b) and (not c));
end Behavioral;
```

İPUCU

Bu kod, process blogu dışında yazildi. Yani eş zamanlı (concurrent) bir atamadir ve doğrudan kombinasyonel mantık çıkarır.

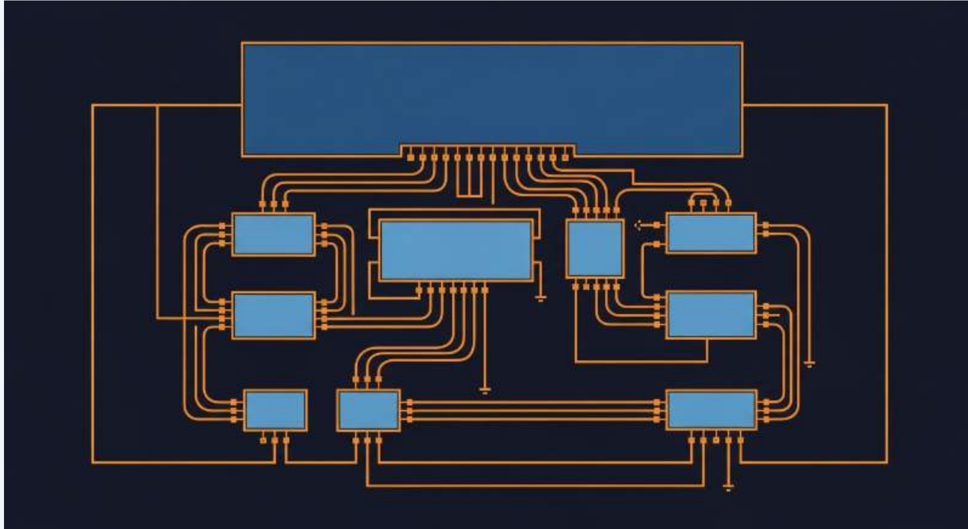
Ara Sinyaller ile Tasarım

Karmaşık ifadeleri parçalayarak daha okunabilir hale getirebilirsiniz:

```vhdl
architecture Behavioral of gate_level_comb2 is
signal sig1 : std_logic := '0';
signal sig2 : std_logic := '0';
signal sig3 : std_logic := '0';

begin
    sig1 <= (a and b and (not c));
    sig2 <= ((not a) and b and c);
    sig3 <= ((not a) and (not b) and (not c));

    y <= sig1 or sig2 or sig3;
end Behavioral;
```

Hiyerarsik Tasarım

Büyük tasarımları küçük alt bileşenlere bolup, bunları bir ust modilde birlestirmeye hiyerarsik tasarım denir. Her alt bileşen ayrı bir entity/architecture çiftidir.



Hiyerarsik tasarım: Alt bileşenler bir ust modülde birlestiririlir

```vhdl
-- Alt bile_en tan1mlama (component)
component comp1 is
port (
    a : in  std_logic;
    b : in  std_logic;
    c : in  std_logic;
    y : out std_logic
);
end component;

-- Alt bile_en örnekleme (instantiation)
I1 : comp1
port map(
    a => a,
    b => b,
    c => c,
    y => comp1_out
);
```

Eş Zamanlı Atamalar

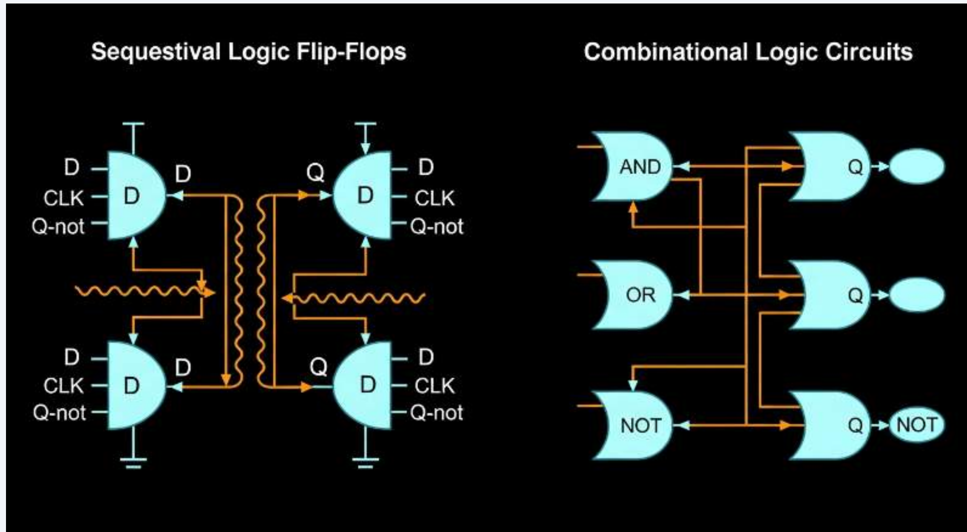
Process blogu dışındaki atamalar eş zamanlı (concurrent) çalışır. Toplama, çıkarma, koşullu atama gibi işlemler doğrudan yazilabilir:

```vhdl
-- Ko_ullu sinyal atamas1 (when/else)
y_o <= std_logic_vector(signed(a_i) + signed(b_i))
       when s_i = '0' else
       std_logic_vector(signed(a_i) - signed(b_i));
```

Process ile Kombinasyonel Tasarım

Daha karmaşık kombinasyonel mantık için process blogu kullanılabilir. Hassasiyet listesine tüm okunan sinyaller eklenir (saat sinyali değil):

```vhdl
process (s_i, s_a, s_b) begin
    if (s_i = '0') then
        y_o <= std_logic_vector(s_a + s_b);
    else
        y_o <= std_logic_vector(s_a - s_b);
    end if;
end process;
```



Kombinasyonel mantık: belleksiz, girişe bağlı. Ardisil mantık: Flip-Flop ile saat sinyaline bağlı

Kombinasyonel Process'te Dikkat Edilecekler

> **DİKKAT**
>
> Kombinasyonel process blogunda 3 kritik kural vardır: 1) Hassasiyet listesinde tüm okunan sinyaller olmalı. 2) Tüm if ifadelerinde else dalii olmalı (yoksa latch çıkar!). 3) Bir sinyal aynı process'te hem okunup hem yazilmamali (geri besleme döngüsü oluşur).

- Eksik hassasiyet listesi: Yanlis simülasyon davranışı oluşturur.
- Eksik else dali: İstenmeyen latch çıkarır. Vivado uyarı verir.
- Geri besleme döngüsü: Sinyal hem okunup hem yazilirsa salınım olabilir.

Multiplexer (MUX) Tasarımı

Multiplexer, birden fazla giriş arasından birini seçim sinyaline göre çıkışa yönlendiren devredir. VHDL'de with/select veya when/else ile tanımlanabilir.

```vhdl
-- 4:1 MUX - with/select kullanımı
with sel select
    y <= a when "00",
         b when "01",
         c when "10",
         d when others;

-- 4:1 MUX - when/else kullanımı
y <= a when sel = "00" else
     b when sel = "01" else
     c when sel = "10" else
     d;
```

```
-- 4:1 MUX - process ile case kullanımı
process (sel, a, b, c, d) begin
    case sel is
        when "00"   => y <= a;
        when "01"   => y <= b;
        when "10"   => y <= c;
        when others => y <= d;
    end case;
end process;
```

Decoder (Kod Çözücü) Tasarımı

Decoder, N bitlik girişi 2^N bitlik çıkışa dönüştürür. Sadece bir çıkış aktif olur. Bellek adres çözme ve çip seçimi için kullanılır.

```vhdl
-- 3:8 Decoder
process (addr) begin
    decode_out <= (others => '0');  -- varsayilan: tüm çıkı_l
    case addr is
        when "000" => decode_out(0) <= '1';
        when "001" => decode_out(1) <= '1';
        when "010" => decode_out(2) <= '1';
        when "011" => decode_out(3) <= '1';
        when "100" => decode_out(4) <= '1';
        when "101" => decode_out(5) <= '1';
        when "110" => decode_out(6) <= '1';
        when "111" => decode_out(7) <= '1';
        when others => null;
    end case;
end process;
```

Öncelik Kodlayici (Priority Encoder)

Priority encoder, birden fazla giriş aktif oldugunda en yüksek öncelikli girişi kodlar. Kesme (interrupt) yönetimi gibi uygulamalarda kullanılır.

```vhdl
-- 8:3 Öncelik Kodlayici
process (req) begin
    code <= "000";
    valid <= '0';
    if    req(7) = '1' then code <= "111"; valid <= '1';
    elsif req(6) = '1' then code <= "110"; valid <= '1';
    elsif req(5) = '1' then code <= "101"; valid <= '1';
    elsif req(4) = '1' then code <= "100"; valid <= '1';
    elsif req(3) = '1' then code <= "011"; valid <= '1';
    elsif req(2) = '1' then code <= "010"; valid <= '1';
    elsif req(1) = '1' then code <= "001"; valid <= '1';
    elsif req(0) = '1' then code <= "000"; valid <= '1';
    end if;
end process;
```

> **BİLGİ**
> Öncelik kodlayicida if/elsif yapısı kullanılır cunku öncelik sirasi onemlidir. Eger tüm girişler esit oncelikte ise case/when yapısı tercih edilir.

Aritmetik Birimler

FPGA'larda toplama ve çıkarma işlemleri CLB içindeki elde tasiyici zincirleri (carry chain) ile verimli şekilde gerçekleştirilir. Çarpma işlemleri ise DSP dilimleri kullanir.

```vhdl
-- Toplayıcı
sum <= std_logic_vector(unsigned(a) + unsigned(b));

-- Cikarici
diff <= std_logic_vector(signed(a) - signed(b));

-- Çarpıcı (DSP dilimi kullanir)
product <= std_logic_vector(signed(a) * signed(b));

-- Kaydırma ile 2'nin kuvveti ile çarpma/bölme
doubled <= data(6 downto 0) & '0';       -- 2 ile carp (sola
halved  <= '0' & data(7 downto 1);       -- 2 ile bol (saga k
```

Parametrik Tasarım (Generic ile)

Generic parametreler kullanarak yeniden kullanılabilir bileşenler tasarlanabilir. Bu yaklasim, farklı bit genişliklerinde aynı mantığı kullanmanizi sağlar.

```vhdl
-- Generic N-bit toplayıcı
entity adder_generic is
generic (
    N : integer := 8
);
port (
    a_i   : in  std_logic_vector(N-1 downto 0);
    b_i   : in  std_logic_vector(N-1 downto 0);
    sum_o : out std_logic_vector(N downto 0)  -- N+1 bit (elde dahil)
);
end adder_generic;

architecture Behavioral of adder_generic is
begin
    sum_o <= std_logic_vector(
        resize(unsigned(a_i), N+1) + resize(unsigned(b_i), N+1)
    );
end Behavioral;
```

Bölüm 6

Ardışıl Mantık Tasarımı

Flip-Flop, reset, pipeline ve senkronizasyon

Ardışıl Mantık Tasarımı

Ardisil mantikte çıkış, girişlerin yanı sira önceki duruma da bağlıdır. Saat sinyali ile senkronize çalışır. Flip-Flop'lar veri saklama görevi görür.

Temel Ardisil Tasarım

Aynı mantık fonksiyonunu (y = abc' + a'bc + a'b'c') ardisil olarak yazarsak, rising_edge(clk) kosulu eklenir ve çıkış Flip-Flop üzerinden geçiş yapar:

```vhdl
architecture Behavioral of gate_level_seq is
begin
    process (clk) begin
        if rising_edge(clk) then
            y <= (a and b and (not c)) or
                 ((not a) and b and c) or
                 ((not a) and (not b) and (not c));
        end if;
    end process;
end Behavioral;
```

BİLGİ

Hassasiyet listesinde sadece 'clk' var. Bu, çıkış sinyali 'y' için Flip-Flop çıkartılacağı anlamına gelir. Veri, saatin yükselen kenarinda yakalanir.

Reset Sinyali

Reset sinyali, tasarımı bilinen bir duruma getirmek için kullanılır. İki temel reset türü vardır:

Özellik	Senkron Reset	Asenkron Reset
Çalışması	Sadece saat kenarinda	Saat bağımlı değil, an..

Hassasiyet listesi	Sadece clk	clk ve rst
VHDL kodu	if rising_edge then if..	if rst then ... elsif ..
Avantaj	Daha temiz zamanlama	Aninda tepki

VHDL

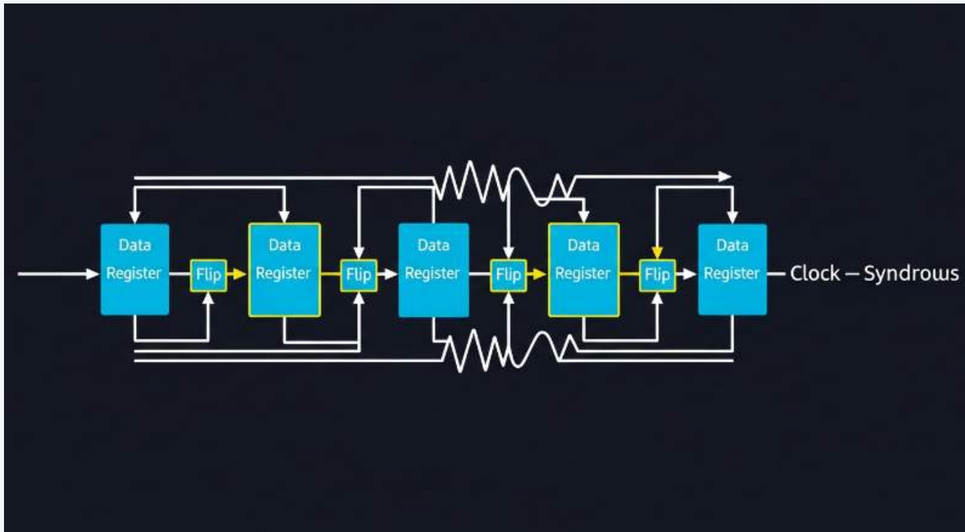
```
-- SENKRON RESET
process (clk) begin
    if rising_edge(clk) then
        if (rst = '1') then
                 y_int <= '1';  -- reset de eri
        else
            y_int <= (a and b and (not c)) or
                     ((not a) and b and c) or
                     ((not a) and (not b) and (not c));
        end if;
    end if;
end process;

-- ASENKRON RESET
process (clk, rst) begin
    if (rst = '1') then
             y_int <= '1';  -- an1nda reset
    elsif rising_edge(clk) then
        y_int <= (a and b and (not c)) or
                 ((not a) and b and c) or
                 ((not a) and (not b) and (not c));
    end if;
end process;
```

İPUCU

SRAM tabanlı FPGA'larda (AMD, Intel) başlangıç değeri vermek için reset sinyali sart değildir. ASIC veya flash tabanlı FPGA'larda (Microchip) reset sinyali gereklidir.

Pipeline (Boru Hatti) Tasarımı

Pipeline, karmaşık mantığı birden fazla aşamaya bölerek her asamayi ayrı bir Flip-Flop ile kayit altina alir. Bu sayede daha yüksek saat frekanslarina ulasilabilir.



Pipeline: Her aşama bir FF ile birbirinden ayrılır, daha yüksek saat frekansı sağlar

```vhdl
signal ff1, ff2, ff3 : std_logic := '0';

process (clk) begin
    if rising_edge(clk) then
        if (rst = '1') then
            y <= '0'; ff1 <= '0'; ff2 <= '0'; ff3 <= '0';
        else
                    ff1 <= (a and b and (not c));         -- 1. a_ama
                    ff2 <= ((not a) and b and c);         -- 1. a_ama
                    ff3 <= ((not a) and (not b) and (not c)); -- 1. a
                    y   <= ff1 or ff2 or ff3;             -- 2. a_am
        end if;
    end if;
end process;
```

2-FF Senkronizasyon

Farklı saat alanlarından (clock domain) gelen sinyalleri güvenli şekilde almak için 2 asamali FF senkronizasyon kullanılır. Bu, metastabilite sorununu önler.

```vhdl
signal ffa : std_logic_vector(1 downto 0) := (others => '0');

process (clk) begin
    if rising_edge(clk) then
            ffa(0) <= a;          -- 1. FF: Asenkron giri_i yakala
        ffa(1) <= ffa(0);   -- 2. FF: Kararli sinyal uret
    end if;
end process;
 -- ffa(1) art1k güvenle kullan1labilir
```

DİKKAT

Metastabilite: Bir FF'in setup veya hold zamaninin ihlal edilmesiyle olusan belirsiz durumdur. Farklı saat alanları arasında veri aktariminda 2-FF veya 3-FF senkronizasyon kullanilmalidir. Veri için çift saatli FIFO veya Gray kodlu sayıcılar tercih edilir.

Sayıcı (Counter) Tasarımı

Sayicilar, ardisil mantık tasarımının en temel yapı taşlarından biridir. Yukari sayıcı, aşağı sayıcı veya yukarı/aşağı sayıcı olarak tasarlanabilir.

```vhdl
-- N-bit yukar1/a_a 1 say1c1
entity counter is
generic (N : integer := 8);
port (
    clk    : in  std_logic;
    rst    : in  std_logic;
    en     : in  std_logic;
       up_down: in  std_logic;  -- '1': yukar1, '0': a_a 1
    count  : out std_logic_vector(N-1 downto 0)
);
end counter;

architecture Behavioral of counter is
    signal cnt : unsigned(N-1 downto 0) := (others => '0');
begin
    process (clk) begin
        if rising_edge(clk) then
            if rst = '1' then
                cnt <= (others => '0');
            elsif en = '1' then
                if up_down = '1' then
                    cnt <= cnt + 1;
                else
                    cnt <= cnt - 1;
                end if;
            end if;
        end if;
    end process;
    count <= std_logic_vector(cnt);
end Behavioral;
```

Kaydırma Yazmacı (Shift Register)

Shift register, verileri bit bit kaydiran ardisil bir yapıdır. Seri-paralel ve paralel-seri dönüşüm, gecikme hattı ve veri senkronizasyonu için kullanılır.

```vhdl
-- 8-bit sola kayd1rma yazmac1 (SIPO: Seri giri_, Paralel ç1k
signal shift_reg : std_logic_vector(7 downto 0) := (others => '0');

process (clk) begin
    if rising_edge(clk) then
        if rst = '1' then
            shift_reg <= (others => '0');
        elsif shift_en = '1' then
            shift_reg <= shift_reg(6 downto 0) & serial_in;
        end if;
    end if;
end process;
parallel_out <= shift_reg;
```

İPUCU

Shift register'lar FPGA'da SRL (Shift Register LUT) olarak da sentezlenebilir. Bu, FF yerine LUT kullanarak kaynak tasarrufu sağlar. Vivado bunu otomatik olarak optimize edebilir.

Saat Etkinleştirme (Clock Enable)

Clock enable, ana saati bölmeden daha düşük frekansta çalışma imkanı sağlar. Saat alanı sayisini artırmadan farklı hızlarda çalışan modüller oluşturabilirsiniz.

```vhdl
-- Clock enable üreteci: Ana saat / N
signal ce_counter : integer range 0 to 99 := 0;
signal clk_en     : std_logic := '0';

process (clk) begin
    if rising_edge(clk) then
        if ce_counter = 99 then
            ce_counter <= 0;
            clk_en <= '1';    -- Her 100 çevrimde bir aktif
        else
            ce_counter <= ce_counter + 1;
            clk_en <= '0';
        end if;
    end if;
end process;

-- Clock enable ile çal1_an modül
process (clk) begin
    if rising_edge(clk) then
        if clk_en = '1' then
                        -- Bu kisim ana saatin 1/100'u h1z1nda çal1_1r
            slow_counter <= slow_counter + 1;
        end if;
    end if;
end process;
```

> **DİKKAT**
>
> Clock enable ile saat bölme arasindaki fark onemlidir! Clock enable tek saat alaninda çalışır (güvenli). Sayıcı ile üretilen saat sinyali ise yeni bir saat alanı oluşturur ve CDC sorunlarina yol açabilir.

Kenar Algılama (Edge Detection)

Bir sinyalin yükselen veya dusen kenarini algılamak için gecikme yazmacı (delay register) kullanılır. Bu teknik buton basma, veri hazır sinyali gibi olayları yakalamak için kullanılır.

```vhdl
signal input_d  : std_logic := '0';  -- Gecikmeli kopya
signal rising   : std_logic;
signal falling  : std_logic;
signal any_edge : std_logic;

process (clk) begin
    if rising_edge(clk) then
        input_d <= input_signal;  -- Bir saat çevrimi gecikme
    end if;
end process;

rising   <= input_signal and (not input_d);      -- Yükselen kenar
falling  <= (not input_signal) and input_d;      -- Dusen kenar
any_edge <= input_signal xor input_d;            -- Her iki kenar
```

Debounce (Buton Titreşim Önleme)

Mekanik butonlar basildiginda kisa sureli titreşimler (bouncing) üretir. Bu titreşimler birden fazla tetiklemeye neden olabilir. Debounce devresi bu sorunu çözer.

```vhdl
-- Basit debounce devresi
signal btn_sync  : std_logic_vector(1 downto 0) := "00";
signal btn_count : integer range 0 to 999999 := 0;
signal btn_clean : std_logic := '0';

process (clk) begin
    if rising_edge(clk) then
             -- 2-FF senkronizasyon (buton asenkron giri_tir)
        btn_sync <= btn_sync(0) & btn_raw;

             -- 10ms sayıcı (100 MHz saat için)
        if btn_sync(1) /= btn_clean then
            if btn_count = 999999 then
                btn_clean <= btn_sync(1);
                btn_count <= 0;
            else
                btn_count <= btn_count + 1;
            end if;
        else
```

```
            btn_count <= 0;
        end if;
    end if;
end process;
```

BİLGİ

Debounce süresi genellikle 10-20 ms arasında seçilir. Bu süre, mekanik butonun titreşim suresinden uzun olmalıdır. Çok kisa süre yanlis tetiklemeye, çok uzun süre ise yavas tepkiye neden olur.

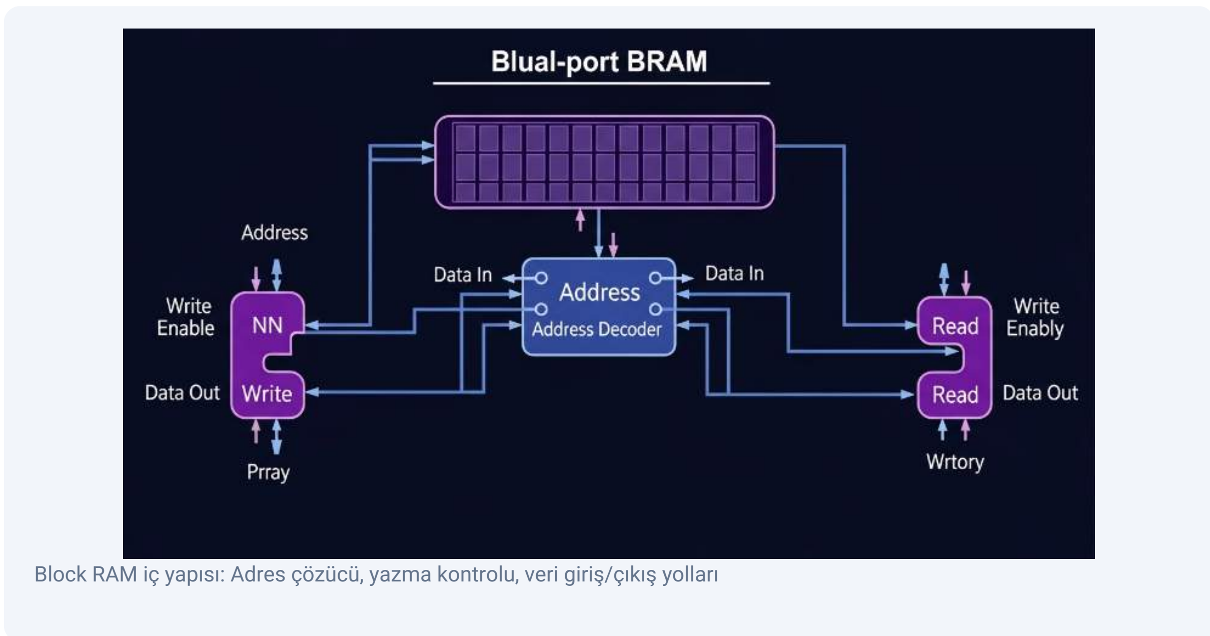
Bölüm 7

Block RAM (BRAM) Tasarımı

BRAM yapısı, tek/çift port, okuma/yazma modları

Block RAM (BRAM) Nedir?

Block RAM, FPGA içinde yerleşik olan özelleştirilmiş bellek bloklaridir. Veri depolama, FIFO kuyruk yapıları ve arama tablolari için kullanılır.



Block RAM iç yapısı: Adres çözücü, yazma kontrolu, veri giriş/çıkış yolları

BRAM Çıkarımı (İnfer)

BRAM çıkarımak için önce 2 boyutlu bir sinyal yapısı tanımlanır. Sonra okuma ve yazma için dijital devre yazılır:

```vhdl
-- 2 boyutlu bellek tipi tanımlama
type t_bram is array (0 to 255) of std_logic_vector(31 downto 0);
signal bram : t_bram;

-- 256 satır, her satır 32-bit genisliginde
```

BRAM Port Yapilari

Yapı	Kısaltma	Açıklama
Tek Portlu	SP	1 adet okuma/yazma portu
Basit Çift Portlu	SDP	1 okuma/yazma + 1 sade..
Gerçek Çift Portlu	TDP	2 adet okuma/yazma portu

Tek Portlu BRAM Örneği

VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity spbram is
port (
    clk    : in  std_logic;
    we_i   : in  std_logic;
    addr_i : in  std_logic_vector(9 downto 0);
    din_i  : in  std_logic_vector(7 downto 0);
    dout_o : out std_logic_vector(7 downto 0)
);
end spbram;

architecture Behavioral of spbram is
type t_ram is array (0 to 2**10-1) of std_logic_vector(7 downto 0);
signal ram : t_ram := (others => (others => '0'));
begin
    process (clk) begin
        if rising_edge(clk) then
            if (we_i = '1') then
                ram(to_integer(unsigned(addr_i))) <= din_i;
            end if;
            dout_o <= ram(to_integer(unsigned(addr_i)));
        end if;
    end process;
end Behavioral;
```

Generic BRAM

Generic parametreler kullanarak genişlik ve derinliği ayarlanabilir BRAM tasarlayabilirsiniz:

```vhdl
entity spbram_gen is
generic (
    WIDTH : integer := 8;
    DEPTH : integer := 14
);
port (
    clk    : in  std_logic;
    we_i   : in  std_logic;
    addr_i : in  std_logic_vector(DEPTH-1 downto 0);
    din_i  : in  std_logic_vector(WIDTH-1 downto 0);
    dout_o : out std_logic_vector(WIDTH-1 downto 0)
);
end spbram_gen;
```

Read-First ve Write-First Modlari

Mod	Davranış
Write-First	Yazilan veri aynı saat çevriminde ..
Read-First	Önceki veri okunur, yeni veri bir ..

BİLGİ

Vivado netlist görünümünde BRAM ilkelini (primitive) bulup Properties sekmesinden WRITE_MODE_A ve WRITE_MODE_B özelliklerini kontrol edebilirsiniz.

Çıkış Yazmaci (Output Register)

AMD FPGA'larda Block RAM'ler çıkış yazmacı ile veya olmadan yapılandırılabilir. Çıkış yazmacı olmadan okuma gecikmesi 1 saat çevrimi, çıkış yazmacı ile 2 saat çevrimidir.

İPUCU

Neden daha fazla gecikme? Ekstra yazmaç, BRAM ile mantık arasında tampon görevi görür. Yüksek hızlı veya derin pipeline tasarımlarda zamanlama kapanmasina (timing closure) yardımcı olur. DO_REG ayari bu ozelligi kontrol eder.

Basit Çift Portlu (SDP) BRAM

SDP BRAM'de bir port sadece yazmak, diger port sadece okumak için kullanılır. Farklı saat alanlariyla kullanılabilir, bu özellik onu CDC uygulamalarında ideal kılar.

```
-- SDP BRAM: Ayri yazma ve okuma portlari
entity sdp_bram is
generic (
    WIDTH : integer := 8;
    DEPTH : integer := 10
);
port (
    -- Yazma portu
    wr_clk  : in  std_logic;
    wr_en   : in  std_logic;
    wr_addr : in  std_logic_vector(DEPTH-1 downto 0);
    wr_data : in  std_logic_vector(WIDTH-1 downto 0);
    -- Okuma portu
    rd_clk  : in  std_logic;
    rd_addr : in  std_logic_vector(DEPTH-1 downto 0);
    rd_data : out std_logic_vector(WIDTH-1 downto 0)
);
end sdp_bram;

architecture Behavioral of sdp_bram is
    type t_ram is array (0 to 2**DEPTH-1)
        of std_logic_vector(WIDTH-1 downto 0);
    signal ram : t_ram := (others => (others => '0'));
begin
    -- Yazma i_lemi
    process (wr_clk) begin
        if rising_edge(wr_clk) then
            if wr_en = '1' then
                ram(to_integer(unsigned(wr_addr))) <= wr_data;
            end if;
        end if;
    end process;

    -- Okuma i_lemi (farkl1 saat alan1 olabilir)
    process (rd_clk) begin
        if rising_edge(rd_clk) then
            rd_data <= ram(to_integer(unsigned(rd_addr)));
        end if;
    end process;
end Behavioral;
```

BRAM Başlangıç Degerleri

BRAM'i başlangıç degerleriyle doldurarak ROM (Read-Only Memory) olarak kullanabilirsiniz. Bu yöntem, sinyal işleme katsayilari veya arama tablolari için idealdir.

```vhdl
-- Sabit degerlerle ROM olu_turma
type t_rom is array (0 to 15) of std_logic_vector(7 downto 0);
signal rom : t_rom := (
    x"00", x"19", x"32", x"4B",
    x"64", x"7D", x"96", x"AF",
    x"C8", x"AF", x"96", x"7D",
    x"64", x"4B", x"32", x"19"
);

-- Dosyadan okuma ile BRAM ba_langıç (simülasyon ve sentez)
impure function init_ram_from_file(filename : string)
    return t_ram is
    file ram_file : text open read_mode is filename;
    variable line_v : line;
    variable ram_v  : t_ram;
begin
    for i in t_ram'range loop
        readline(ram_file, line_v);
        hread(line_v, ram_v(i));  -- hex formatinda oku
    end loop;
    return ram_v;
end function;

signal ram : t_ram := init_ram_from_file("init_data.hex");
```

Dağıtılmış RAM vs Block RAM

Özellik	Dağıtılmış RAM (LUT RAM)	Block RAM (BRAM)
Kaynak	LUT'lardan oluşturulur	Özel BRAM blokları
Kapasite	Küçük (< 256 bit)	Büyük (18Kb / 36Kb)
Okuma	Asenkron (kombinasyonel)	Senkron (1-2 saat çevr..
Gecikme	Çok düşük	1-2 saat çevrimi
Kullanım	Küçük FIFO, register d..	Büyük bellek, video ta..
Sentez	Otomatik (küçük boyutl..	Otomatik veya IP ile

İPUCU

Vivado, küçük bellekleri otomatik olarak dağıtılmış RAM, büyük bellekleri BRAM olarak sentezler. ram_style attribute'u ile bunu kontrol edebilirsiniz: attribute ram_style : string; attribute ram_style of ram : signal is "block"; (veya "distributed")

FIFO (First In, First Out) Yapısı

FIFO, verilerin sırayla yazilip sırayla okundugu bir bellek yapıdır. Farklı hızlarda çalışan modüller

arasında veri tamponu olarak kullanılır.

- Senkron FIFO: Tek saat alaninda çalışır. Yazma ve okuma aynı saatle senkronize.
- Asenkron FIFO: İki farklı saat alaninda çalışır. CDC için güvenli veri aktarimi sağlar.
- Temel sinyaller: wr_en (yazma), rd_en (okuma), full (dolu), empty (bos), data_in, data_out.
- Gray kodlu sayıcılar: Asenkron FIFO'larda adres sayicilari Gray kodda tutularak CDC güvenliği sağlanır.

> **BİLGİ**
>
> FIFO tasarımı karmaşık olabilir. Vivado'nun FIFO Generator IP'si, senkron ve asenkron FIFO'lari otomatik olarak oluşturur. Özel ihtiyaclariniz yoksa IP kullanmanız önerilir.

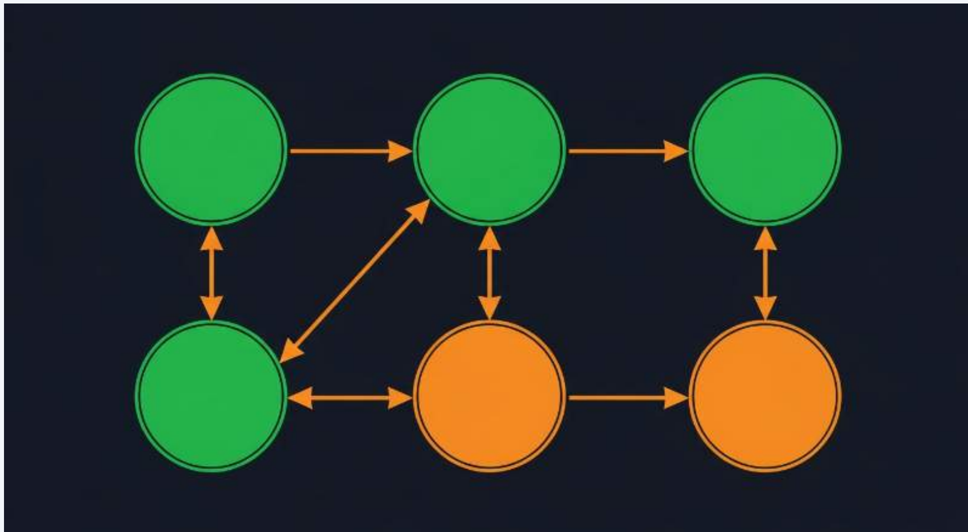
Bölüm 8

Durum Makinesi (FSM) Tasarımı

Sonlu durum makinesi, kodlama ve geçişler

Sonlu Durum Makinesi (FSM) Nedir?

FSM (Finite State Machine), belirli sayida durumu olan ve belirli koşullara göre durumlar arasi geçiş yapan dijital devredir. Kontrol mantığı, protokol yönetimi ve sira tabanlı işlemler için kullanılır.



5 durumlu FSM örneği: Durumlar arasi geçişler koşullara bağlıdır

FSM VHDL Örneği

Asagidaki ornekte 5 durumlu (S0-S4) bir FSM tasarımı gösterilmiştir. Durumlar arasi geçişler giriş kosullarina bağlıdır:

```vhdl
entity fsm is
port (
    clk : in  std_logic;
    rst : in  std_logic;
    a_i : in  std_logic;
    b_i : in  std_logic;
    c_o : out std_logic;
    d_o : out std_logic
);
end fsm;

architecture Behavioral of fsm is
type t_state is (S0, S1, S2, S3, S4);
signal state : t_state := S0;
begin
    process (clk) begin
        if rising_edge(clk) then
            if (rst = '1') then
                state <= S0;
                c_o <= '0'; d_o <= '0';
            else
                                -- varsayilan ç1k1_ de erleri
                c_o <= '0';
                d_o <= '0';
                case state is
                    when S0 =>
                        if (a_i = b_i) then
                            state <= S1;
                        elsif (a_i = '0') then
                            state <= S2;
                            c_o   <= '1';
                        end if;
                    when S1 =>
                        c_o <= '1';
                        if (a_i = '1') then
                            state <= S0;
                        elsif (a_i /= b_i) then
                            state <= S2;
                        elsif (b_i = '1') then
                            state <= S4;
                        end if;
                    when S2 =>
                        d_o <= '1';
                        if (b_i = '0') then
                            state <= S3;
                        end if;
                    when S3 =>
                        state <= S0;
                        c_o <= '1'; d_o <= '1';
                    when S4 =>
                        d_o <= '1';
                        if (a_i = '0' and b_i = '0') then
                            state <= S1;
                        elsif (a_i = '1' and b_i = '0') then
                            state <= S0;
                        end if;
                    when others => null;
                end case;
            end if;
```

```
        end if;
    end process;
end Behavioral;
```

İPUCU

Process başında çıkışları varsayilan degerlere sıfırlayın ('0'). Boylece sadece gerekli durumlarda çıkış değerini degistirmeniz yeterli olur. Bu yöntem kodu daha temiz yapar.

FSM Kodlama Yontemleri

FSM durumlarini kodlamak için farklı yontemler kullanılabilir. Vivado'da sentez ayarlarindan seçim yapabilirsiniz:

Yontem	5 Durum Örneği	FF Sayisi
Sırasal (Sequential)	000, 001, 010, 011, 100	3 bit
Bir-Sicak (One-Hot)	00001, 00010, 00100, 0..	5 bit

- One-hot avantaji: Gecisler tek bit degisikligine bağlı, kod çözme daha basit.
- One-hot avantaji: Daha hızlı zamanlama, sonraki durum mantığı daha az kombinasyonel gecikme.
- One-hot avantaji: Gecersiz durum gecislerini tespit etmek kolay (uzay uygulamaları için önemli).
- Sırasal kodlama: Daha az FF kullanir, küçük tasarımlarda uygundur.

Moore ve Mealy Karsilastirmasi

FSM'ler çıkışların nasıl uretildigine göre iki ana kategoriye ayrılır:

Özellik	Moore Makinesi	Mealy Makinesi
Çıkış bagimliligi	Sadece mevcut duruma	Mevcut durum + girişler
Çıkış zamanlama	Durum geçişinden sonra	Giriş degisikligiyle a..
Durum sayısı	Genellikle daha fazla	Daha az durum yeterli
Tepki süresi	1 saat çevrimi gecikme	Aninda tepki (kombinas..
Kararlillik	Daha kararlı çıkışlar	Glitch'e yatkin olabilir

BİLGİ

Pratikte çoğu tasarım Moore ve Mealy'nin karışımı olarak yazılır. Kayitli (registered) çıkışlar Moore benzeri davranır, kombinasyonel çıkışlar Mealy benzeri davranır.

İki-Process FSM Stili

FSM tasarımı tek process yerine iki process ile yazilabilir: biri durum geçişleri (ardisil), digeri çıkış mantığı (kombinasyonel). Bu ayrim büyük FSM'lerde kodu daha okunabilir kılar.

```vhdl
architecture Behavioral of fsm_two_proc is
    type t_state is (IDLE, RUN, DONE);
    signal state, next_state : t_state := IDLE;
begin
    -- Process 1: Durum yazmacı (ardisil)
    REG : process (clk) begin
        if rising_edge(clk) then
            if rst = '1' then
                state <= IDLE;
            else
                state <= next_state;
            end if;
        end if;
    end process;

    -- Process 2: Sonraki durum + çıkış mantığı (kombinasyonel)
    COMB : process (state, start, data_valid) begin
        -- Varsayilan değerler
        next_state <= state;
        busy <= '0';
        done <= '0';

        case state is
            when IDLE =>
                if start = '1' then
                    next_state <= RUN;
                end if;
            when RUN =>
                busy <= '1';
                if data_valid = '1' then
                    next_state <= DONE;
                end if;
            when DONE =>
                done <= '1';
                next_state <= IDLE;
            when others =>
                next_state <= IDLE;
        end case;
    end process;
end Behavioral;
```

Guvenli FSM Tasarımı

Gerçek uygulamalarda gecersiz durumlara dusme riski vardır (radyasyon, gürültü vb.). Guvenli FSM tasarımı için önlemler alinmalidir.

- when others dalini her zaman ekleyin ve bilinen bir duruma (genellikle IDLE/RESET) yönlendirin.

- One-hot kodlamada gecersiz durum kontrolu ekleyin.
- Kritik uygulamalarda watchdog sayıcı kullanın: Belirli süre içerisinde beklenen geçiş olmazsa RESET'e donun.
- Vivado'da FSM extraction'i kapatirsaniz kendi kodlamanizi kontrol edebilirsiniz.

```vhdl
-- Watchdog ile güvenli FSM
signal watchdog : integer range 0 to 9999 := 0;

process (clk) begin
    if rising_edge(clk) then
        if rst = '1' or state = IDLE then
            watchdog <= 0;
        else
            if watchdog = 9999 then
                state <= IDLE;  -- Zaman asimi, güvenli duruma don
                watchdog <= 0;
            else
                watchdog <= watchdog + 1;
            end if;
        end if;
    end if;
end process;
```

Pratik FSM Örneği: UART Verici

UART (Universal Asynchronous Receiver-Transmitter) verici, tipik bir FSM uygulamasıdır. IDLE -> START_BIT -> DATA_BITS -> STOP_BIT durumlarini içerir.

```vhdl
type t_uart_state is (IDLE, START_BIT, DATA_BITS, STOP_BIT);
signal uart_state : t_uart_state := IDLE;
signal bit_index  : integer range 0 to 7 := 0;
signal tx_data    : std_logic_vector(7 downto 0);

process (clk) begin
    if rising_edge(clk) then
        if rst = '1' then
            uart_state <= IDLE;
            tx <= '1';  -- Bos hat '1' seviyesinde
        else
            case uart_state is
                when IDLE =>
                    tx <= '1';
                    if tx_start = '1' then
                        tx_data <= data_in;
                        uart_state <= START_BIT;
                    end if;
                when START_BIT =>
                                    t x  < =  ' 0 ' ;    - -  B a _ l a n g ı ç  b i t i :  ' 0 '
                    if baud_tick = '1' then
                        uart_state <= DATA_BITS;
                        bit_index <= 0;
                    end if;
                when DATA_BITS =>
```

```
                tx <= tx_data(bit_index);
                if baud_tick = '1' then
                    if bit_index = 7 then
                        uart_state <= STOP_BIT;
                    else
                        bit_index <= bit_index + 1;
                    end if;
                end if;
            when STOP_BIT =>
                tx <= '1';  -- Biti_ biti: '1'
                if baud_tick = '1' then
                    uart_state <= IDLE;
                end if;
        end case;
    end if;
  end if;
end process;
```

İPUCU

UART verici örneği, FSM'nin gerçek dünyada nasıl kullanildigini gösterir. Her durum belirli bir görevi yerine getirir ve belirli koşullar altında bir sonraki duruma gecer. baud_tick sinyali, baud rate hızında darbe üreten ayrı bir sayıcıdan gelir.

Bölüm 9

Sık Karşılaşılan Sorular ve Kaynaklar

Temel sorular, cevaplar ve referanslar

FPGA ve VHDL - Sık Karşılaşılan Sorular

Bu bölümde, FPGA alaninda sık karşılaşılan temel sorular ve cevaplari yer almaktadir. Sorular temel kavramlardan ileri konulara kadar geniş bir yelpazeyi kapsar.

Soru 1: FPGA'nın İçinde Ne Var?

Cevap: Bir FPGA'nın içinde bes temel yapı blogu bulunur:

- Programlanabilir Mantık Elemanlari (Programmable Logic Elements): CLB (Karmaşık Mantık Blogu) içinde LUT (Arama Tablosu) ve FF (Flip-Flop) bulunur. LUT kombinasyonel mantığı, FF ise ardisil mantığı (veri saklama) gerçekleştirir. Bunlar FPGA'nın temel işlem birimleridir.
- Programlanabilir Bağlantı Kaynakları (Programmable Routing Resources): Yonlendiriciler (routers) ve anahtarlar (switches) içerir. CLB'ler arasindaki baglantiyi sağlar. FPGA alanının yaklaşık %60-70'ini kaplayanlar bu yollardir.
- G/C Blokları (I/O Blocks): Yapılandırılabilir giriş/çıkış birimleridir. input (giriş), output (çıkış) veya inout (çift yönlü) olarak ayarlanabilir. LVDS (Low-Voltage Differential Signaling - Düşük Voltajli Diferansiyel Sinyalleme), LVCMOS (Low-Voltage Complementary Metal-Oxide Semiconductor), SSTL (Stub Series Terminated Logic - DDR bellek arayüzü standardı), HSTL (High-Speed Transceiver Logic - Yüksek hızlı alici-verici mantığı) gibi çeşitli elektriksel standartları destekler.
- Block RAM (BRAM): Sertleşmiş (hardened), optimize edilmis bellek kaynaklaridir. Her biri genellikle birkaç kB kapasitededir (örneğin Xilinx 7 serisi: 36 Kb / blok). Veri tamponu, FIFO, arama tablosu gibi amaclarla kullanılır.
- DSP Blokları (DSP Slices): Sertleşmiş, optimize edilmis aritmetik kaynaklaridir. Özellikle çarpma işlemlerini verimli şekilde gerçekleştirir. İçinde çarpıcı (multiplier), toplayıcı (accumulator) ve on-toplayıcı (pre-adder) bulunur. Dijital sinyal işleme, filtre ve matematiksel hesaplamalar için kullanılır.

İPUCU

Sertleşmiş (hardened) bloklar, FPGA'nın silikon üzerine sabit olarak üretilen özel birimlerdir. Programlanabilir mantiktan (LUT) çok daha hızlı ve verimlidir. BRAM ve DSP blokları bu kategoridedir. Karsiliginda CLB'ler 'yumusak' (soft) mantık olarak adlandırılır.

Soru 2: CLB'nin İçinde Ne Var?

Cevap: CLB (Complex Logic Block - Karmaşık Mantık Blogu), FPGA'nın temel işlem birimidir. İçinde dort ana bileşen bulunur:

- LUT (Look-Up Table - Arama Tablosu): Kombinasyonel mantık fonksiyonlarını gerçekleştirir. Örneğin bir 4-LUT, 4 giriş (in0-in3) alir ve farklı çıkışlar üretebilir: LUT4_out (4 girişli fonksiyon), LUT3_out[0] ve LUT3_out[1] (3 girişli fonksiyonlar), LUT2_out[0] ve LUT2_out[1] (2 girişli fonksiyonlar). Boylece tek bir LUT birden fazla küçük fonksiyonu aynı anda gerceklestirebilir.
- FF (Flip-Flop): Veri saklama elemanlaridir. CLK (saat) sinyali ile senkronize çalışır. Reset, regin (register input) ve regout (register output) sinyalleri ile kontrol edilir. Her CLB içinde birden fazla FF bulunur ve her biri bir bit veri saklar.
- Hızlı Elde Tasiyicilari (Fast Carry Generators): Toplama ve çıkarma işlemlerinde elde (carry) sinyalinin hızlı iletilmesini sağlar. cin (carry in) girişinden cout (carry out) çıkışına özel kablolu yollarla bağlanır. Bu yollar programlanabilir bağlantı yollarindan çok daha hızlıdır.
- Multiplexer'lar (MUX): CLB içinde çok sayida MUX bulunur. Bu MUX'lar LUT çıkışlarını, FF girişlerini ve genel çıkışları (out[0], out[1]) yönlendirmek için kullanılır. Ayrica scin/scout (scan chain) sinyalleri ile test ve debug amacli tarama zinciri (scan chain) oluşturulabilir.

BİLGİ

CLB içindeki MUX'lar sayesinde LUT çıkışı doğrudan çıkışa (kombinasyonel) veya FF üzerinden çıkışa (ardisil) yönlendirilebilir. Bu esneklik, aynı CLB'nin hem kombinasyonel hem ardisil mantık için kullanılmasını sağlar.

Soru 3: LUT'un Mimarisi Nasıl ve Nasıl Çalışır?

Cevap: LUT (Look-Up Table - Arama Tablosu), FPGA'daki mantığın temel yapı taşıdır. Tüm olası giriş kombinasyonlari için çıkış değerlerini saklayarak kombinasyonel mantık fonksiyonlarını gerçekleştirir. Bir gerceklik tablosu (truth table) gibi çalışır.

İç yapısı, küçük bir SRAM (Static RAM - Statik Rastgele Erisimli Bellek) belleği olarak düşünülebilir. N girişli bir LUT, 2^N adet SRAM hücresi (bitcell) içerir. Örneğin LUT4 (4 girişli LUT) 16 SRAM hücresi gerektirir, LUT6 (6 girişli LUT) ise 64 SRAM hücresi gerektirir.

- SRAM Hucreleri (Bitcells): Her hucre bir bit veri saklar. Bu değerler FPGA yapılandırıldığında (bitstream yüklendiğinde) programlanır. Her hucre wordline (söz hattı) ve bitline (bit hattı) ile erisime açılır.
- Multiplexer (MUX): Giriş sinyalleri (A, B, C, D...) multiplexer'in seçim girişleri olarak kullanılır.

Örneğin 4 girişli LUT'ta 16:1 MUX, giriş kombinasyonuna göre ilgili SRAM hücresinin değerini çıkışa yönlendirir.

- Calisma Prensibi: Giriş değerleri bir adres oluşturur. Bu adres, SRAM bellegindeki ilgili konumu seçer. O konumdaki değer çıkışa (Y) aktarılır. Boylece herhangi bir kombinasyonel mantık fonksiyonu, LUT'a doğru değerler yuklenerek gerçekleştirilebilir.

Örnek: Tam toplayıcı (full adder) fonksiyonu bir LUT içinde gerçekleştirilebilir. A, B ve Cin (elde girişi) olmak üzere 3 giriş alinir, S (toplam) ve Cout (elde çıkışı) üretilir. Gerçeklik tablosundaki her satır bir SRAM hucresine karşılık gelir.

A	B	Cin	S (Toplam)	Cout (Elde)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

İPUCU

Modern FPGA'larda genellikle LUT6 (6 girişli) kullanılır. Bir LUT6, iki ayrı LUT5 fonksiyonu olarak veya tek bir 6 girişli fonksiyon olarak yapılandırılabilir. Bu esneklik, kaynak kullanimini optimize eder.

Soru 4: Sentez Çıktısı Nedir?

Cevap: Sentez çıktısı, hedef FPGA'nın temel bileşenleri (LUT, FF vb.) ile ifade edilen teknoloji haritalanmış netlist'tir.

Soru 5: Son Atama Kurali

```vhdl
process (clk) begin
    if (rising_edge(clk)) then
        if sel = '0' then
            a <= b + c;
        else
            a <= b - c;
        end if;
        if (op) then
            a <= x"03";    -- Bu son atama, önceki atamayl ez
        end if;
    end if;
end process;
```

BİLGİ

sel=1, op=true durumunda cevap: a = x"03". Process içinde son atama geçerlidir.

Soru 6: Coklu Eş Zamanlı Atama Hatasi

```vhdl
a <= b + c;
a <= b - c;   -- HATA: Aynl sinyale iki e_ zamanll atama
```

Cevap: Sentez hatası! 'coklu sürücü' (multi-driven net) oluşur.

Soru 7: when/else Yapısı

```vhdl
-- Ko_ullu sinyal atamasl
a <= b + c when sel = '0' else b - c;

-- Esdegeri:
process (b,c,sel) begin
    if (sel = '0') then a <= b + c;
    else a <= b - c;
    end if;
end process;
```

Soru 8: İstenmeyen Latch Problemi

Kombinasyonel process'te if ifadesinde else dali yoksa ve varsayilan atama yoksa latch çıkar. Vivado uyarı verir ama sentezler.

```vhdl
-- HATALI: Latch çıkar!
process (sel,b,c) begin
    if (sel = '0') then
        a <= b + c;
    end if;    -- else yok! a eski de erini koruyor -> latch
end process;

-- DOGRU: Varsayilan atama veya else ekleyin
process (sel,b,c) begin
    a <= b - c;    -- varsayilan de er
    if (sel = '0') then
        a <= b + c;
    end if;
end process;
```

Soru 9: Hassasiyet Listesi Eksikligi

Cevap: Kombinasyonel process'te tüm okunan sinyaller hassasiyet listesinde olmalıdır. Eksikse simülasyon yanlis sonuç üretir, ancak sentezlenen donanım yine de doğru çalışır.

Soru 10: Signal ve Variable Farkı

Cevap: Variable ':=' ile atanır ve anında güncellenir. Signal '<=' ile atanır ve process sonunda güncellenir. Variable yereldir, signal globaldir.

Soru 11: Ardisil Process'te Sira Önemli mi?

```vhdl
-- PROCESS1 ve PROCESS2 aynı devreyi çıkarır!
PROCESS1 : process (clk) begin       PROCESS2 : process (clk) begin
if rising_edge(clk) then             if rising_edge(clk) then
    a <= b xor c;                        c <= b xor a;
    b <= a xor c;                        b <= a xor c;
    c <= b xor a;                        a <= b xor c;
end if;                               end if;
end process;                          end process;
```

Cevap: Sinyal guncellemeleri process sonunda olduğu için sira önemli değildir. Her iki process da aynı devreyi çıkarır.

Soru 12: Metastabilite Nedir?

Cevap: FF'in setup/hold zamaninin ihlali ile olusan belirsiz durumdur. Önleme: Asenkron girişler için 2-3 asamali FF senkronizasyon. Kontrol sinyalleri için handshake. Veri için çift saatli FIFO kullanılır.

Soru 13: FPGA Elde Tasiyici Yapısı

Cevap: FPGA'lar toplama/çıkarma işlemlerinin gecikmesini azaltmak için özel elde tasiyici zincirleri (carry chain) kullanir. Bunlar CLB içindeki kablolu yollardir.

Soru 14: FSM'de Her İki Kosul Dogru Olursa?

```
VHDL
when S0 =>
    if (cond1) then
        state <= S1;            -- cond1 do ru ise burasi çal1_1
    elsif (cond2) then
        state <= S2;            -- cond2 de do ru olsa bile bura
    end if;
```

Cevap: if/elsif yapısında ilk doğru koşul çalışır, diger dallar atlanır. state = S1 olur.

Soru 15: FPGA ve ASIC Arasindaki Fark Nedir?

Cevap: FPGA yeniden programlanabilirken ASIC (Application Specific IC) üretimden sonra degistirilemez. FPGA geliştirme süresi kisadir ama birim maliyeti yuksektir. ASIC üretim maliyeti dusuktur ama geliştirme süresi ve NRE (Non-Recurring Engineering) maliyeti çok yuksektir. Prototipleme ve düşük hacimli üretim için FPGA, yüksek hacimli üretim için ASIC tercih edilir.

Soru 16: Setup ve Hold Zamani Nedir?

Cevap: Setup zamani, verinin saat kenarindann ONCE kararlı olmasi gereken minimum suredir. Hold zamani, verinin saat kenarından SONRA kararlı kalmasi gereken minimum suredir. Bu surelerin ihlali metastabiliteye yol açar.

Soru 17: Clock Domain Crossing (CDC) Nedir?

Cevap: Farklı saat alanları arasında veri aktarimidir. Tek bit kontrol sinyalleri için 2-FF senkronizasyon, coklu bit veri için asenkron FIFO veya handshake protokolleri kullanılır. Gray kodlu sayıcılar CDC'de güvenli geçiş sağlar.

Soru 18: Timing Closure Nedir?

Cevap: Tasarimin tüm zamanlama gereksinimlerini karsiladigi durumudur. Setup ve hold ihlali yoksa 'timing met' denir. Timing closure saglanamazsa: pipeline ekleme, mantık basitleştirme, saat frekansını düşürme veya fiziksel kısıtlamalar ekleme yöntemleri kullanılır.

Soru 19: FPGA'da Kaynak Paylasimi (Resource Sharing) Nedir?

Cevap: Aynı donanım birimini (örneğin bir çarpıcı) farklı zaman dilimlerinde farklı işlemler için kullanmaktir. Bu yöntem alan tasarrufu sağlar ama kontrol mantığı gerektirir. Örneğin bir DSP dilimini bir saat çevriminde A*B, sonraki çevrimde C*D için kullanabilirsiniz.

Soru 20: Senkron ve Asenkron Tasarım Farkı

Cevap: Senkron tasarımda tüm işlemler tek bir saat sinyaline bağlıdır. Asenkron tasarımda saat sinyali yoktur, giriş degisikliklerinde çıkış hemen değişir. FPGA tasarımında senkron tasarım tercih edilir cunku: zamanlama analizi kolaydir, metastabilite riski azdir ve araclar senkron tasarım için optimize edilmistir.

Soru 21: Generic Nedir ve Neden Kullanilir?

Cevap: Generic, VHDL'de bileşen parametrelerini dinamik yapmak için kullanılır. Örneğin veri genişliği, bellek derinliği gibi değerleri generic yaparak aynı kodu farklı konfigurasyonlarda yeniden kullanabilirsiniz. Bu, tasarımı modular ve esnek kılar.

Soru 22: FPGA'da Guc Tuketimi Nasıl Azaltilir?

- Saat kapatma (clock gating): Kullanilmayan blokların saatini kapatma.
- Düşük saat frekansı: Gerekenden fazla hiz kullanmama.
- Pipeline dengeleme: Gereksiz mantık katmanlarini azaltma.
- Düşük voltajli FPGA aileleri secme (örneğin Lattice iCE40).
- Block RAM kullanma: Dagilmis RAM yerine BRAM tercih etme.

Soru 23: FIFO Ne Zaman Kullanilir?

Cevap: FIFO (First In, First Out) su durumlarda kullanılır: 1) Farklı saat alanlarinda çalışan modüller arasi veri aktarimi (asenkron FIFO). 2) Farklı hızlarda veri üreten ve tuketen modüller arasi tampon. 3) Paket tabanlı veri işleme sistemlerinde geçici depolama. 4) DMA (Direct Memory Access) işlemlerinde burst veri transferi.

Soru 24: FPGA'da Debugging Nasıl Yapilir?

- ILA (Integrated Logic Analyzer): FPGA içinde cip içi mantık analizoru. Gerçek zamanli sinyal gozlemi.
- VIO (Virtual Input/Output): Yazilim üzerinden FPGA'ya giriş verme ve çıkış okuma.

- JTAG/ChipScope: Donanım debug arayüzü ile iç sinyalleri inceleme.
- LED ve GPIO: Basit debug için durum sinyallerini LED'lere baglama.
- Simülasyon: En etkili debug yöntemi. Hatayi önce simülasyonda yakalayip duzeltmek çok daha hızlıdır.

Soru 25: Partial Reconfiguration Nedir?

Cevap: Kismi yeniden yapılandırma, FPGA'nın bir bolumunu çalışma sırasında degistirirken geri kalan kismin calismaya devam etmesidir. Avantajlari: 1) Alan tasarrufu - farklı islevler aynı bolgeyi paylasmabilir. 2) Sistem kesintisiz güncelleme. 3) Guc tasarrufu - kullanilmayan bolgeler kapatilabilir. AMD Vivado bu ozelligi 'Dynamic Function eXchange (DFX)' olarak adlandirir.

Kaynaklar ve Referanslar

Resmi Belgeler

- AMD/Xilinx UG953: 7 Series FPGA and Zynq 7000 SoC Libraries Guide
- IEEE Standard 1076 - VHDL Language Reference Manual
- IEEE Standard 1164 - Multivalue Logic System for VHDL

FPGA Üretici Kaynakları

- AMD/Xilinx Vivado Design Suite (xilinx.com)
- Intel/Altera Quartus Prime (intel.com)
- Lattice Diamond (latticesemi.com)
- Gowin EDA (gowinsemi.com)

BİLGİ

Bu kitap egitim amacli hazirlanmistir. Icerik, FPGA ve VHDL konularinda profesyonel deneyim ve endustri kaynaklarindan derlenerek olusturulmustur.