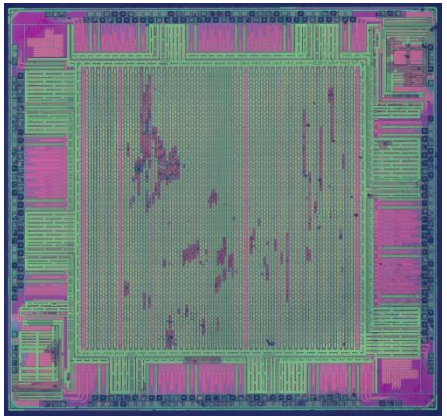
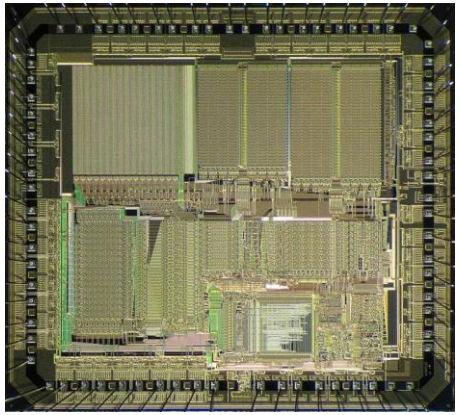
İçindekiler

- Sinyal İşleme Donanımları
- FPGA
- FPGA ile Tasarım
- VHDL
- Uygulama Gerçekleştirme
- Kart Üzerinde Çalıştırma

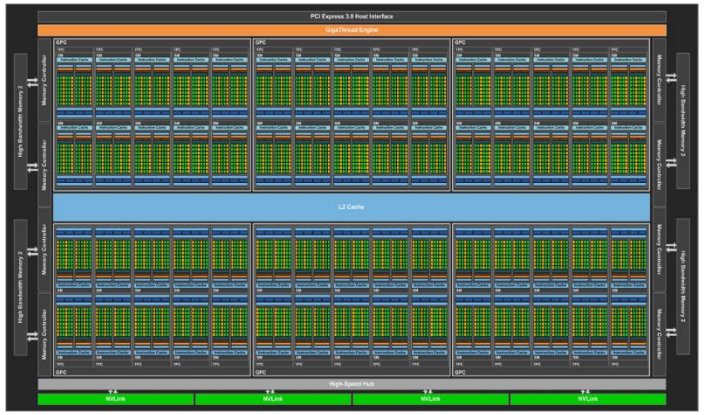
Sinyal İşleme Donanımları



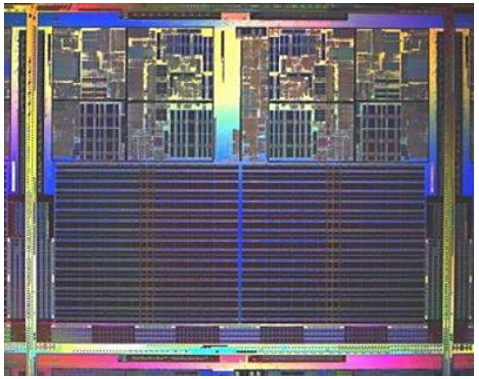
FPGA



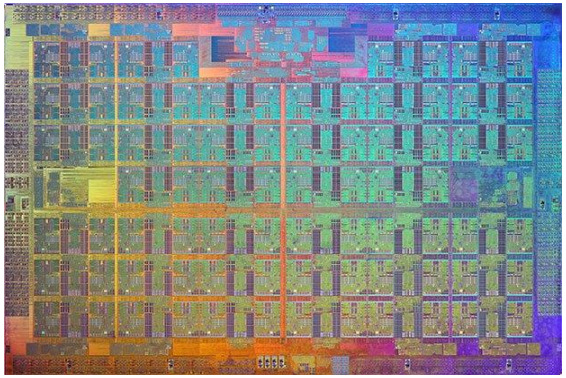
DSP



GPU



CPU



Intel Xeon Phi

Sinyal İşleme Donanımları

Hangisini Seçmeliyim?

	CPU	Phi	GPU	DSP	FPGA
İşlem Gücü	Orta	Yüksek	Yüksek	Orta	Yüksek
Güç Tüketimi	Yüksek	Yüksek	Yüksek	Orta	Düşük
Latency	Orta	Yüksek	Yüksek	Düşük	Düşük
Paralellik	Orta	Yüksek	Yüksek	Orta	Yüksek

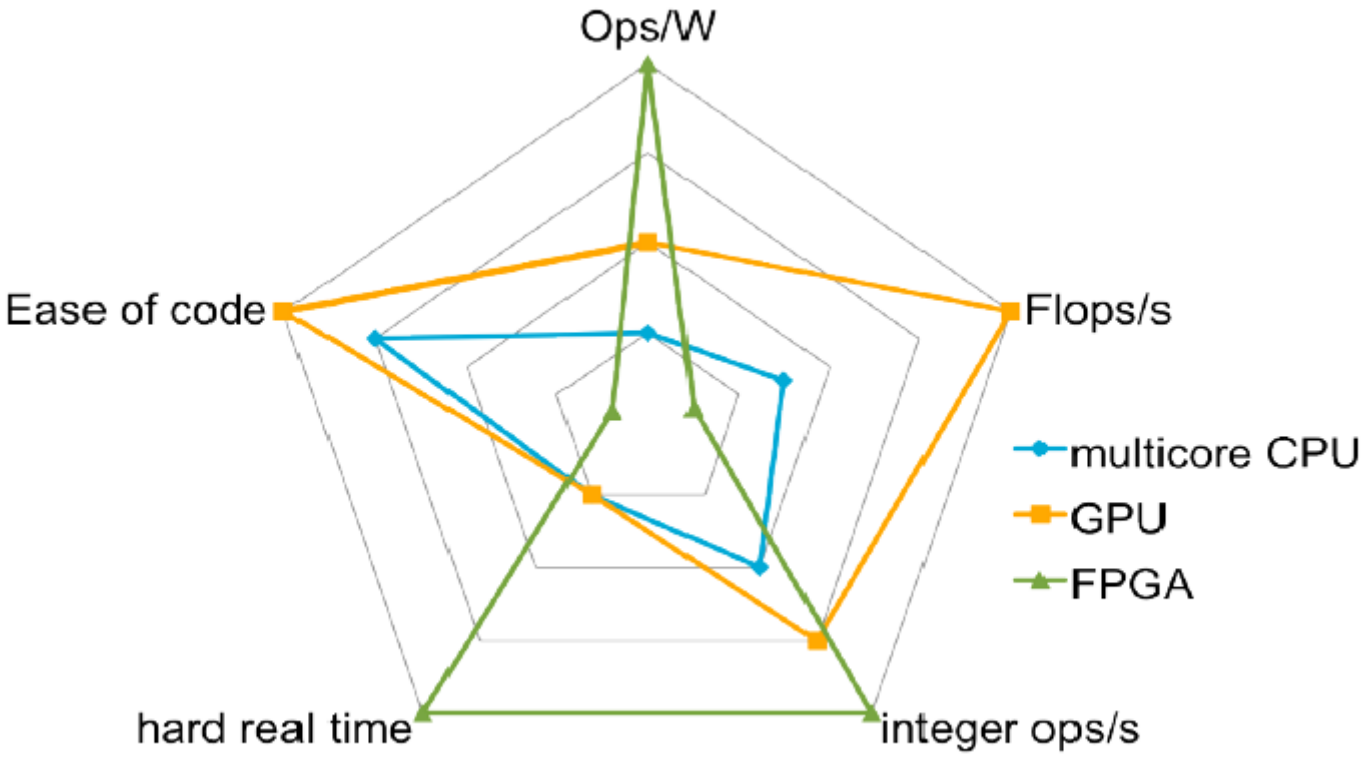
Hangisini Seçmeliyim?

İş Yükü	CPU	Phi	GPU	DSP	FPGA
Sıralı	Yüksek	Düşük	Düşük	Düşük	Düşük
Tekrarlanan	Yüksek	Düşük	Düşük	Orta	Orta
Paralel	Orta	Yüksek	Yüksek	Yüksek	Yüksek
Hafıza Yoğun	Yüksek	Orta	Orta	Düşük	Düşük

	CPU	Phi	GPU	DSP	FPGA
Uygulama Geliştirme	Yüksek	Orta	Orta	Orta	Düşük

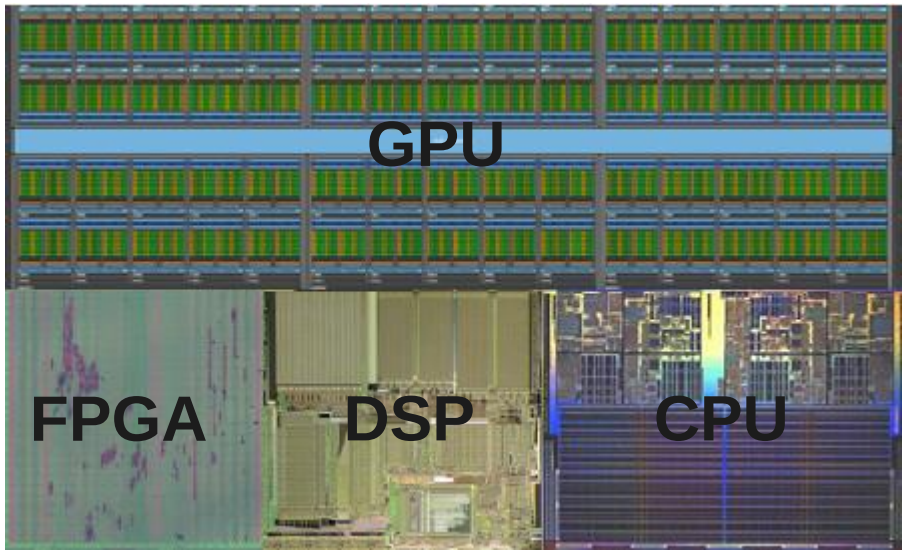
FPGA Tabanlı Sinyal ve Görüntü İşleme

Hangisini Seçmeliyim?



Hangisini Seçmeliyim?

Hibrit bir yapı?



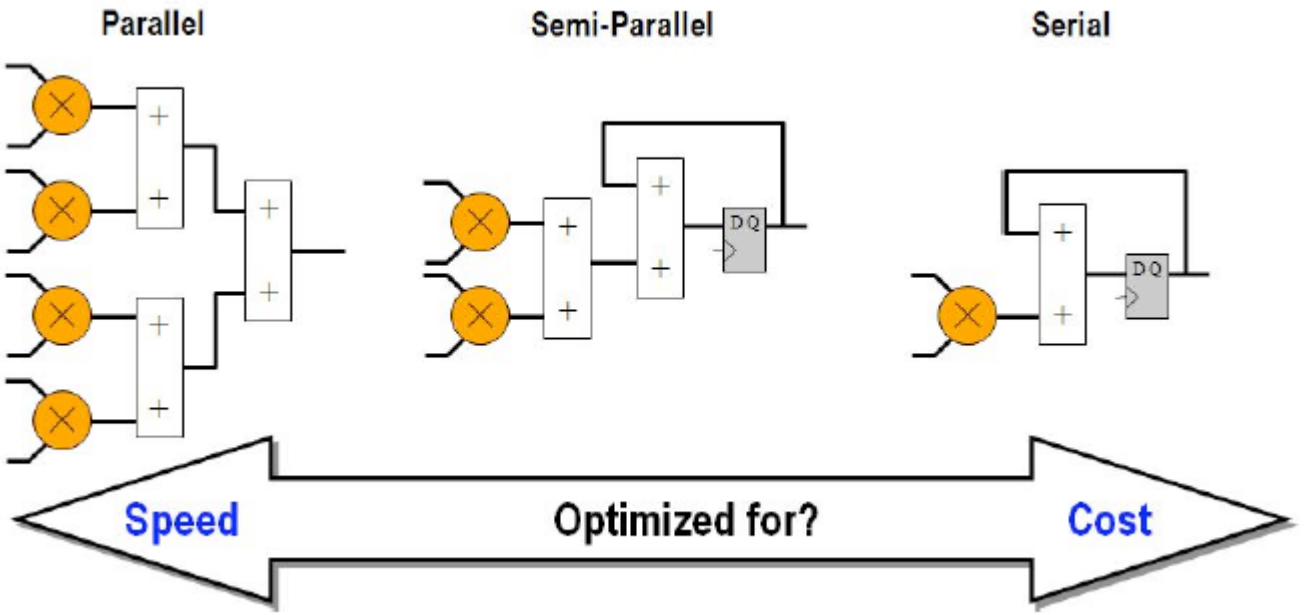
Hangisini Seçmeliyim?

Hibrit Yapılardaki Problemler?

- Donanım – Yazılım ayrışımı
- Algoritmaların Paralelleştirilmesi
- Hafıza birimine Ulaşım
- Kullanıcı Arayüzleri
- Senkronizasyon
- Maliyet

Neden FPGA?

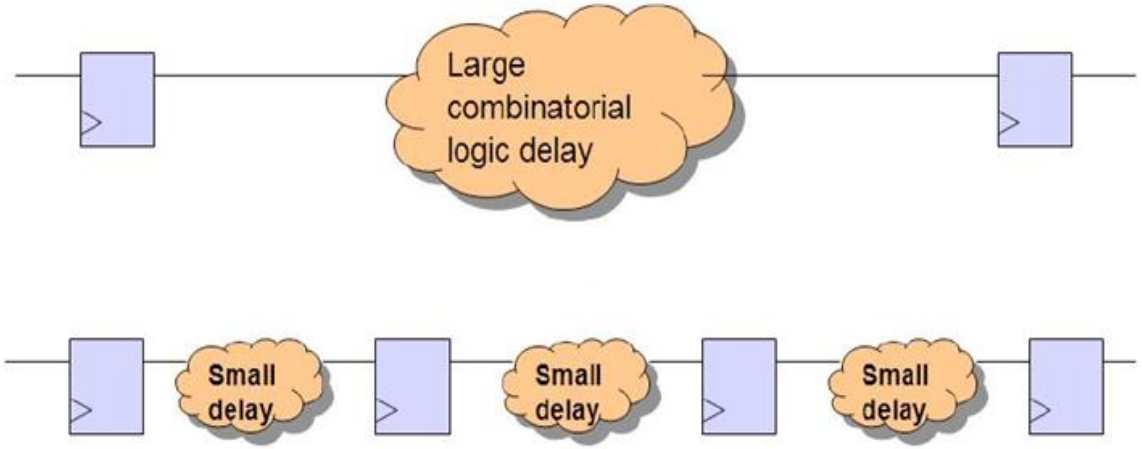
Hız-Alan-Güç



Neden FPGA?

Pipelining

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity normal is
port (Clk : in std_logic;
        data_in : in integer;
        data_out : out integer;
        a,b,c : in integer
      );
end normal;

architecture Behavioral of normal is

signal data,result : integer := 0;

begin

data <= data_in;
data_out <= result;

--process for calcultation of the equation.
PROCESS(Clk,a,b,c,data)
BEGIN
    if(rising_edge(Clk)) then
    --multiplication is done in a single stage.
        result <= a*b*c*data;
    end if;
END PROCESS;

end Behavioral;
```




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity pipelined is
port (Clk : in std_logic;
        data_in : in integer;
        data_out : out integer;
        a,b,c : in integer
      );
end pipelined;

architecture Behavioral of pipelined is

signal i,data,result : integer := 0;
signal temp1,temp2 : integer := 0;

begin

data <= data_in;
data_out <= result;

--process for calcultation of the equation.
PROCESS(Clk)
BEGIN
    if(rising_edge(Clk)) then
    --Implement the pipeline stages using a for loop and case statement.
    --'i' is the stage number here.
    --The multiplication is done in 3 stages here.
    --See the output waveform of both the modules and compare them.
        for i in 0 to 2 loop
            case i is
                when 0 => temp1 <= a*data;
                when 1 => temp2 <= temp1*b;
                when 2 => result <= temp2*c;
                when others => null;
            end case;
        end loop;
    end if;
END PROCESS;
```

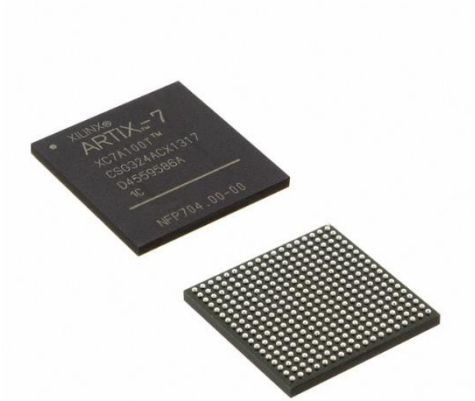
FPGA Nedir ?

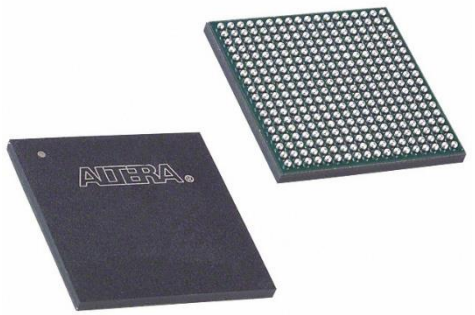
FPGA programlanabilir mantık blokları ve bu bloklar arasındaki ara bağlantılardan oluşan ve geniş uygulama alanlarına sahip olan sayısal tümleşik devrelerdir.*

Test veya üretim amaçlı kullanılabilirler.

Günümüzde üretilen FPGA'lar çok daha fazla gelişmiş özellikler içermektedirler.

Xilinx ve Altera FPGA konusunda önde gelen firmalardan olup FPGA pazarının %90'nını kontrol etmektedirler**. Ayrıca piyasada Lattice Semiconductor, Achronix Microsemi, gibi üreticiler de mevcuttur.





*https://tr.wikipedia.org/wiki/FPGA

**http://www.eejournal.com/archives/articles/20140225-rivalry/

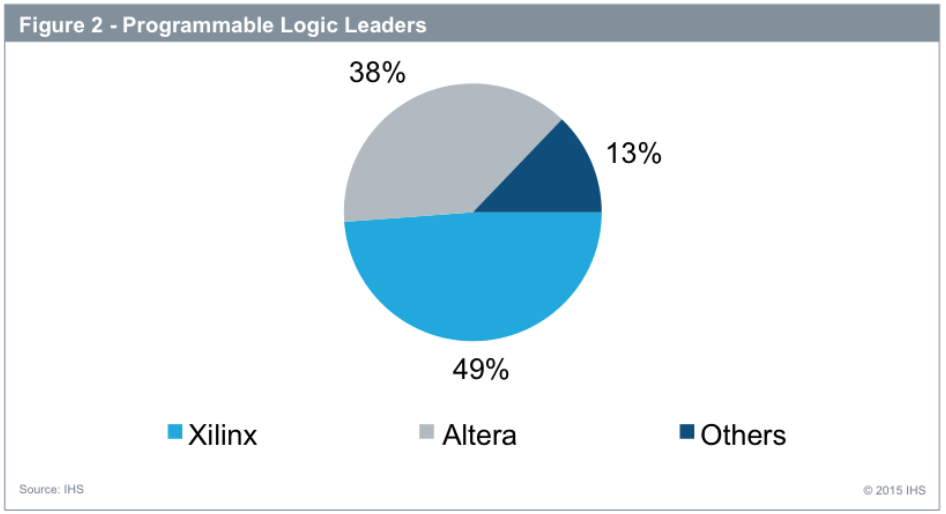
FPGA Nedir ?

FPGA programlanabilir mantık blokları ve bu bloklar arasındaki ara bağlantılardan oluşan ve geniş uygulama alanlarına sahip olan sayısal tümleşik devrelerdir.*

Test veya üretim amaçlı kullanılabilirler.

Günümüzde üretilen FPGA'lar çok daha fazla gelişmiş özellikler içermektedirler.

Xilinx ve Altera FPGA konusunda önde gelen firmalardan olup FPGA pazarının %90'nını kontrol etmektedirler**. Ayrıca piyasada Lattice Semiconductor, Achronix Microsemi, gibi üreticiler de mevcuttur.



*https://tr.wikipedia.org/wiki/FPGA

**http://www.eejournal.com/archives/articles/20140225-rivalry/

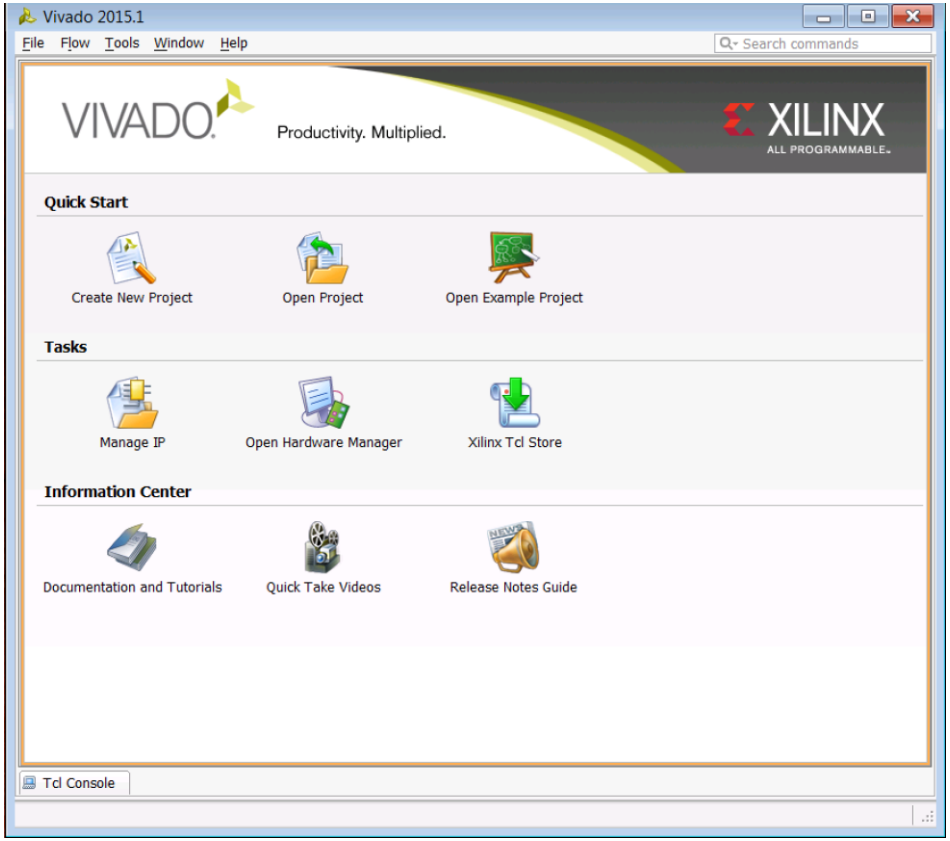
FPGA Nedir ?

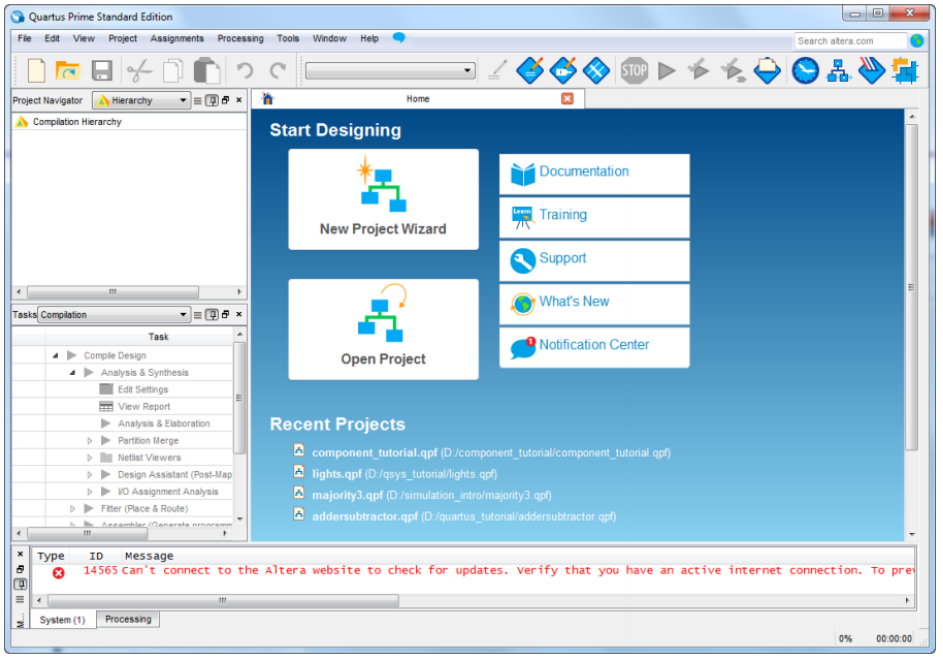
FPGA ile tasarım yapılırken kullanılan iki popüler donanım tanımlama dili mevcuttur. Bunlar **VHDL** (VHSIC* Hardware Description Language) ve **Verilog** dilleridir.

FPGA ile tasarım yapılırken kullanılan diller standart olsa da, çalışma ortamı olarak mutlaka **FPGA** üreticisinin yayınladığı yazılımları kullanmak gerekmektedir. Bununla beraber Synopsys gibi firmalar birden fazla markayı destekleyen yazılımlara sahiptirler.

*VHSIC: Very High Speed Integrated Circuit

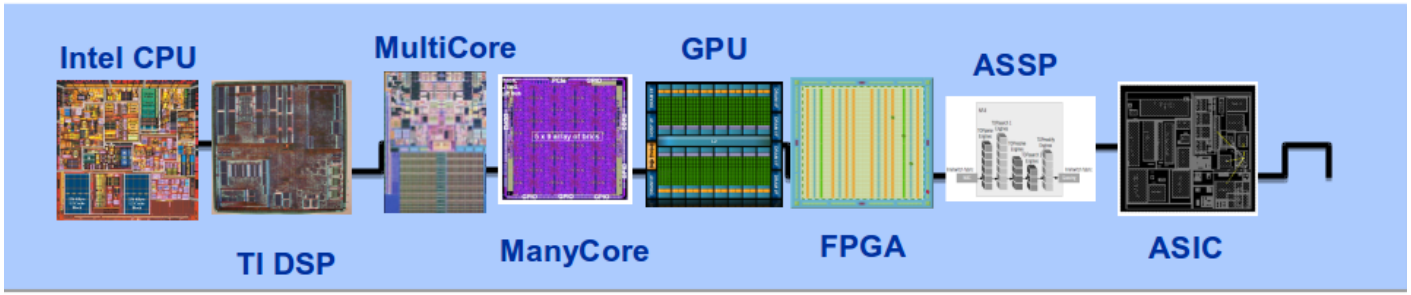
FPGA Nedir ?


Vivado 2015.1
File Flow Tools Window Help
Search commands
VIVADO
Productivity. Multiplied.
XILINX
ALL PROGRAMMABLE.
Quick Start
Create New Project
Open Project
Open Example Project
Tasks
Manage IP
Open Hardware Manager
Xilinx Tcl Store
Information Center
Documentation and Tutorials
Quick Take Videos
Release Notes Guide
Tcl Console


Quartus Prime Standard Edition
File Edit View Project Assignments Processing Tools Window Help
Start Designing
New Project Wizard
Open Project
Documentation
Training
Support
What's New
Notification Center
Recent Projects
component_tutorial.qpf
lights.qpf
majority3.qpf
addersubtractor.qpf
Type ID Message
14565 Can't connect to the Altera website to check for updates. Verify that you have an active internet connection.
System (1)
Processing

FPGA Nedir ?

Programmability: Where do FPGA's fit?



Flexibility, Programming Abstraction

Performance, Area and Power Efficiency

CPU:
- Market-agnostic
- Accessible to many programmers (C++)
- Flexible, portable

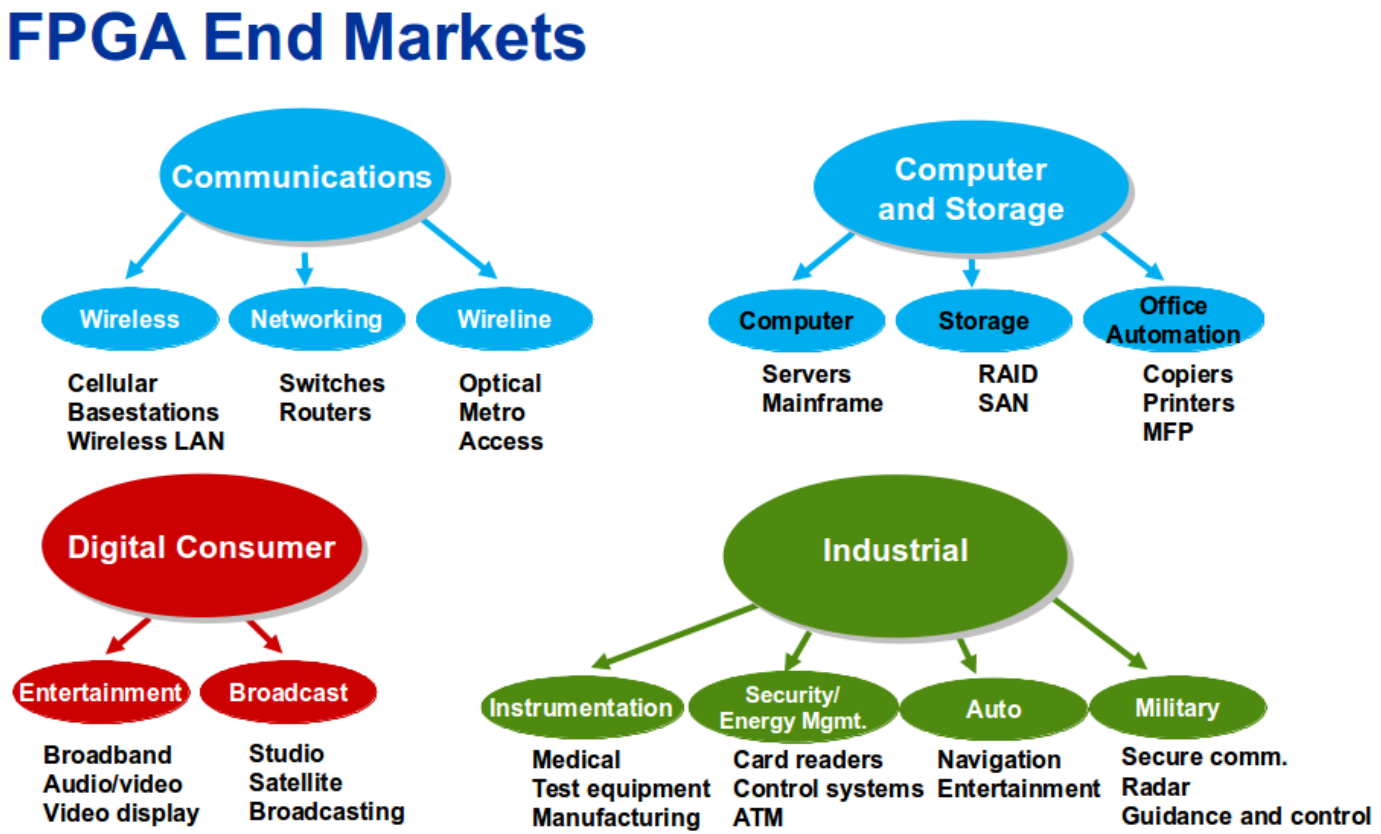
FPGA:
- Somewhat Restricted Market
- Harder to Program (Verilog)
- More efficient than SW
- More expensive than ASIC

ASIC
- Market-specific
- Fewer programmers
- Rigid, less programmable
- Hard to build (physical)

ALTERA

2

FPGA Nedir ?

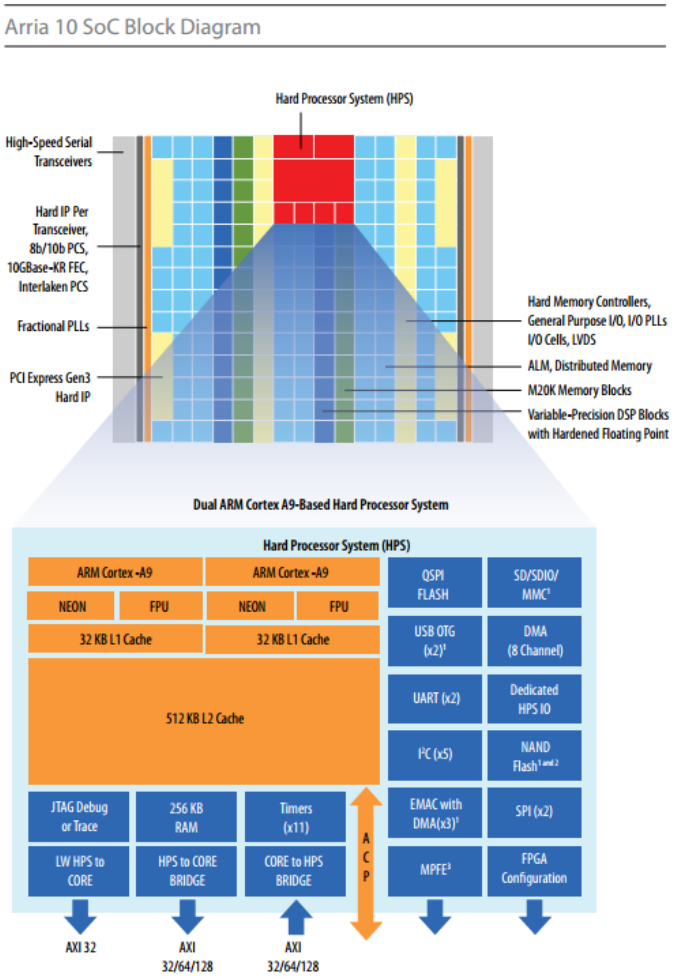

FPGA End Markets
Communications
Wireless
Networking
Wireline
Cellular
Basestations
Wireless LAN
Switches
Routers
Optical
Metro
Access
Computer and Storage
Computer
Storage
Office Automation
Servers
Mainframe
RAID
SAN
Copiers
Printers
MFP
Digital Consumer
Entertainment
Broadcast
Broadband
Audio/video
Video display
Studio
Satellite
Broadcasting
Industrial
Instrumentation
Security/ Energy Mgmt.
Auto
Military
Medical
Test equipment
Manufacturing
Card readers
Control systems
ATM
Navigation
Entertainment
Secure comm.
Radar
Guidance and control
3 / 61

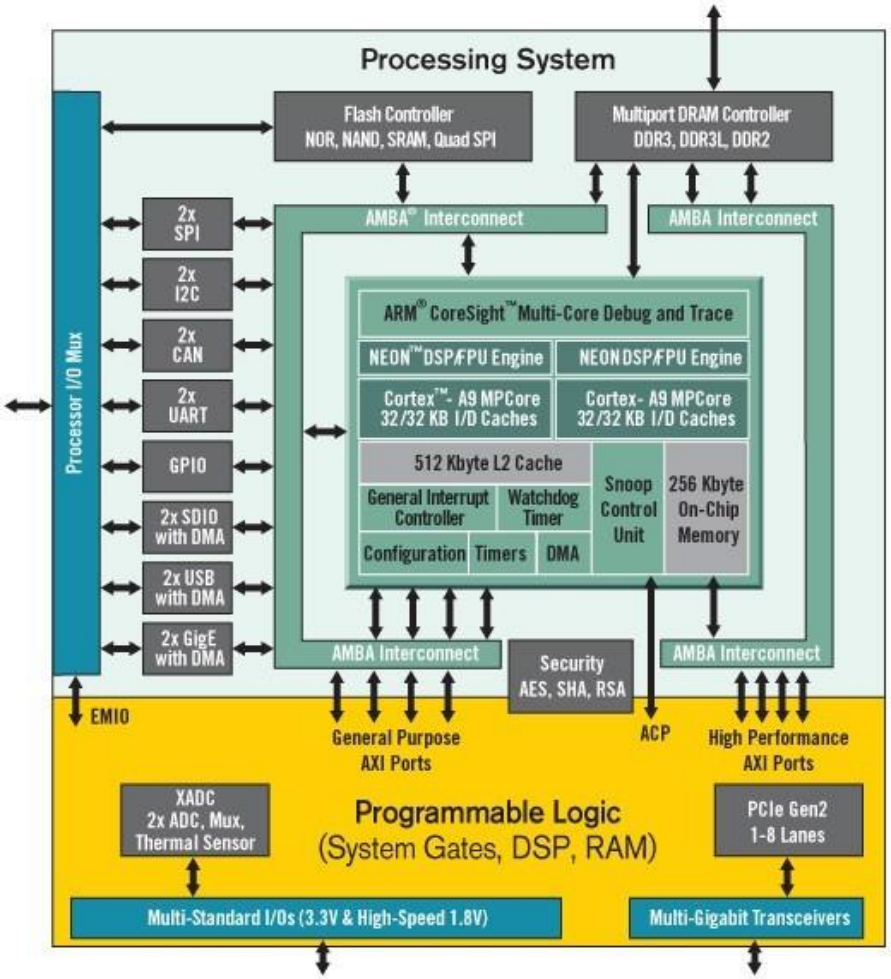
FPGA Nedir ?

CİHAZ	TÜR	İŞLEM GÜCÜ
AMD R9 FURY	GPU	8.6 TFLOPS
NVIDIA P100	GPU	10.6 TFLOPS
Xilin Virtex VU13P	FPGA	4.5 TFLOPS
Altera Stratix GX2800	FPGA	9.2 TFLOPS
Intel Core i7 5960x (8C @ 3GHZ - AVX2)	CPU	0.354 TFLOPS
Intel Xeon Phi 7120 (61C @ 1.238 GHZ)	Coprocessor	2.4 TFLOPS

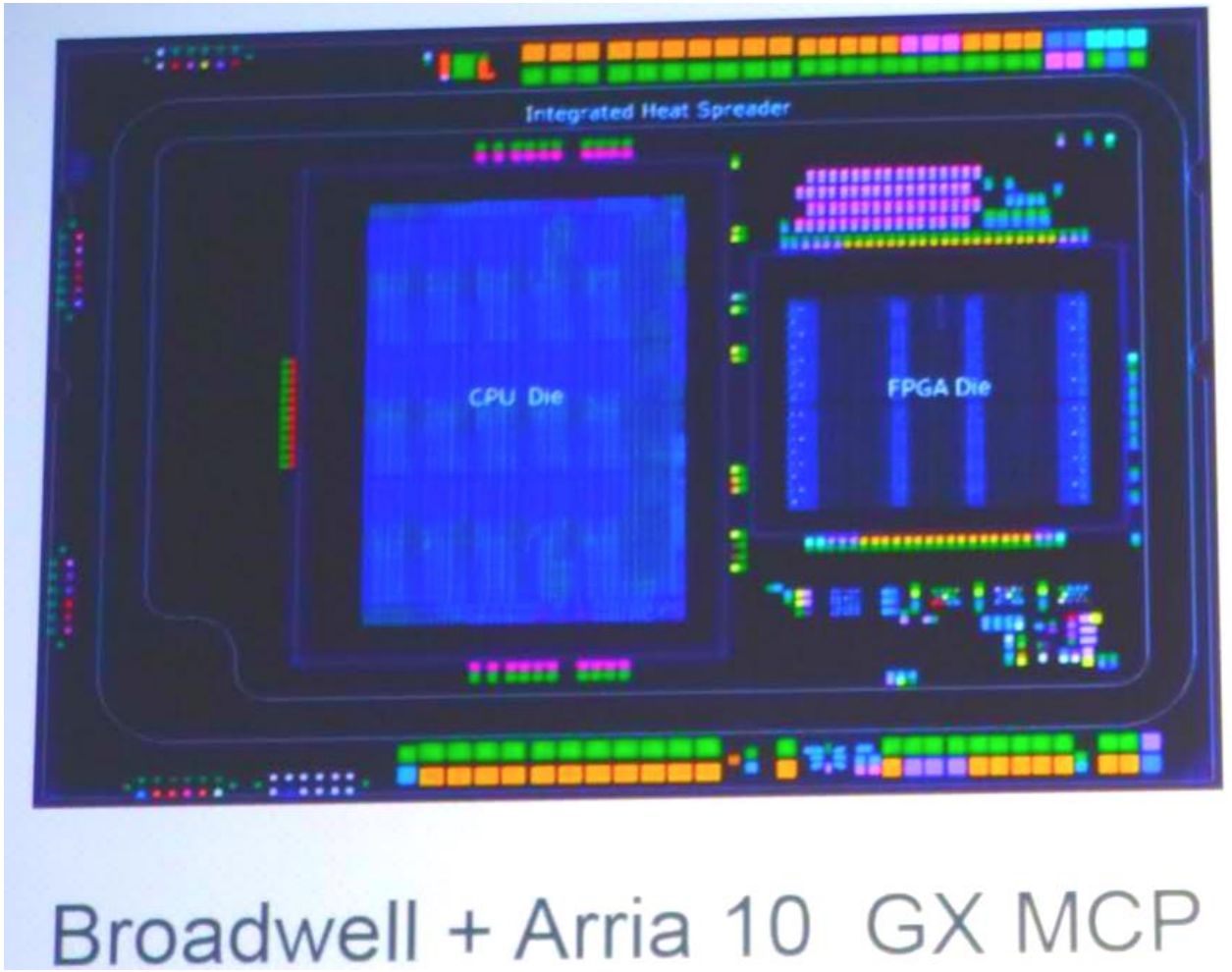
http://www.intel.com/content/www/us/en/benchmarks/server/xeon-phi/xeon-phi-theoretical-maximums.html
http://www.xilinx.com/products/technology/dsp.html
https://www.altera.com/products/fpga/stratix-series/stratix-10/features.html#dsp
https://en.wikipedia.org/wiki/List_of_AMD_graphics_processing_units
https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units
https://www.pugetsystems.com/labs/articles/Linpack-performance-Haswell-E-Core-i7-5960X-and-5930K-594/

FPGA Nedir ?


Arria 10 SoC Block Diagram
Hard Processor System (HPS)
High-Speed Serial Transceivers
Hard IP Per Transceiver, 8b/10b PCS, 10GBase-KR FEC, Interlaken PCS
Fractional PLLs
PCI Express Gen3 Hard IP
Hard Memory Controllers, General Purpose I/O, I/O PLLs I/O Cells, LVDS
ALM, Distributed Memory
M20K Memory Blocks
Variable-Precision DSP Blocks with Hardened Floating Point
Dual ARM Cortex A9-Based Hard Processor System
Hard Processor System (HPS)
ARM Cortex -A9
ARM Cortex -A9
NEON
FPU
NEON
FPU
32 KB L1 Cache
32 KB L1 Cache
512 KB L2 Cache
JTAG Debug or Trace
256 KB RAM
Timers (x11)
LW HPS to CORE
HPS to CORE BRIDGE
CORE to HPS BRIDGE
ACP
QSPI FLASH
SD/SDIO/ MMC
USB OTG (x2)
DMA (8 Channel)
UART (x2)
Dedicated HPS IO
I2C (x5)
NAND Flash
EMAC with DMA(x3)
SPI (x2)
MPFE
FPGA Configuration
AXI 32
AXI 32/64/128
AXI 32/64/128


Processing System
Flash Controller NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller DDR3, DDR3L, DDR2
Processor I/O Mux
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Interconnect
AMBA Interconnect
ARM® CoreSight™ Multi-Core Debug and Trace
NEON™ DSP/FPU Engine
NEON DSP/FPU Engine
Cortex™- A9 MPCore 32/32 KB I/D Caches
Cortex- A9 MPCore 32/32 KB I/D Caches
512 Kbyte L2 Cache
General Interrupt Controller
Watchdog Timer
Configuration
Timers
DMA
Snoop Control Unit
256 Kbyte On-Chip Memory
AMBA Interconnect
Security AES, SHA, RSA
AMBA Interconnect
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XADC 2x ADC, Mux, Thermal Sensor
Programmable Logic (System Gates, DSP, RAM)
PCIe Gen2 1-8 Lanes
Multi-Standard I/Os (3.3V & High-Speed 1.8V)
Multi-Gigabit Transceivers

FPGA Nedir ?


Integrated Heat Spreader
CPU Die
FPGA Die
Broadwell + Arria 10 GX MCP

FPGA ile Tasarım

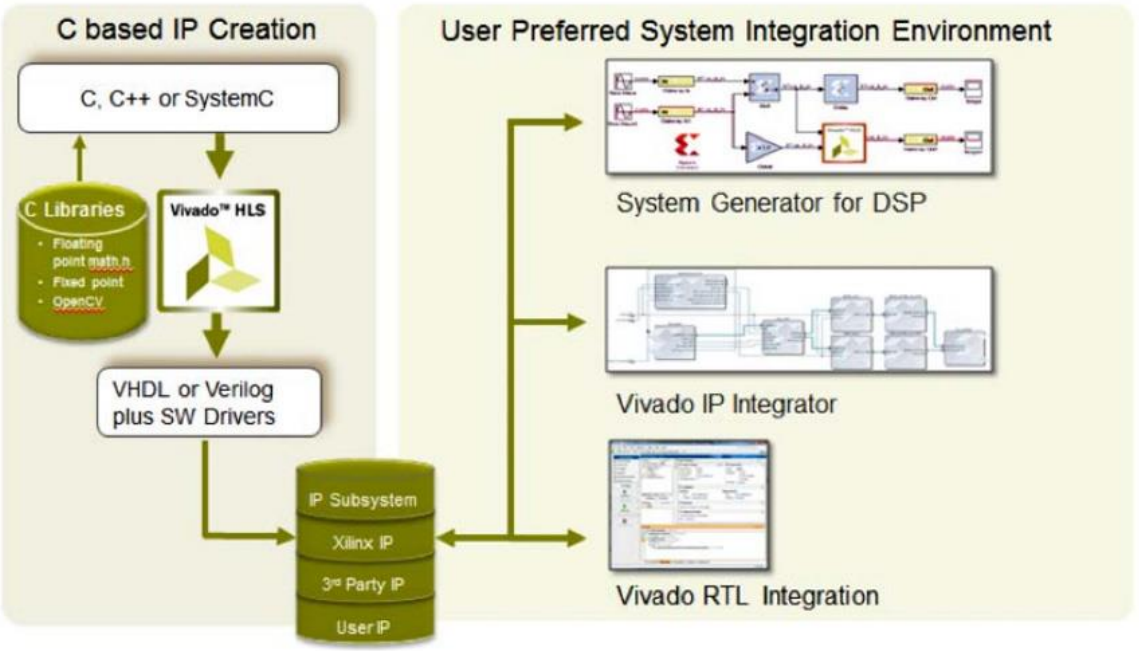
FPGA ile tasarım yaparaken kullanılabilecek pek çok geliştirme aracı günümüzde mevcuttur.

Tasarımcı, kendi tecrübe ve deneyim seviyesine göre bu yollardan birini kullanarak projesini gerçekleştirebilir.

Peki hangi araç ve yol tercih edilmeli ?

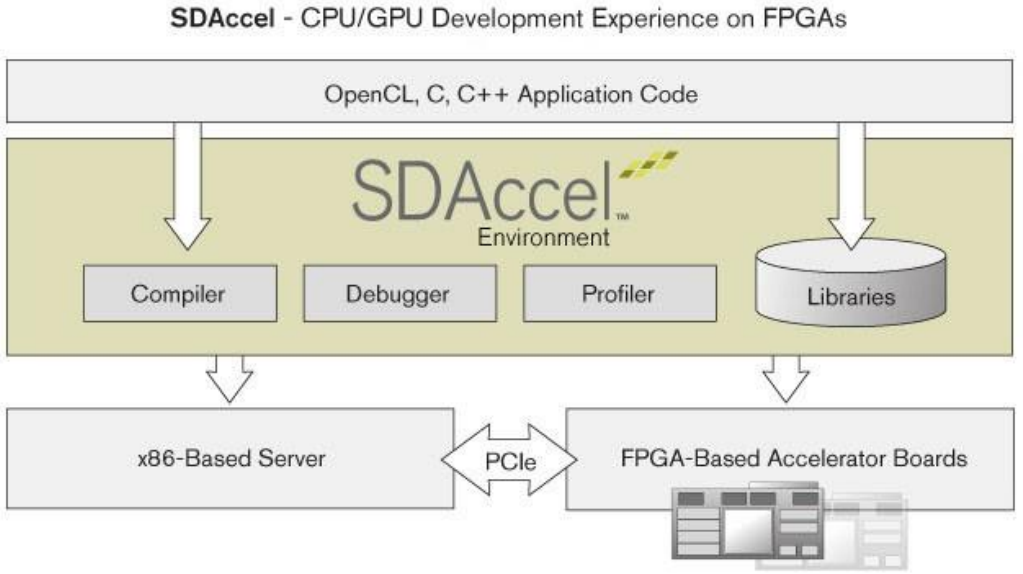
FPGA ile Tasarım

- **Xilinx Vivado HLS**
- Xilinx SDAccel
- Altera OpenCL SDK



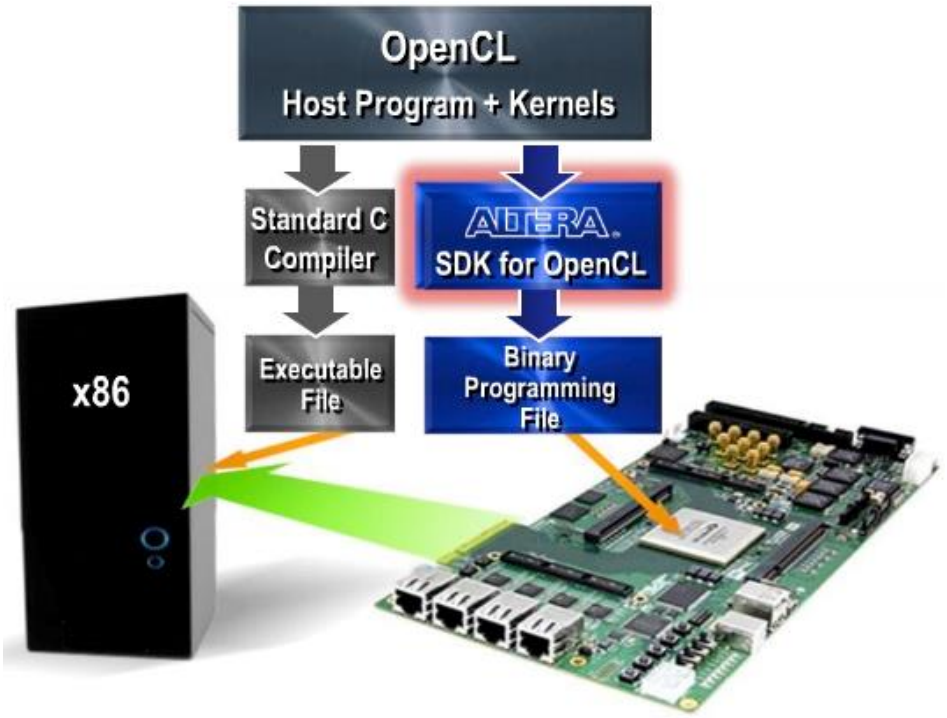
FPGA ile Tasarım

- Xilinx Vivado HLS
- **Xilinx SDAccel**
- Altera OpenCL SDK



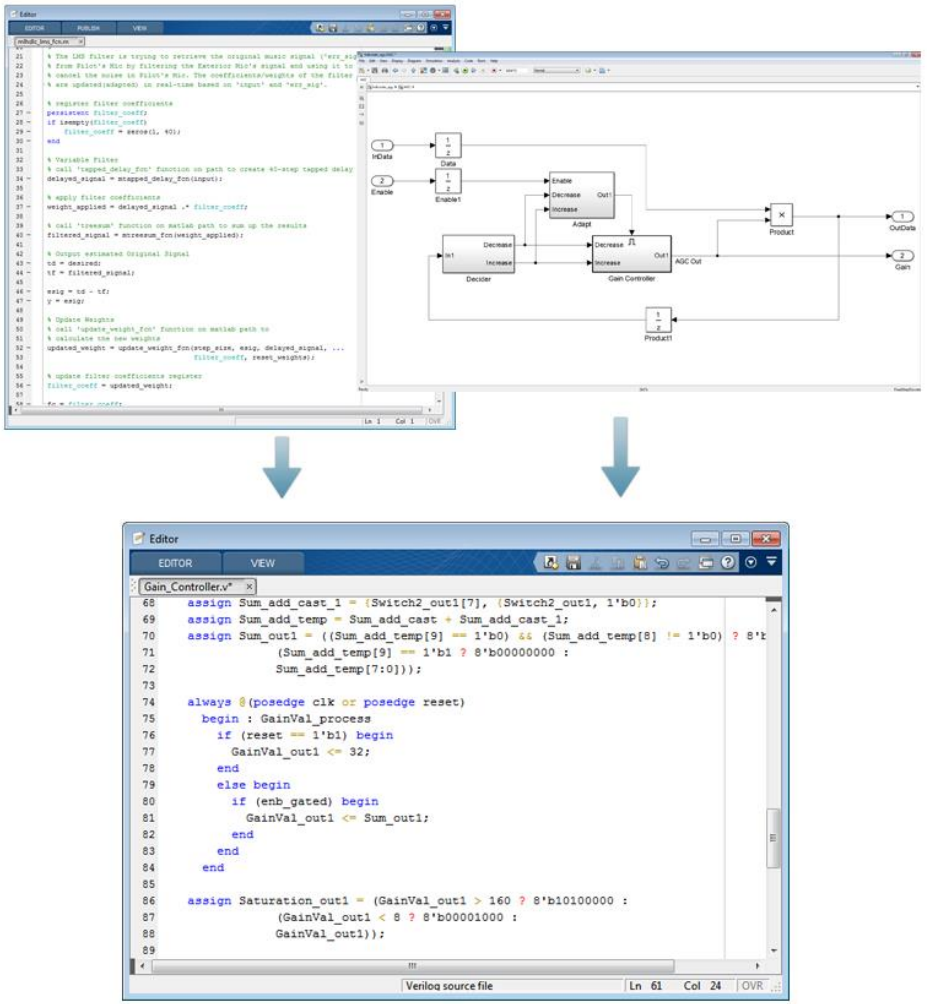
FPGA ile Tasarım

- Xilinx Vivado HLS
- Xilinx SDAccel
- **Altera OpenCL SDK**



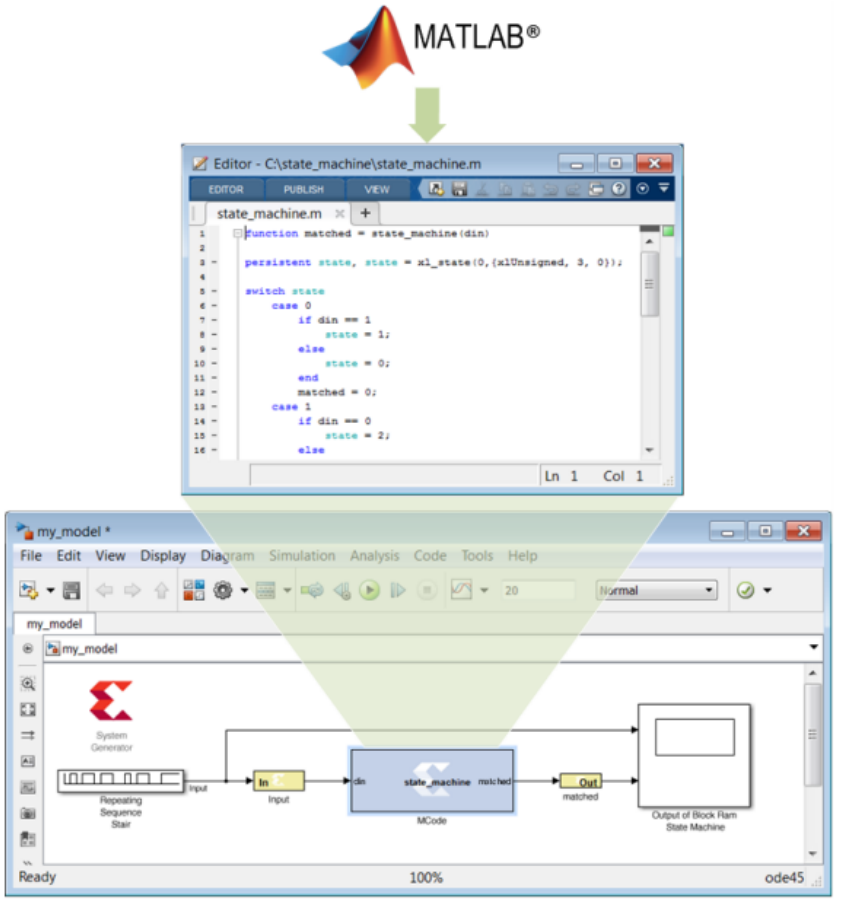
FPGA ile Tasarım

- **Matlab HDL Coder**
- Vivado System Generator For DSP
- Altera DSP Builder



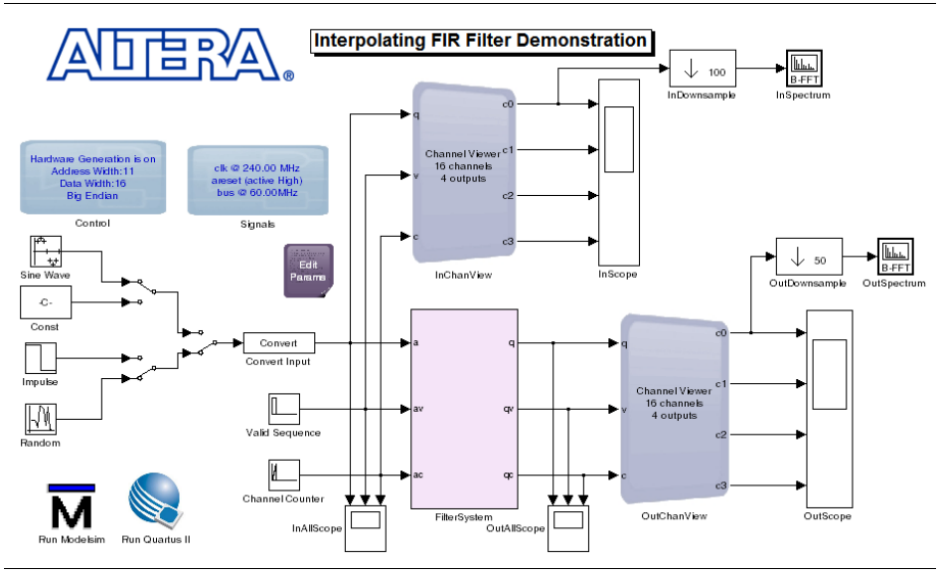
FPGA ile Tasarım

- Matlab HDL Coder
- **Vivado System Generator For DSP**
- Altera DSP Builder



FPGA ile Tasarım

- Matlab HDL Coder
- Vivado System Generator For DSP
- **Altera DSP Builder**



FPGA ile Tasarım

Kullanılabilecek 3 temel yöntemden bahsedilebilir.

- VHDL/Verilog dilleri ile alt seviyeden tasarım.

(VHDL mi Verilog mu ?)

```vhdl
-- (this is a VHDL comment)

-- import std_logic from the IEEE library
library IEEE;
use IEEE.std_logic_1164.all;

-- this is the entity
entity name_of_entity is
   port (
         IN1 : in std_logic;
         IN2 : in std_logic;
         OUT1: out std_logic);
end entity name_of_entity;

-- here comes the architecture
architecture name_of_architecture of name_of_entity is

-- Internal signals and components would be defined here

begin

  OUT1 <= IN1 and IN2;

end architecture name_of_architecture;
```

```
// (4) state waiting for 1T.Lo or 2T.Lo
DEC_STATE_1T2T_LO: begin
    if (POS_EDGE & USEC_PHD) begin
        if (T1_LOW_PLSE) begin
            NextState = DEC_STATE_1T_HI;   // found low pulse so sort out which kind it is
        end
        else begin
            if (T2_LOW_PLSE) begin
                Sh_Dat = 1;                       // log a 1 bit
                Sh_Enb = 1;
                NextState = DEC_STATE_1T2T_HI;
            end
            else begin
                NextState = DEC_STATE_RESET;      // reset if some other pulse width
            end
        end
    end
end
```

FPGA ile Tasarım

$\mathbf{L}(1) \leftarrow \mathbf{L}(1)/\mathbf{D}(1)$;
for $i \leftarrow 2$ **to** n **do**
  $\mathbf{D}(i) \leftarrow \mathbf{D}(i) - \mathbf{L}(i-1) \times \mathbf{U}(i)$;
  $\mathbf{L}(i) \leftarrow \mathbf{L}(i)/\mathbf{D}(i)$;
  $\mathbf{b}(i) \leftarrow \mathbf{b}(i) - \mathbf{L}(i-1) \times \mathbf{b}(i-1)$;
end

Algorithm 1: The TDMA LU-decomposition/Forward Substitution

$\mathbf{b}(n) \leftarrow \mathbf{b}(n)/\mathbf{D}(n)$;
for $i \leftarrow n-1$ **to** 1 **do**
  $\mathbf{b}(i) \leftarrow \mathbf{b}(i) - \mathbf{b}(i+1) \times \mathbf{U}(i+1)$;
  $\mathbf{b}(i) \leftarrow \mathbf{D}(i)$;
end

Algorithm 2: The TDMA Backward Substitution

A tri-diagonal linear system has the form

$$
\begin{aligned}
a_{1,1}x_1 + a_{1,2}x_2 &= b_1 \\
a_{i,(i-1)}x_{i-1} + a_{i,i}x_i + a_{i,(i+1)}x_{i+1} &= b_i, \;\; i \in [2, 3, \ldots, n-1] \\
a_{n,(n-1)}x_{n-1} + a_{n,n}x_n &= b_n.
\end{aligned}
$$

Such a system can be represented compactly as four arrays of n elements,

$$
\begin{aligned}
\mathbf{U} &= [0, a_{1,2}, a_{1,3}, \ldots, a_{(n-1),n}] \\
\mathbf{D} &= [a_{1,1}, a_{2,2}, \ldots, a_{n,n}] \\
\mathbf{L} &= [a_{2,1}, a_{3,2}, \ldots, a_{n,(n-1)}, 0] \\
\mathbf{b} &= [b_1, b_2, \ldots, b_n]
\end{aligned}
$$

David J. Warne, Neil A. Kelson, Ross F. Hayward, Comparison of High Level FPGA Hardware Design for Solving Tri-diagonal Linear Systems, Procedia Computer Science, Volume 29, 2014, Pages 95-101, ISSN 1877-0509

FPGA ile Tasarım

Table 1: Device Utilisation

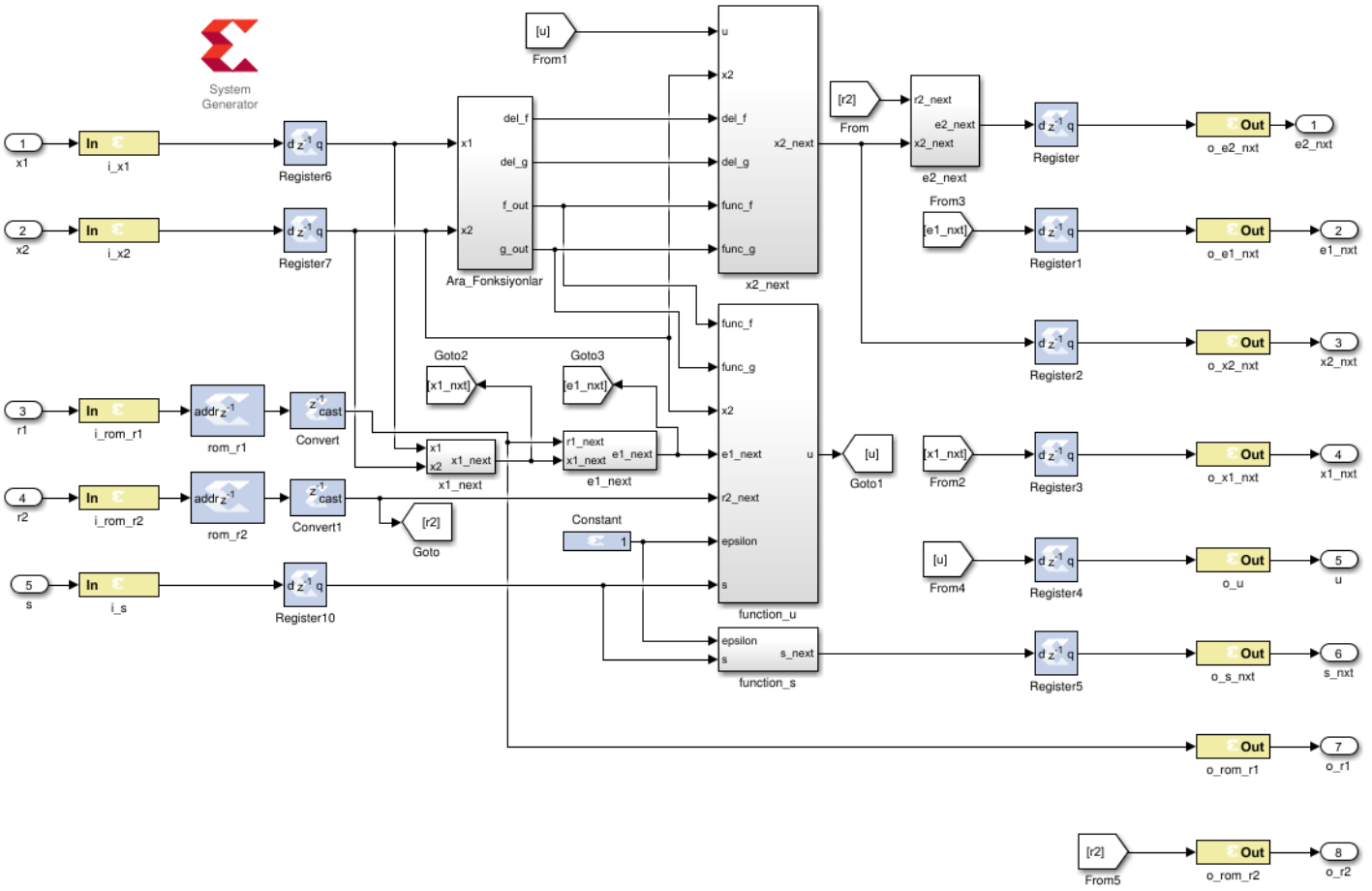
Logic Utilisation	VHDL	Altera OCL	Xilinx HLS
LUT	211,297	22,255	3,765
FF	262,504	39,465	2,310
DSP	-	46	10
BRAM (Kb)	-	557	180

Table 2: Run-time Comparison and Solution Error

	VHDL	Altera OCL[1]	Xilinx HLS
Run-time (secs)	0.000671	0.002752	0.000465
$\|\|\mathbf{x} - \mathbf{b}\|\|_\infty$	1.79e-04	6.60e-05	4.40e-05

David J. Warne, Neil A. Kelson, Ross F. Hayward, Comparison of High Level FPGA Hardware Design for Solving Tri-diagonal Linear Systems, Procedia Computer Science, Volume 29, 2014, Pages 95-101, ISSN 1877-0509

FPGA ile Tasarım



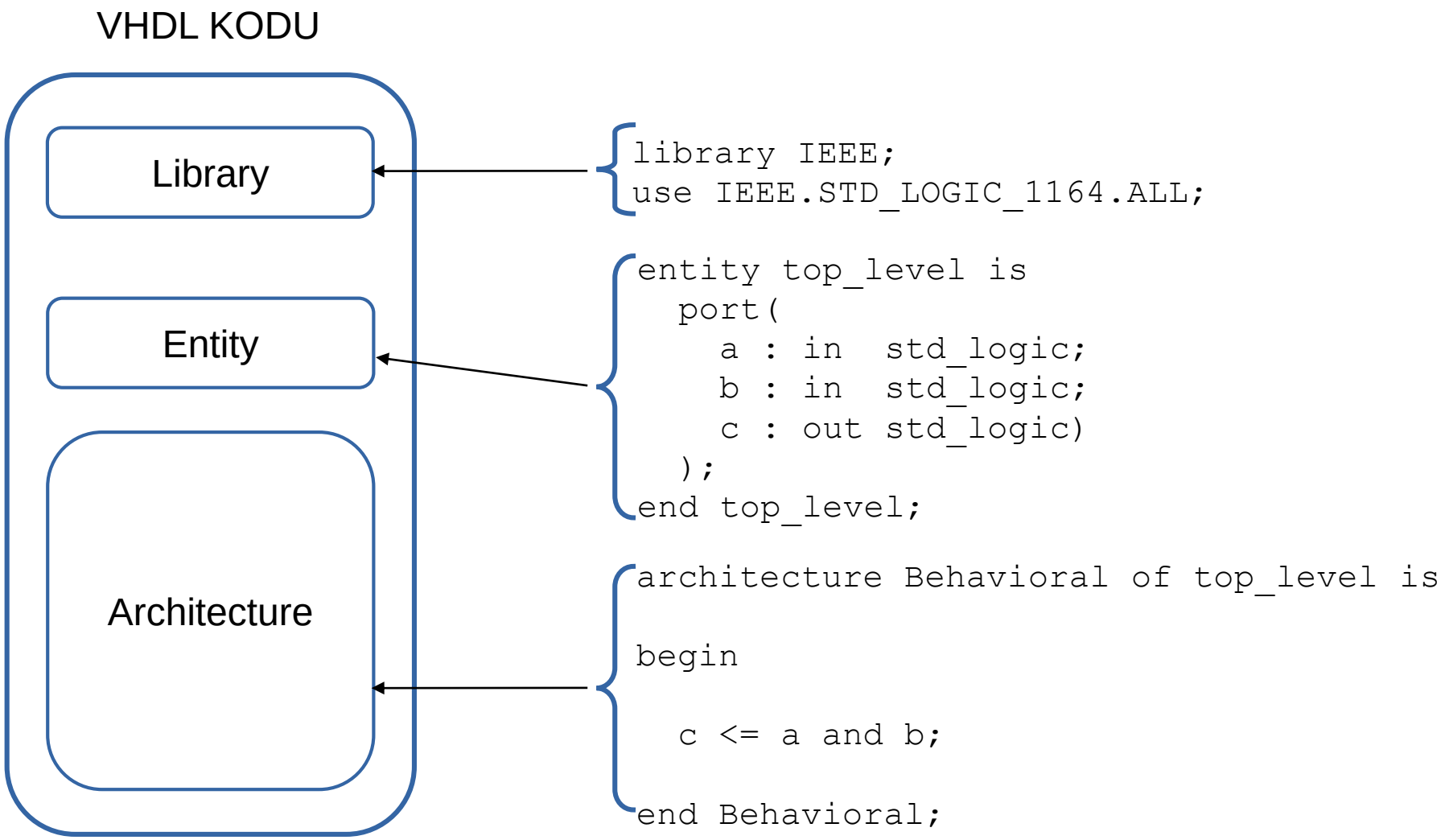
FPGA ile Tasarım

```
    o_e2_nxt => o_e2_nxt,
    o_rom_r1 => o_rom_r1,
    o_rom_r2 => o_rom_r2,
    o_s_nxt => o_s_nxt,
    o_u => o_u,
    o_x1_nxt => o_x1_nxt,
    o_x2_nxt => o_x2_nxt
  );
end structural;

-:---   top_sys.vhd     Bot L9312   (VHDL/es)
```

```
latency_1: if (latency <= 1) generate
data <= core_data_out;
end generate;
end behavior;

-:---   top_sys_entity_declarations.vhd   Bot L3822   (VHDL/es)
```

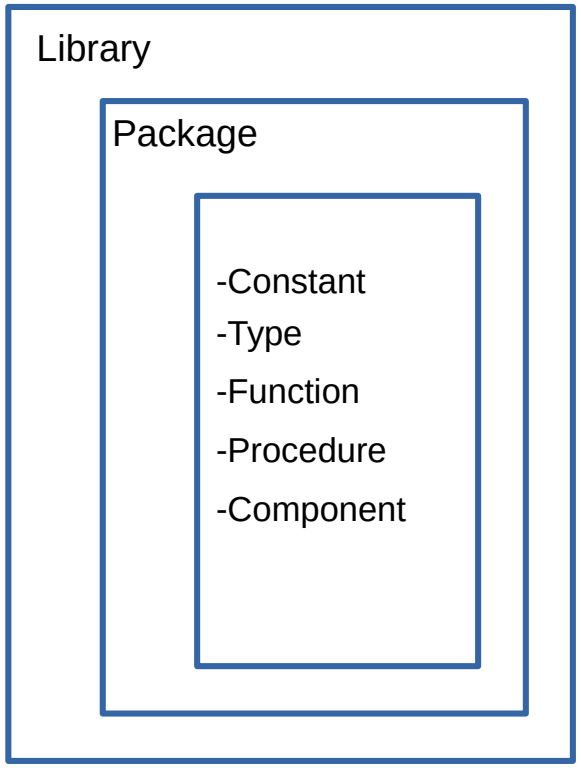
FPGA ile Tasarım

VHDL


VHDL KODU
Library
Entity
Architecture
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity top_level is
port(
a : in std_logic;
b : in std_logic;
c : out std_logic)
);
end top_level;
architecture Behavioral of top_level is
begin
c <= a and b;
end Behavioral;

VHDL - Library

Library içerisinde pek çok alt bileşen mevcuttur.



Synopsys

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

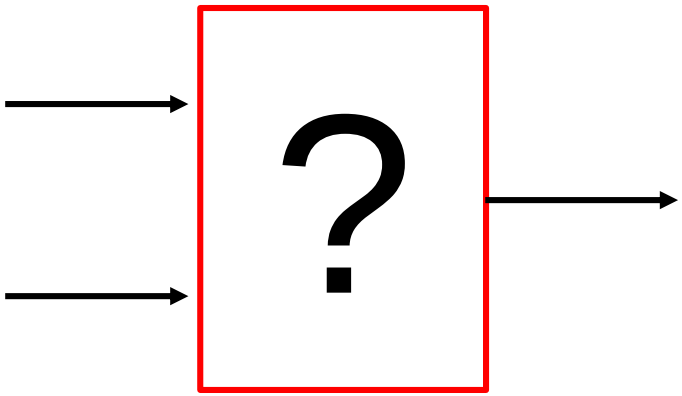
IEEE

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

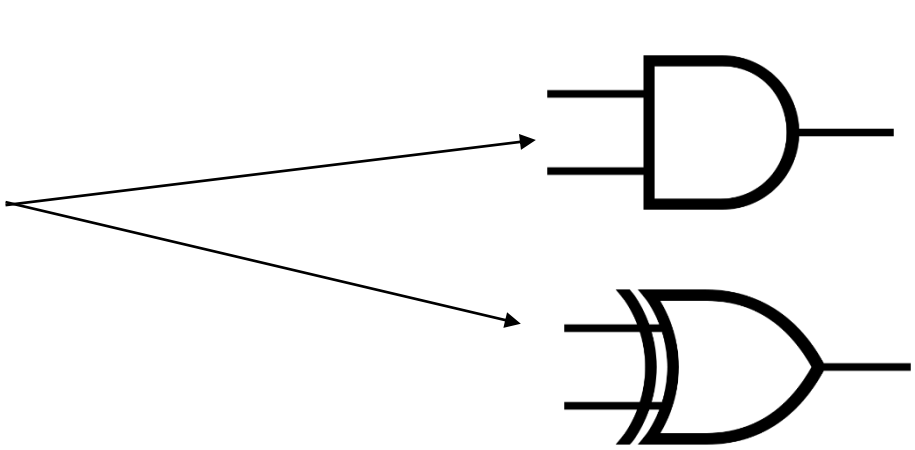
http://www.synthworks.com/papers/vhdl_math_tricks_mapld_2003.pdf

VHDL - Entity

Entity kısmında tasarıma ait giriş ve çıkışlar tanımlanır.




```
entity top_level is
  port(
    a : in  std_logic;
    b : in  std_logic;
    c : out std_logic)
  );
end top_level;
```

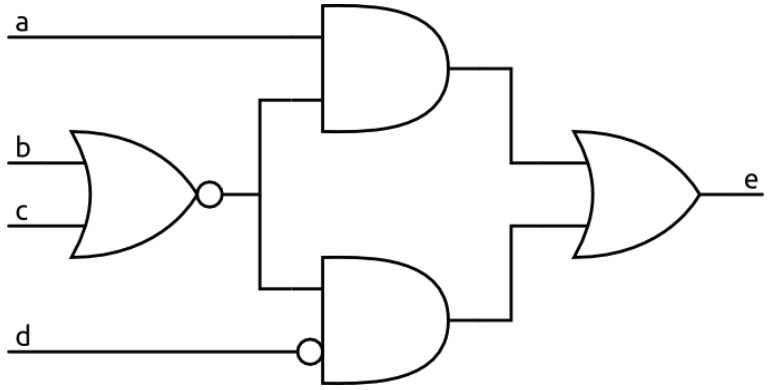


VHDL - Architecture

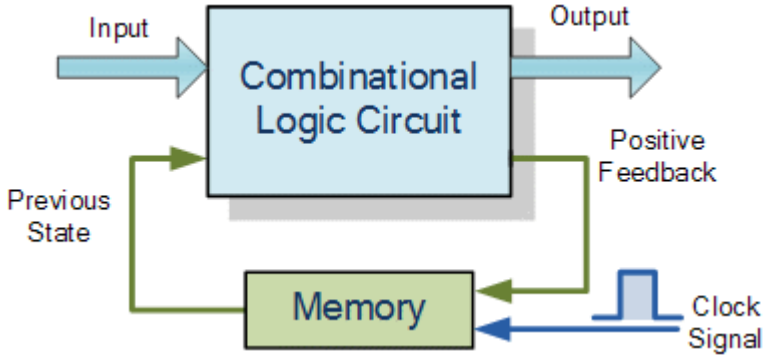
Architecture kısmında tasarımın iç yapısı inşa edilir

```
architecture Behavioral of top_level
is
   [constant tanımlama]
   [signal tanımlama]
   [type tanımlama]
   [component tanımlama]
   [attribute tanımlama]

begin

   [Kombinasyonel devreler]
   [Ardışıl devreler]
   [Component bağlantıları]

end Behavioral;
```

Kombinasyonel Devre



Ardışıl Devre



VHDL – Veri Tipleri

VHDL dilinde ön tanımlı olarak gelen veri tipleri ile kullanıcı tanımlı veri tipleri mevcuttur. Ön tanımlı veri tiplerinin bir kısmı sentezlenebilr iken bir kısmı ise sentezlenemez.

```
-----------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

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

VERİ TİPİ	DEĞERLERİ
BIT, BIT_VECTOR	'0', '1'
STD_LOGIC, STD_LOGIC_VECTOR	'X', '0', '1', 'Z', (U)
BOOLEAN	'TRUE', 'FALSE'
NATURAL	0 ile 2.147.483.647
INTEGER	± 2.147.483.647
SIGNED	± 2.147.483.647
UNSIGNED	0 ile 2.147.483.647
TYPE	Kullanıcı Tanımlı
SUBTYPE	Kullanıcı Tanımlı
ARRAY	Var olan tiplere göre (tek tip içerebilir)
RECORD	Var olan tiplere göre (farklı tipler içerebilir)

VHDL – Veri Tipleri

```
a <= '1';          -- bit, std_logic, std_ulogic

b <= "00101100"; -- std_logic_vector,std_ulogic_vector,bit_vector,
                   -- unsigned, signed , 28 sayısı

c <= "0010_1100";-- b ile aynı okuma kolaylığı sağlar

d <= O"34";        -- 28 sayısının sekizlik tabanda ifadesi

e <= X"1C";        -- 28 sayısının HEX ifadesi

f <= 28;           -- integer gösterim
```

VHDL – Veri Tipleri - TYPE

TYPE ifadesi VHDL'de sıklıkla kullanılan bir tanımlamadır. Bu sayede tasarımda kişiye özel veri tiplerinin kullanılabilmesini sağlar. En sık kullanım biçimi "**Enumarated Type**" tarzıdır.

Bir diğer kullanımı ise var olan tiplerden yeni, kişiye özel tipler oluşturmaktır.

Ayrıca **ARRAY** tanımlamaları da bu ifade ile oluşturulmaktadır.

Enumarated Type

```
type durumlar is (BOSTA, BASLA, BITIR);
type renkler is (KIRMIZI, SARI, YESIL);
```

Var Olandan

```
type int_5bit is range 0 to 31;
type notlar is range 0 to 100;
```

ARRAY Tanımlama

```
type matris is array (0 to 7) of std_logic_vector(7 downto 0);
type matris is array (0 to 7, 3 downto 0) of std_logic;
```

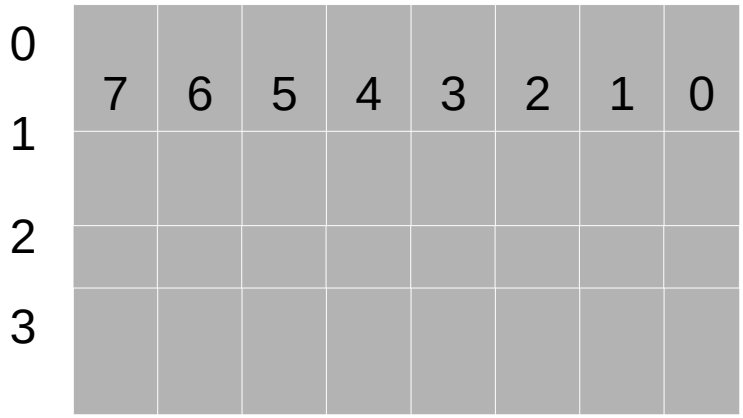
VHDL – Veri Tipleri - ARRAY

VHDL dilinde ön tanımlı olarak dizi yapıları yoktur. Bu yapının kullanıcı tarafından oluşturulması gerekmektedir. Bunun için **TYPE** ifadesi ile dizi yapısı oluşturulur.

```
--------------------1Dx1D---------------------
type satir is array (7 downto 0) of std_logic;
type matris is array (0 to 3) of satir;
signal a : matris;

--------------------1Dx1D---------------------
type matris is array (0 to 3) of
std_logic_vector(7 downto 0);

--------------------2Dx2D---------------------
type matris is array (0 to 3, 7 downto 0) of
std_logic;
```

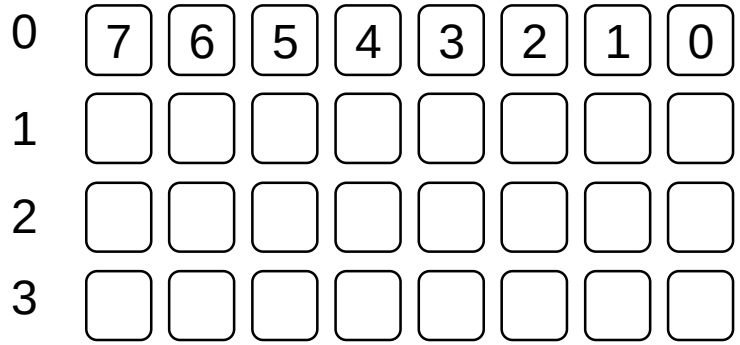
```
type type_ismi is array (boyut_bilgisi) of veri_tipi;
signal signal_ismi : type_ismi := (on tanımlı alacağı değer);
-------------------------------------------------------------------
-
constant const_ismi : type_ismi := (on tanımlı alacağı değer);
-------------------------------------------------------------------
-
variable variable_ismi : type_ismi := (on tanımlı alacağı değer);
```

VHDL – Veri Tipleri - ARRAY



1D Dizi

```
b <= a(0)(5);
c <= a(0)(7 downto 4);
```



2D Dizi

```
b <= a(0,5);
```

VHDL - Operatörler

Operatör Türü	Operatör	Veri Tipi
Atama	<= => :=	Hepsi
Mantık	NOT, AND, OR, NAND, NOR, XOR,XNOR	STD_(U)LOGIC, STD_(U)LOGIC_VECTOR
Aritmetik	+, - , * , ** (/ , mod ve rem sentezlenemez)	INTEGER, SIGNED, UNSIGNED
Karşılaştırma	<, >, <=, >=, =, /=	Hepsi
Birleştirme	&	STD_(U)LOGIC, STD_(U)LOGIC_VECTOR, SIGNED, UNSIGNED

NOT: Tabloda yer almamasına karşın kaydırma işlemleri için de operatörler bulunmakla beraber sentezlenmede sorunlar yaşanabildiği için listeye dahil edilmemiştir. Kaydırma işlemleri için '&' -birleştirme operatörü- kullanılabilmektedir.

VHDL - Operatörler

Sola Kaydırma İşlemi

```
a <= a(6 downto 0)&'0';
```

Sağa Kaydırma İşlemi

```
a <= '0'&(7 downto 1)
```

Kaydırma işlemleri sırasında eklenen değerin sıfır olduğu kabul edilmiştir.

VHDL - Attributes

```
signal a : std_logic_vector(7 downto 0)
```

a'**LOW = 0**

a'**HIGH** = 7

a'**LEFT** = 7

a'**RIGHT** = 0

a'**RANGE** = (7 downto 0)

a'**REVERSE_RANGE** = (0 downto 7)

a'**LENGTH** = 8

Ayrıca bunların dışında kullanılabilen 'EVENT' ve 'STABLE' ifadeleri de mevcuttur.

VHDL - Generic

“Generic” ifadesi VHDL dilinde var olan kullanışlı bir genelleştirme ifadesidir. Bu özellik gözetilerek yapılan tasarımlar daha kolay bir şekilde güncellenebilmekte ve esnek bir kullanım imkanı sunmaktadır.

```
library ieee;
use ieee.std_logic_1164.all;
entity deneme is

  generic (
    m : integer := 8);
  port (
    giris : in  std_logic_vector(m-1 downto 0);
    cikis : out std_logic_vector(m-1 downto 0));
end entity deneme
```

VHDL – Kombinasyonel Devreler

WHEN/ELSE

```
sinyal <= deger1 when koşul1
else
          deger2 when koşul2
else
          deger3;


cikis <= X"00" when rst= '0'
else
         X"0F" when set= '1'
else
         X"F0";
```

WITH/ELSE/WHEN

```
with durumlar select
     cikis <=
     X"00" when BOSTA,
     X"0F" when BASLA,
     X"F0" when BITIR,
     X"FF" when others;
```

NOT: GENERATE ve BLOCK ifadeleri de mevcut olmakla birlikte bu aşamada anlatılmayacaktır.

VHDL – Constant, Signal, Variable

VHDL ile tasarım yapılırken kullanabileceğimiz 3 temel tanımlama vardır.

Constant: İsminden anlaşılacağı üzere sabit değerleri tanımlamak için kullanılır. **Entity**, **Architecture** ve **Package** içerisinde tanımlanabilmekle birlikte en sık kullanılan şekli **Architecture** içinde tanımlamaktır.

Signal: Tasarlanan devreler arasında verileri aktarmak için kullanılan ifadedir. Bir nevi kablo görevi görürler. Değerleri hemen güncellenmeyebilir. Değerleri ancak tek bir kaynaktan değiştirilebilir fakat birden fazla noktadan erişilip okunabilir. Değer atamak için “<=” operatörü kullanılır.

Variable: Process, **Procedure** ve **Function** içinde tanımlanabilirler. Değerleri hemen güncellenir. Sadece tanımlandığı yerden erişilebilir. Değer atamak için “:=” operatörü kullanılır.

VHDL – Ardışıl Devreler

VHDL kodunda 3 temel ardışıl devre tanımlaması vardır. Bunlar **Process**, **Function** ve **Procedure** tanımlamalarıdır. Bu kısımda **Process**'lerden bahsedilecektir.

Bir **Process** içerisinde var olan 4 temel tanımlama mevcuttur. Bunlar **IF**, **CASE**, **LOOP** ve **WAIT** tanımlamalarıdır.

Process içerisinde var olan işlemler sıralı biçimde işlenir. Bir tasarımda birden fazla **Process** olabilir. Birbiri ile neden-sonuç ilişkisi olmayan Process'ler paralel çalışır.

VHDL – Process - IF

Yan tarafta verilen kod ile eşzamanlı olmayan sıfırlama girişine sahip bir D-FF tasarımı gerçekleştirilmiştir.

Process tanımlaması yapılırken Process'i tetikleyek olan sinyaller "**sensitivity list**" ile ifade edilir.

```
library ieee;
use ieee.std_logic_1164.all;
entity dff is
  port (
    clk : in  std_logic;
    rst : in  std_logic;
    d   : in  std_logic;
    q   : out std_logic);
end entity dff;

architecture behave of dff is
-- signal tanımı burada yapılır
Begin

  DFF1: process (clk, rst) is
    -- variable tanımı burada yapılır
  begin  -- process DFF1
    if rst = '0' then
      q <= '0';
    elsif clk'event and clk = '1' then
      q <= d;
    end if;
  end process DFF1;
end architecture behave;
```

VHDL – Process - CASE

Yan tarafta verilen kod ile eşzamanlı sıfırlama girişine sahip bir D-FF tasarımı gerçekleştirilmiştir.

Bir önceki örnekten farklı olarak bu sefer case yapısı kullanılmıştır.

```
library ieee;
use ieee.std_logic_1164.all;

entity dff is
  port (
    clk : in  std_logic;
    rst : in  std_logic;
    d   : in  std_logic;
    q   : out std_logic);

end entity dff;

architecture behave of dff is
begin
  DF1: process (clk, rst) is
  begin
    if clk'event and clk = '1' then
      case rst is
        when '0' => q <= '0' ;
        when '1' => q <= d;
      end case;
    end if;
  end process DF1;

end architecture behave;
```

VHDL – Process - LOOP

VHDL dilinde var olan LOOP ifadesi diğer programlama dillerinde var olandan farklıdır. LOOP içerisinde tanımlanan döngüler bir kerede gerçekleştirilir.

LOOP ifadeleri ki bunlar WHILE LOOP ve FOR LOOP olarak isimlendirilir, tekrar eden tasarım ifadelerini sade bir şekilde yazmak için kullanılır.

```
for i in 0 to 7 loop
  sonuc(i) <= a(i) xor b(i);
end loop;
```

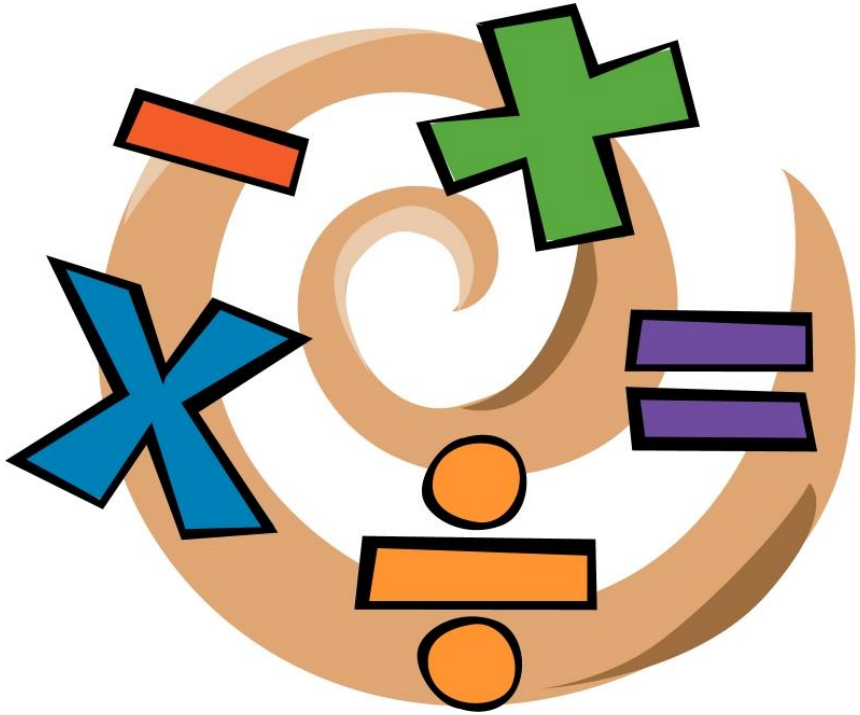
VHDL

VHDL'den önce

VHDL'den sonra

Sayı Biçimleri

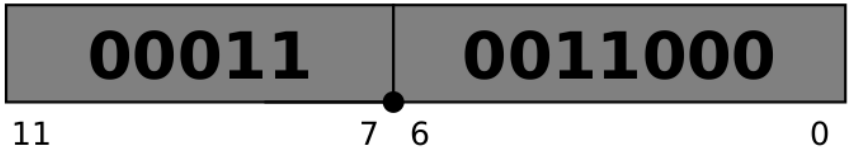
Sayısal sistemlerde matematiksel işlemler yapılırken kullanılan temel iki sayı sisteminden bahsedilebilir.



- **Sabit Noktalı Sayılar**
- **Kayan Noktalı Sayılar**

Sayı Biçimleri

Sabit noktalı sayı formatı sayının tam kısmı ve ondalık kısmının beraberce gösterimi ile elde edilir.



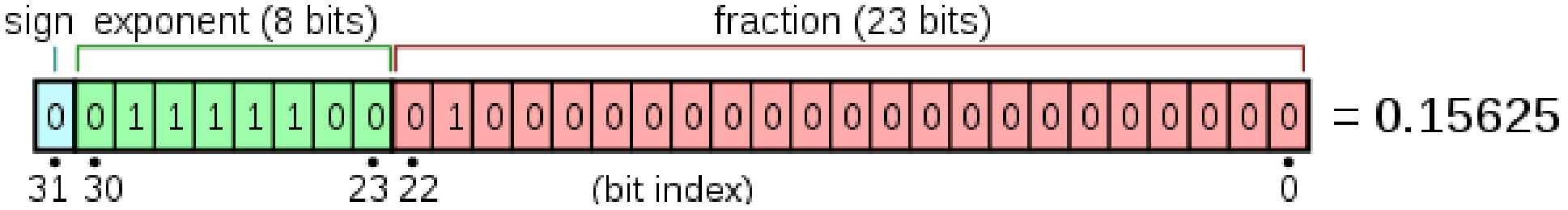
Bu sayı sisteminde tasarlanan yapının ihtiyacına göre azami ve asgari sayı büyüklüğü, istenen hassasiyet gibi bileşenler göz önüne alınarak bit uzunlukları belirlenir.

Sabit noktalı sayılarda kullanılan bitsayısı, ifade edilmeye çalışılan değerin hassasiyetinde doğrudan etkilidir.

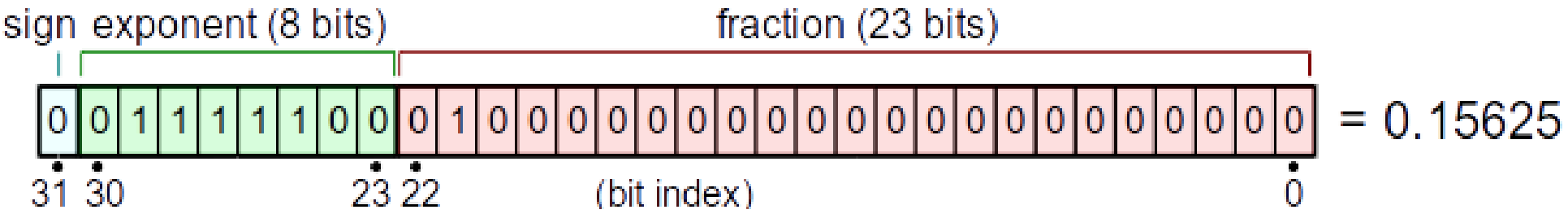
Sayı Biçimleri

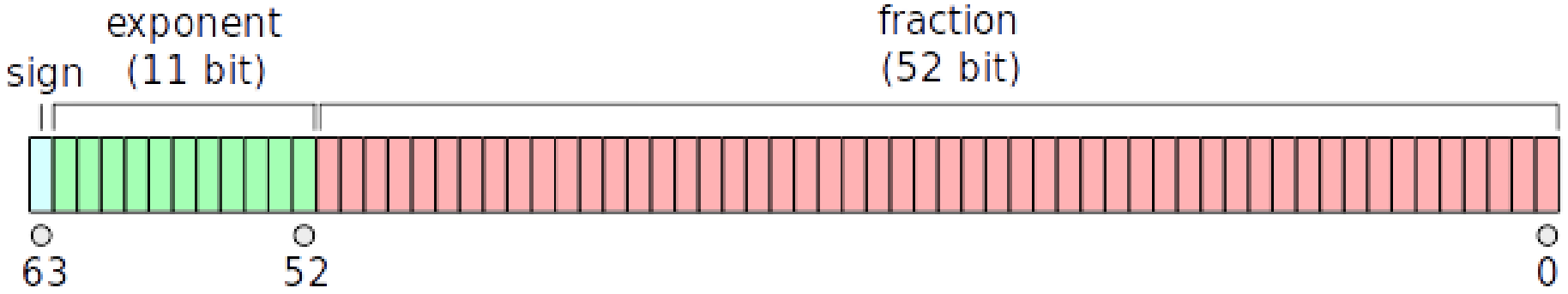
Kayan noktalı sayı biçimi gerçek sayıların bilgisayar ortamında gösterim şeklidir. Bu sayı formatı IEEE-754 standardı olarak da kabul edilmektedir.

Kayan noktalı sayı gösterimi sabit noktalı sayılara göre daha geniş sayı aralığında işlem yapmakta fakat buna karşılık ihtiyaç duyduğu donanım kaynağı miktarı yüksek olmaktadır.



Sayı Biçimleri





Sayı Biçimleri

$$\text{value} = (-1)^{\text{sign}} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i}\right) \times 2^{(e-127)}$$

$$(-1)^{\text{sign}} \left(1 + \sum_{i=1}^{52} b_{52-i} 2^{-i}\right) \times 2^{e-1023}$$

Effective Floating-Point Range

	Binary	Decimal
Single	$\pm (2-2^{-23}) \times 2^{127}$	$\approx \pm 10^{38.53}$
Double	$\pm (2-2^{-52}) \times 2^{1023}$	$\approx \pm 10^{308.25}$

UYGULAMA

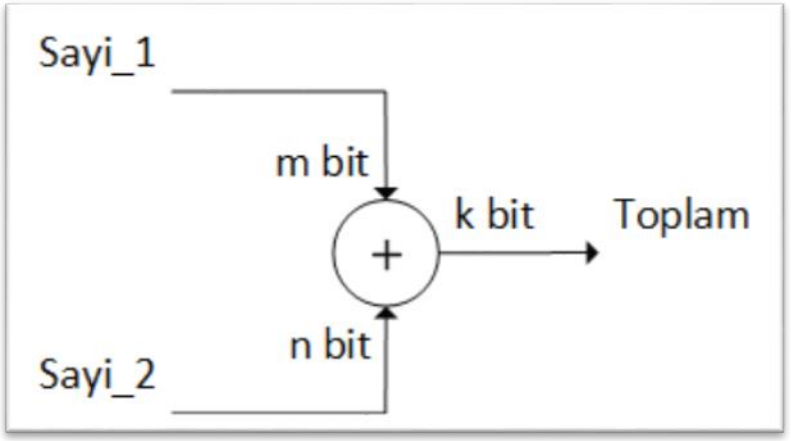
Adımlar

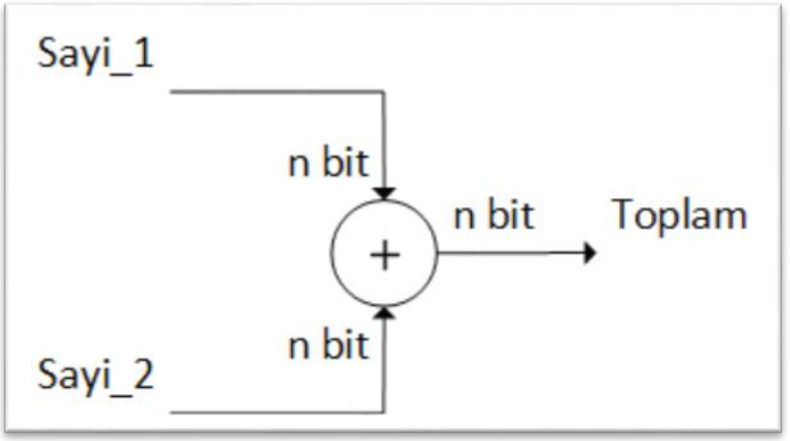
- Aritmetik İşlem Birimlerin Gerçeklenmesi
- Hafıza Birimlerinin Gerçeklenmesi
- Sinyal İşleme Uygulaması : 1 Boyutlu Konvolüsyon
  - MATLAB ile Filtre Katsayılarının Oluşturulması
  - Katsayıların Tam Sayı Formatına Dönüştürülmesi
  - MATLAB ve VHDL benzetim çıkıtlarının karşılaştırılması
- Sentez Sonuçları
- Görüntü İşleme Uygulaması : 2 Boyutlu Konvolüsyon
- Uygulama

Aritmetik İşlemler

Toplama İşlemi

$$k = \max(m, n) + 1$$






```
Toplam <= Sayi_1 + sxt(Sayi_2,
Sayi_1'length);
```

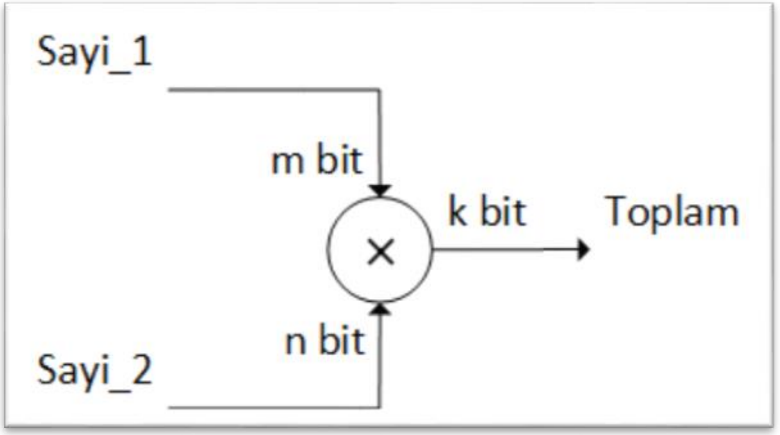
```
Toplam <= add(Sayi_1, Sayi_2);
```

Mehmet Ali Çavuşlu, Cihan Karakuzu, Suhap Şahin, Mehmet Yakut, "Neural Network Training Based on FPGA with Floating Point Number Format and It's Performance", Neural Computing & Application, Volume 20, Number 2,Pages: 195-202, March 2011

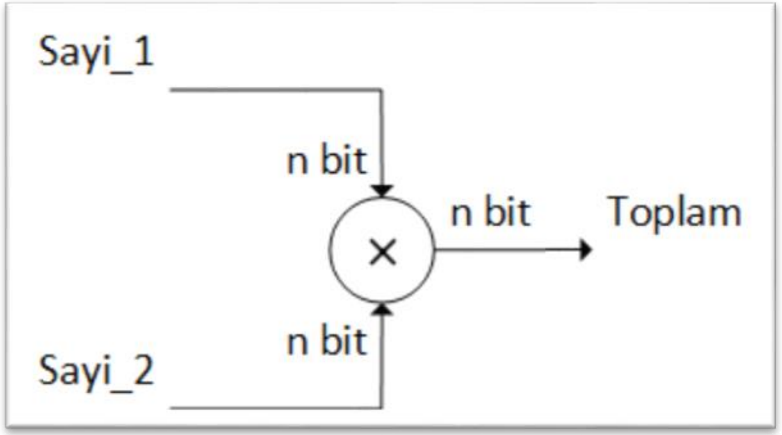
Aritmetik İşlemler

Çarpma İşlemi

$$k = m + n$$




```
Carpim <= Sayi_1 * Sayi_2;
```




```
Carpim <= mul(Sayi_1, Sayi_2);
```

Mehmet Ali Çavuşlu, Cihan Karakuzu, Suhap Şahin, Mehmet Yakut, "Neural Network Training Based on FPGA with Floating Point Number Format and It's Performance", Neural Computing & Application, Volume 20, Number 2,Pages: 195-202, March 2011

Aritmetik İşlemler

Tam Sayılarda 2^N ile Çarpma İşlemi

```
Carpim <= data & conv_std_logic_vector(0, N);
```

Kayan Noktalı Sayılarda 2^N ile Çarpma İşlemi

```
Carpim <= data_s & (data_e + N) & data_f;
```

Tam Sayılarda 2^N ile Bölme İşlemi

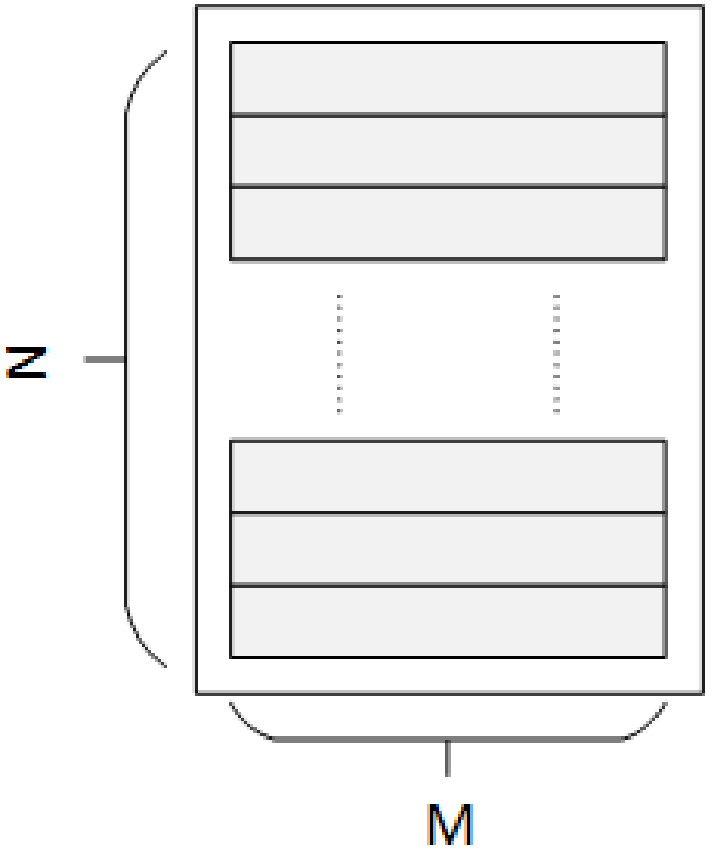
```
Bolum <= data(data'high downto N);
```

Kayan Noktalı Sayılarda 2^N ile Bölme İşlemi

```
Bolum <= data_s & (data_e - N) & data_f;
```

Hafıza Biriminin Oluşturulması

```
type t_hafiza is array (0 to N-1) of
     std_logic_vector(M-1 downto 0);
signal r_hafiza : t_hafiza :=
    (others => (others => '0'));
```



1 Boyutlu Konvolüsyon

- Konvolusyon, giriş sinyali ve doğrusal sistemin dürtü tepki (impulse response) fonksiyonu bilindiğinde çıkış işaretini bulmaya yarayan bir işlemdir.
- Sistemin dürtü tepki fonksiyonu h[n]'in N tane çarpanlı bir sonlu filtre olduğunu varsayarsak, giriş işareti x[n] sonsuz uzunlukta olduğu durumda dahi, filtrenin çıkış işareti y[n] konvolüsyon işlemi :

$$y[n] = x[n] * h[n] = \sum_{k=0}^{N-1} h[k]x[n-k]$$

1 Boyutlu Konvolüsyon

N. dereceden filtrenin katsayısı N + 1 adettir. Gerçekleme aşamasında tepki fonksiyonu için uzunluğu (N + 1) olan bir hafıza oluşturulması gerekmektedir.

```
type t_coef is array (0 to N) of
              std_logic_vector(COEF_LENGTH - 1 downto 0);
signal r_coef : t_coef := (others => (others => '0'));
```

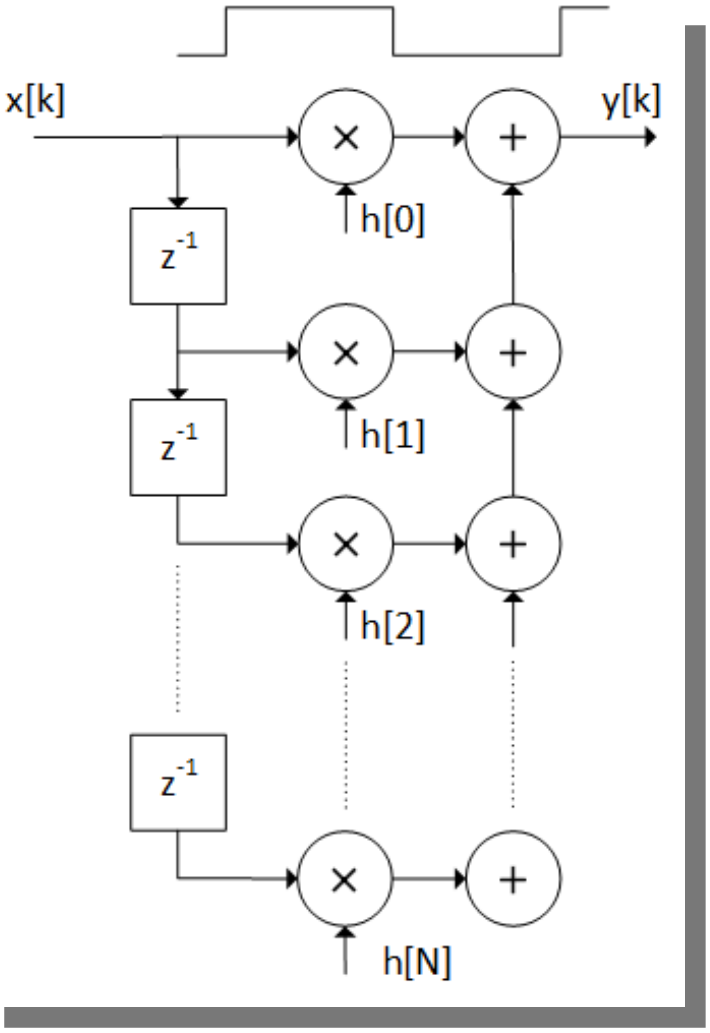
Konvolüsyon işleminin yapılabilmesi için (N + 1) uzunluğunda hafıza biriminde giriş değerleri saklanmaktadır

```
type t_data is array (0 to N) of
          std_logic_vector(DATA_LENGTH - 1 downto 0);
signal r_data : t_data := (others => (others => '0'));
```

1 Boyutlu Konvolüsyon

Tek Saat Darbesi ile Gerçekleme

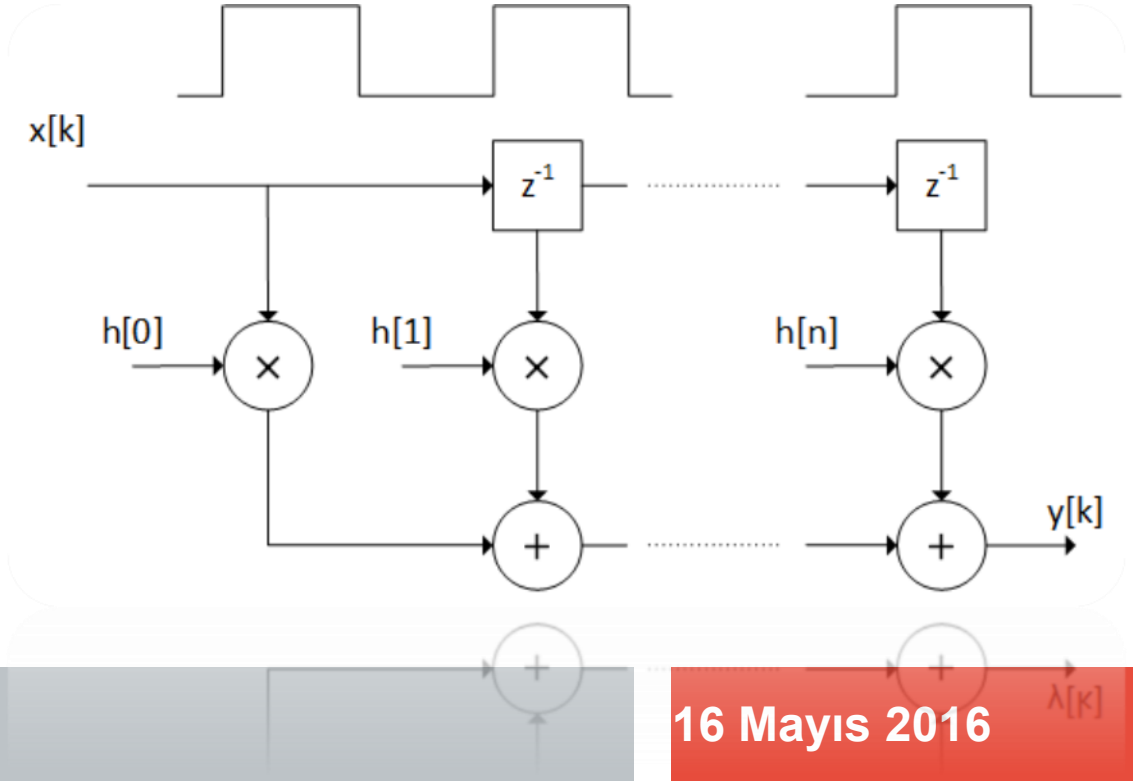
```
Sum_Conv := 0;
for n_i in 0 to N loop
    Sum_Conv := Sum_Conv +
    r_data(n_i) * r_coef(N - n_i);
end loop;
return (Sum_Conv);
```



1 Boyutlu Konvolüsyon

Sıralı Gerçekleme

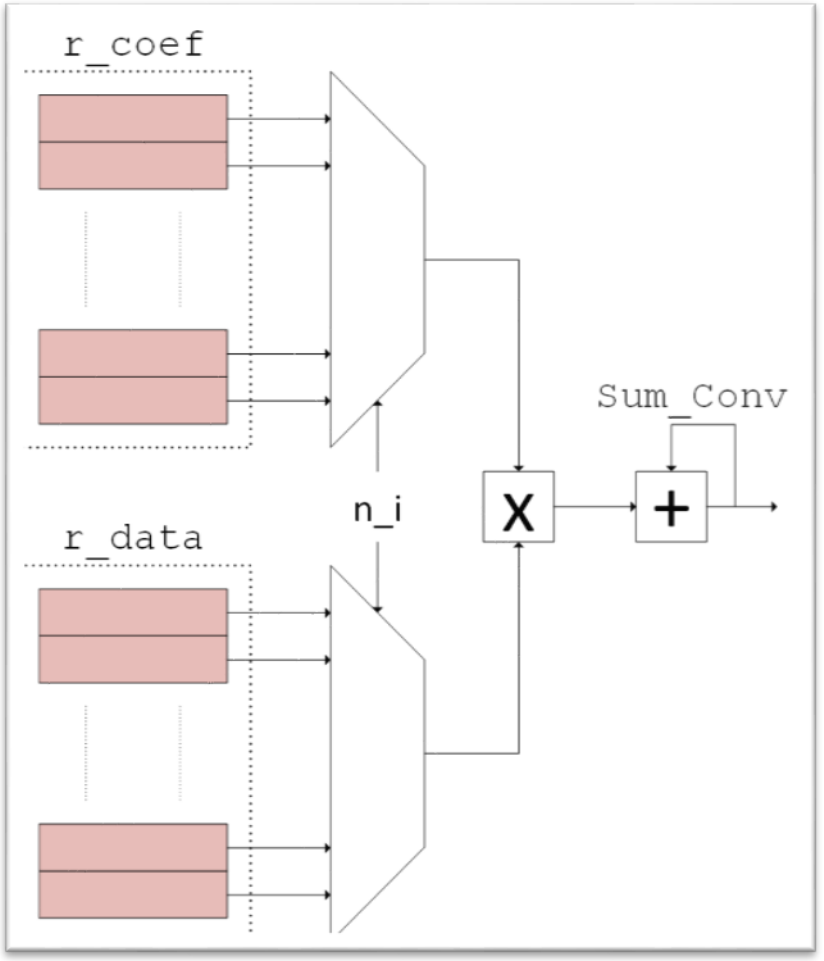
```
when FILTER_CALC =>
   Sum_Conv <= Sum_Conv + r_data(n_i)* r_coef(N - n_i);
   n_i <= n_i + 1;
   if n_i = N then
      n_i <= 0;
      r_Cntrl <= DONE;
   end if;
```



1 Boyutlu Konvolüsyon

Sıralı Gerçekleme

```
when FILTER_CALC =>
   Sum_Conv <= Sum_Conv + r_data(n_i)*
   r_coef(N - n_i);
   n_i <= n_i + 1;
   if n_i = N then
     n_i <= 0;
     r_Cntrl <= DONE;
   end if;
```



1 Boyutlu Konvolüsyon

Konvolüsyon işleminin donanımsal gerçeklenmesine ilişkin FIR filtre uygulaması anlatılacaktır. Uygulama adımları aşağıdaki gibidir:

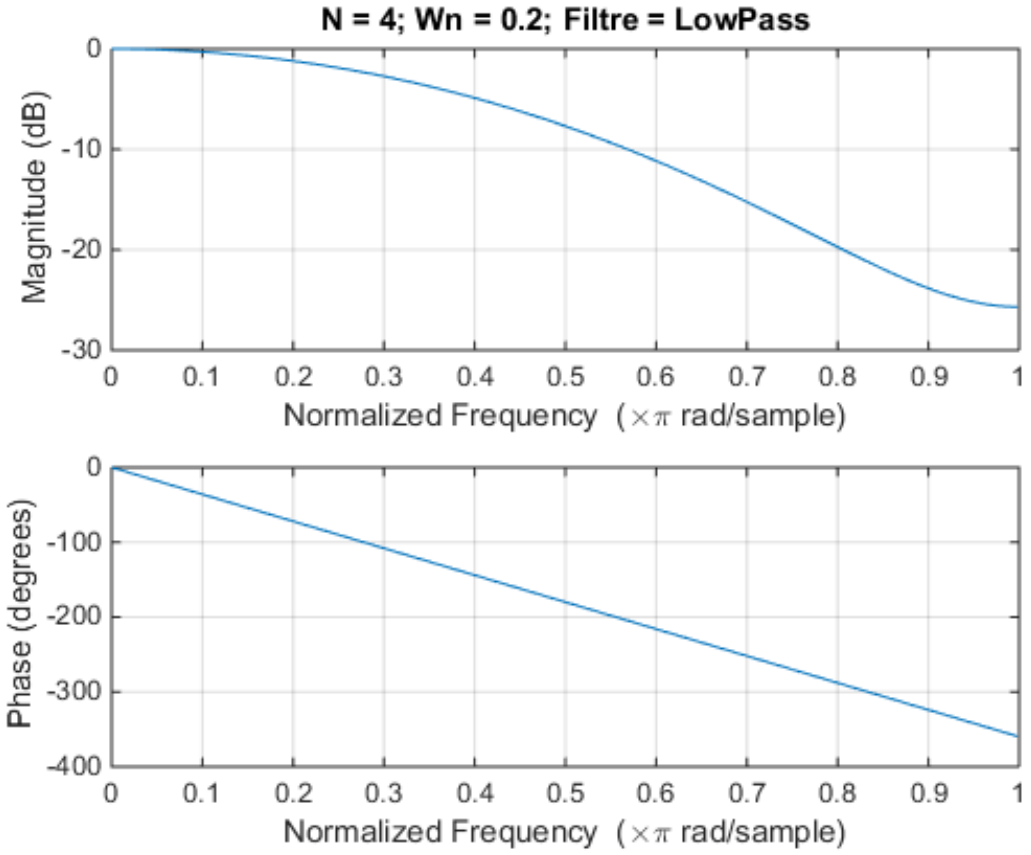
- Uygulama aşamasında öncelikli olarak MATLAB ortamında filtre katsayılarının elde edilmesi,
- Katsayıların tam sayı ve kayan noktalı sayı formatında ifade edilmesi,
- Tam sayı ve kayan noktalı sayılarda gerçekleme,
- Örnek bir uygulama üzerinde tam sayı formatı ile double sayı formatında elde edilen sonuçlar ve FPGA'da elde edilen sonuçların karşılaştırılması

FIR Filtre Katsayılarının Oluşturulması

MATLAB ile FIR Filtre Katsayıların Oluşturulması

$$FilterCoef = fir1(N, Wn, Filtre);$$

```
h[0] = 0,0284;
h[1] = 0,2370;
h[2] = 0,4691;
h[3] = 0,2370;
h[4] = 0,0284;
```



FIR Filtre Katsayılarının Oluşturulması

Katsayıların Kayan Noktalı Sayı Formatında Gösterimi

	Katsayı	Kayan Noktalı Sayı		
		İşaret Biti	Üstel İfade	Kesirli İfade
h[0]	0,0284	0	01111001	11010001010011100011110
h[1]	0,2370	0	01111100	11100101011000000100001
h[2]	0,4691	0	01111101	11100000010110111100000
h[3]	0,2370	0	01111100	11100101011000000100001
h[4]	0,0284	0	01111001	11010001010011100011110

FIR Filtre Katsayılarının Dönüştürülmesi

Katsayıların Sabit Noktalı Sayı Formatında Gösterimi

	Katsayı	Sabit Noktalı Sayı		
		İşaret Biti	Tam Kısım	Ondalık Kısım
h[0]	0,0284	0	00000000	00000111010001010011100
h[1]	0,2370	0	00000000	00111100101011000000100
h[2]	0,4691	0	00000000	01111000000101101111000
h[3]	0,2370	0	00000000	00111100101011000000100
h[4]	0,0284	0	00000000	00000111010001010011100

FIR Filtre Katsayılarının Dönüştürülmesi

Katsayıların Tam Sayı Formatında Gösterimi

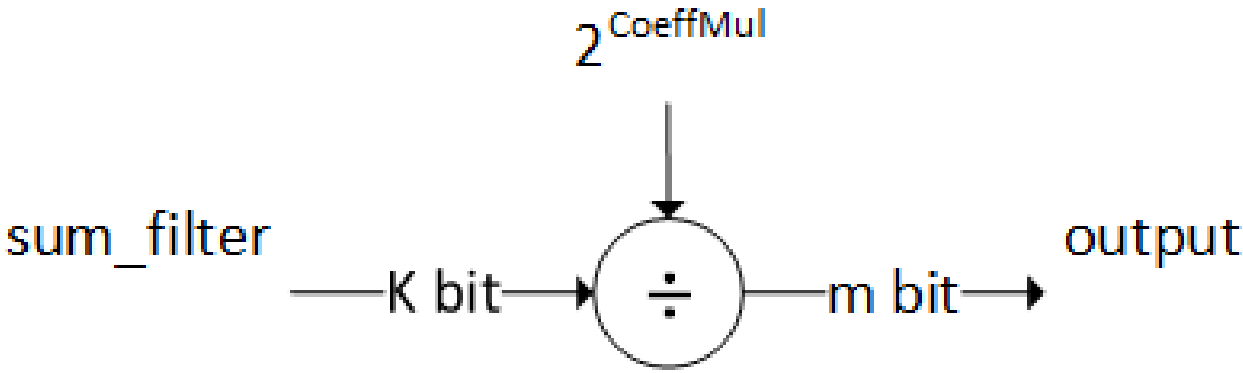
$$FilterCoeffInt = fix(2^{CoefMul} \times FilterCoef)$$

CoefMul = 7	Katsayı	Sabit Noktalı Sayı	
		İşaret Biti	Tam Kısım
$fix(h[0] \times 2^7)$	3	0	0000011
$fix(h[1] \times 2^7)$	30	0	0011110
$fix(h[2] \times 2^7)$	60	0	0111100
$fix(h[3] \times 2^7)$	30	0	0011110
$fix(h[4] \times 2^7)$	3	0	0000011

FIR Filtre Katsayılarının Dönüştürülmesi

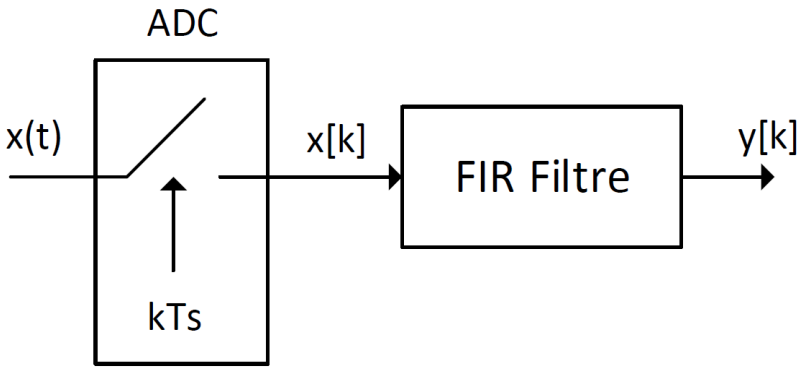
Filtre Çıkışının Normalize Edilmesi

$$m = K - CoefMul$$

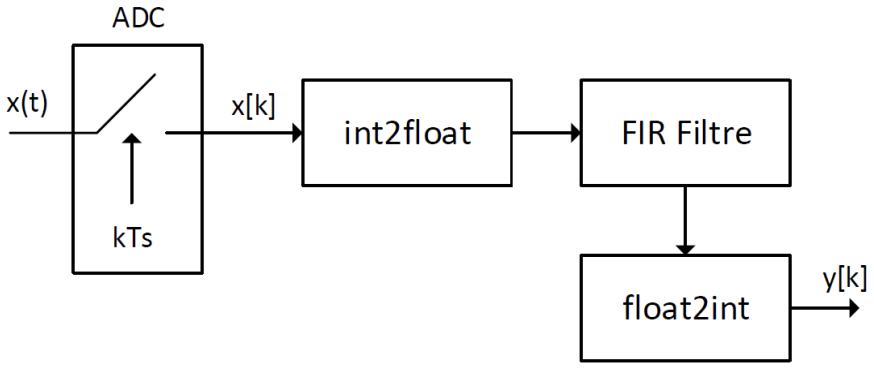


Sayı Formatına Göre Gerçekleme Aşamaları

Tam Sayı Formatında Gerçekleme



Kayan Noktalı Sayı Formatında Gerçekleme

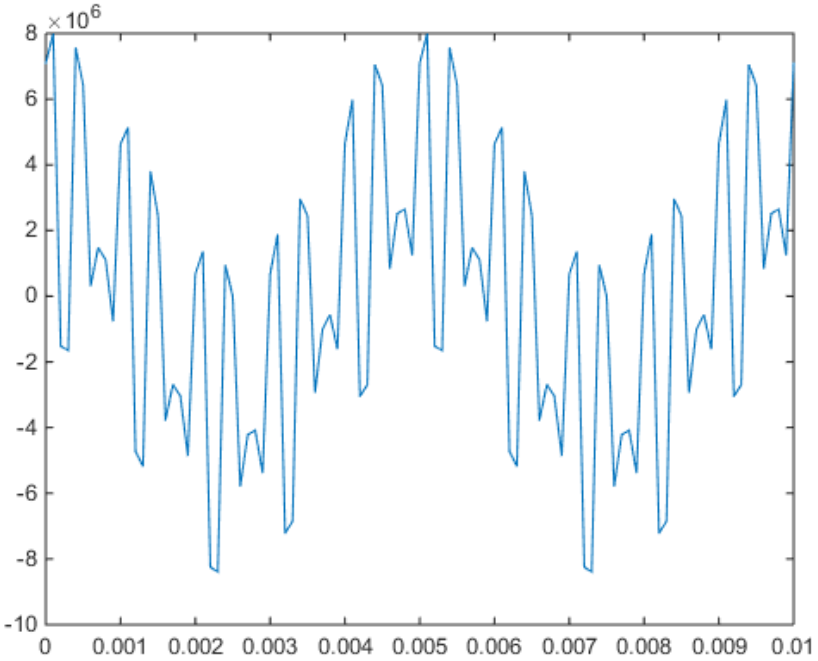


Örnek Uygulama

Giriş Sinyalinin Oluşturulması :

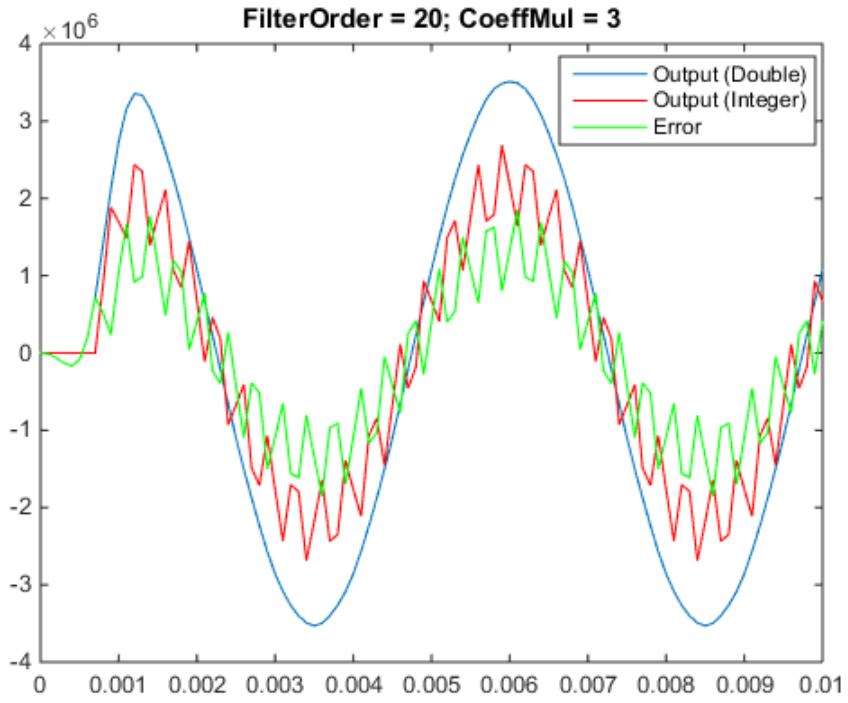
$$Input(t) = \cos\left(\frac{2\pi f_1 t}{f_s}\right) + \cos\left(\frac{2\pi f_2 t}{f_s}\right) + sin\left(\frac{2\pi f_3 t}{f_s}\right)$$

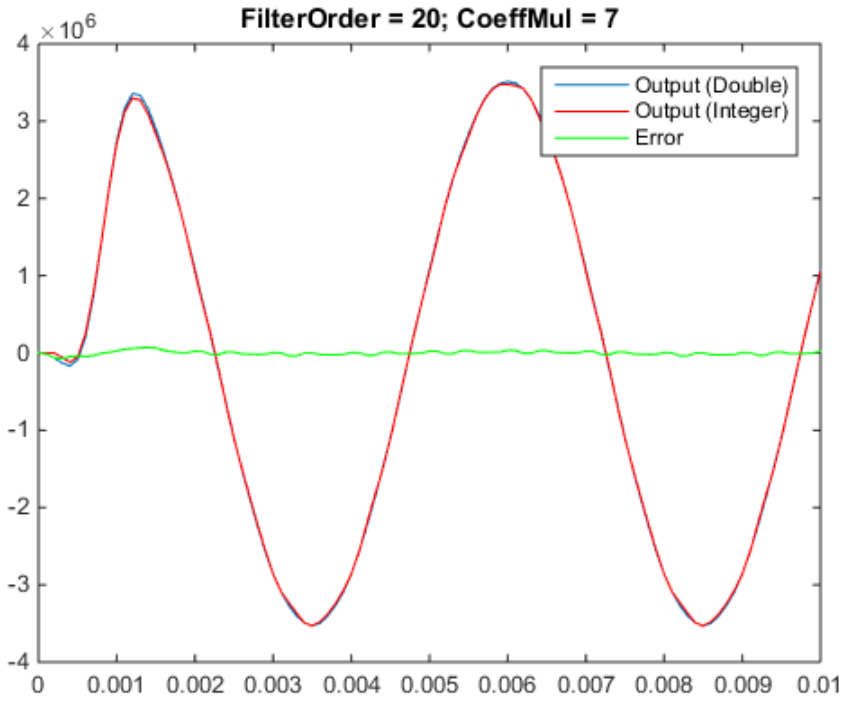
```
f_s = 10000; f_1 = 200;
f_2 = 2000; f_3 = 3000;

n_t = 0 : 1 / f_s : 0.01 ;

Input = cos(2 * pi * f_1 * n_t) +
cos(2 * pi * f_2 * n_t) + sin(2 * pi *
f_3 * n_t);
```



Sayı Formatlarının Karşılaştırılması

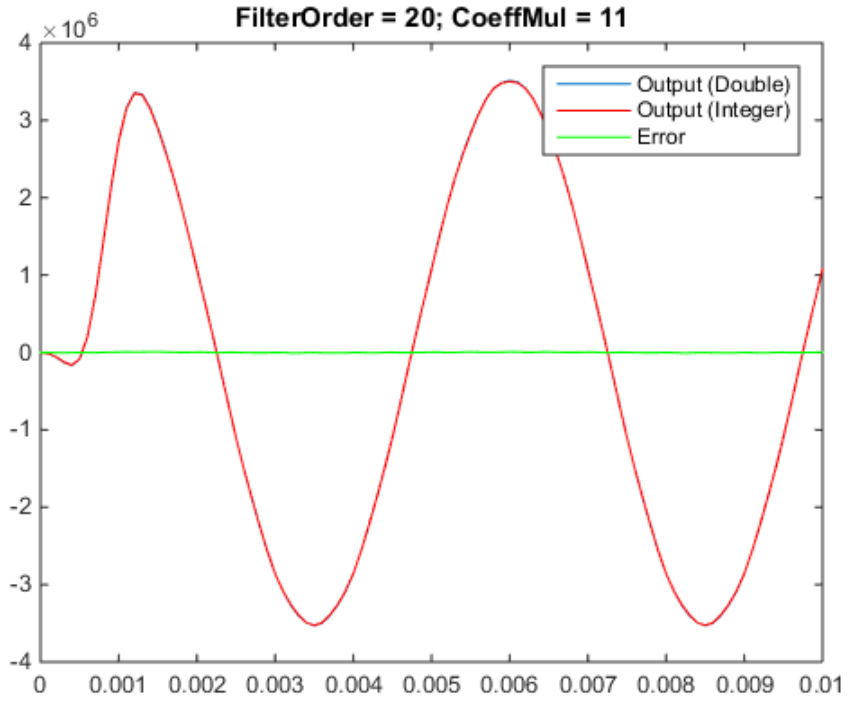
MATLAB ortamında gerçekleştirilen karşılaştırmalar:

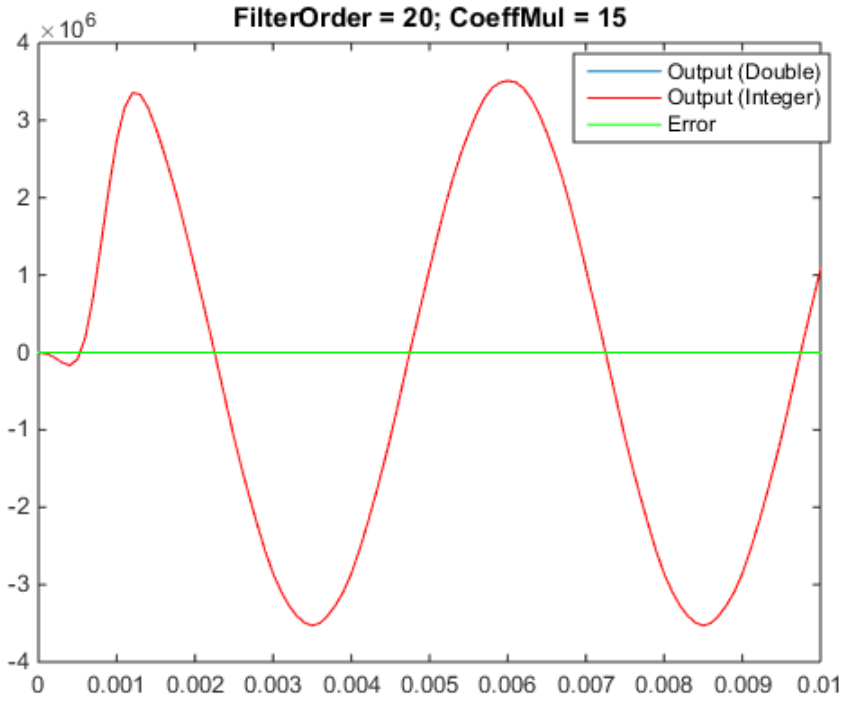




Sayı Formatlarının Karşılaştırılması

MATLAB ortamında gerçekleştirilen karşılaştırmalar:





Karşılaştırma

CoefMul	Ortalama Karesel Hata
	MATLAB_d vs MATLAB_i
3	811742.692435
5	174845.619207
7	19585.309669
9	9564.304627
11	3602.174860
13	754.047501
15	119.116878
19	12.333919
23	0.625085
27	0.049680

MATLAB_d : Double sayı formatında FIR filtre uygulamasının MATLAB ile gerçeklenmesi

MATLAB_i : Farklı CoefMul katsayı değerler, için tam sayı formatında FIR filtre uygulamasının MATLAB ile gerçeklenmesi

FPGA tabanlı gerçekleştirme

Genel (Generic) Parametreler

•**FILTER_ORDER : Filtre katsayılarının ve giriş verilerinin saklanması için gerekli hafıza** birimlerinin uzunluklarını belirlemektedir.

•**COEFF_LENGTH :** Filtre katsayılarının saklanması için gerekli hafıza birimlerinin derinliğini tanımlamaktadır.

•**DATA_LENGTH :** Giriş verilerinin saklanması için gerekli hafıza birimlerinin derinliğini tanımlamaktadır.

•**COEFF_MUL :** Filtre çıkışlarının normalize değeri tanımlanmaktadır.

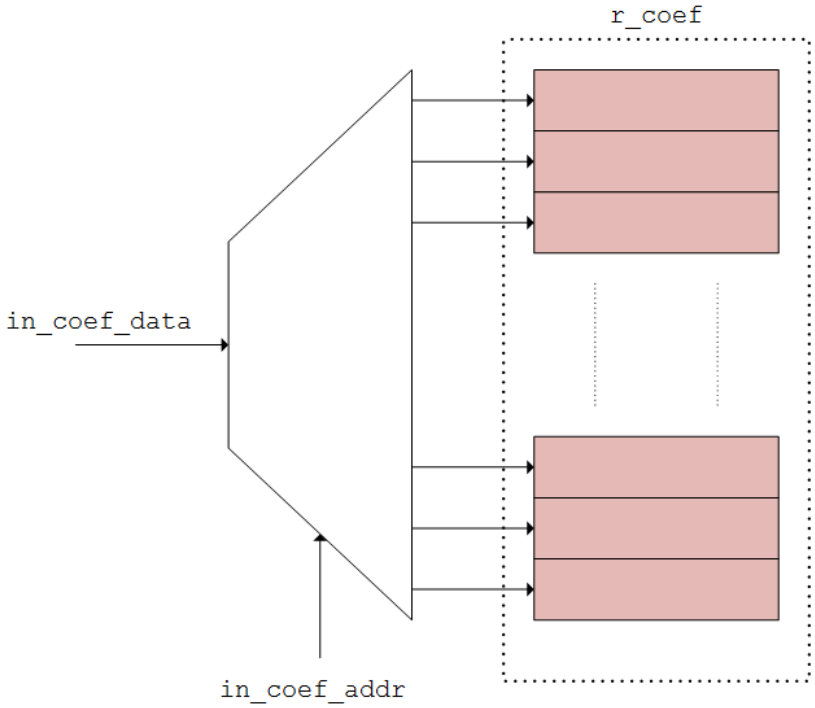
FPGA tabanlı gerçekleştirme

```
Generic(
   DATA_LENGTH  : integer;
   FILTER_ORDER : integer;
   COEFF_LENGTH : integer;
   COEFF_MUL    : integer
);
Port(
   in_clk : in std_logic;
   in_rst : in std_logic;
   in_coef_vld : in std_logic;
   in_coef_addr : in std_logic_vector(log2i(FILTER_ORDER) downto 0);
   in_katsayi_data : in std_logic_vector(COEFF_LENGTH - 1 downto 0);
   in_data : in std_logic_vector(DATA_LENGTH - 1 downto 0);
   in_data_vld : in std_logic;
   out_data : out std_logic_vector(DATA_LENGTH - 1 downto 0);
   out_data_vld : out std_logic
);
```

FPGA tabanlı gerçekleştirme

Katsayıların Hafızaya Yazılması

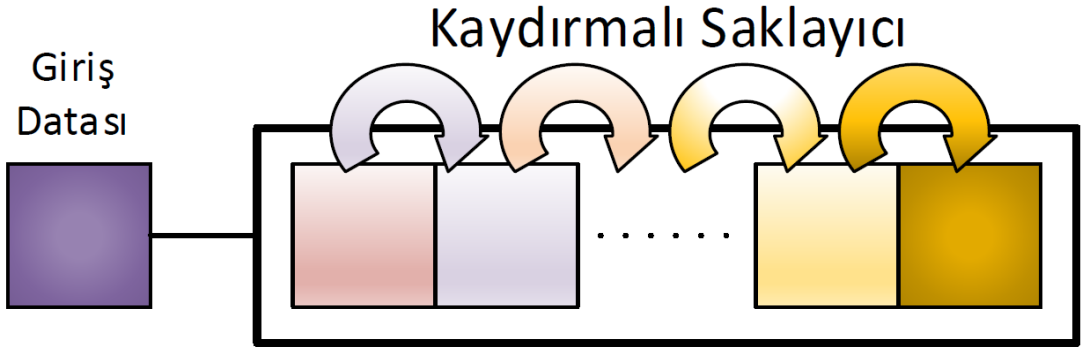
```
process(in_clk, in_rst)
begin
  if in_rst = '1' then
    r_coef <= (others => (others => '0'));

  elsif rising_edge(in_clk) then
     if in_coef_vld = '1' then
       r_coef(conv_integer(in_coef_addr))
       <= in_coef_data;
     end if;
  end if;
end process;
```



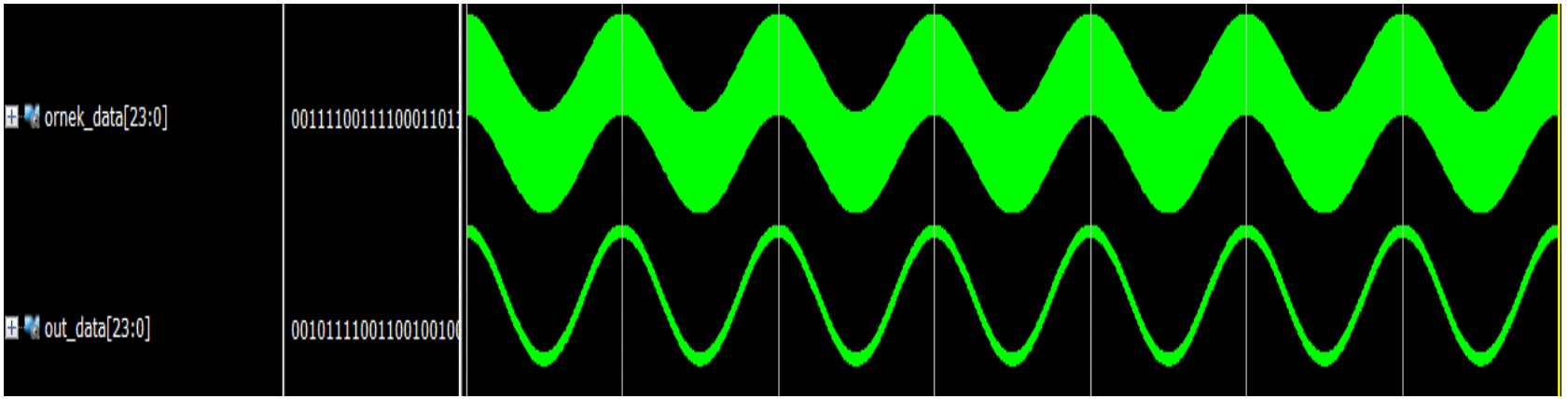
FPGA tabanlı gerçekleştirme

Giriş Verilerinin Saklanması

```
v_data := r_data;
for n_i in N - 2 downto 0 loop
  v_data (n_i + 1) := v_data(n_i);
end loop;
v_data (0) := in_data;
return v_data;
```



FPGA tabanlı gerçekleştirme



Karşılaştırma

CoeffMul	Ortalama Karesel Hata		
	MATLAB_d vs MATLAB_i	MATLAB_d vs FPGA	MATLAB_i vs FPGA
3	811742.692435	811742.692435	0
5	174845.619207	174845.619207	
7	19585.309669	19585.309669	
9	9564.304627	9564.304627	
11	3602.174860	3602.174860	
13	754.047501	754.047501	
15	119.116878	119.116878	
19	12.333919	12.333919	
23	0.625085	0.625085	
27	0.049680	0.049680	

MATLAB_d : Double sayı formatında FIR filtre uygulamasının MATLAB ile gerçeklenmesi
MATLAB_i : Farklı CoefMul katsayı değerler, için tam sayı formatında FIR filtre uygulamasının MATLAB ile gerçeklenmesi

Sentez Sonuçları