Tahmini Ders İçeriği
(Tentative Couse Schedule – Syllabus)

1. Hafta: Sayısal Sinyaller/Sistemler, İkilik Tabanda Sayılar, Taban Aritmetiği, İşaretli/Eksi Sayıların Gösterimi, Sayısal Tasarım Tarihçesi

2. Hafta: İkili Mantık Aritmetiği ve Kapıları, Bool Cebiri Teorisi ve Tanımları, Bool Fonksiyonları, Kapı-Seviyesinde Yalınlaştırma, Karnough Haritası, Önemsenmeyen Durumlar, NAND, NOR, XOR

3-4. Hafta: FPGA, Birleşik (Combinational) Devreler, Aritmetik Modüller, Decoder, Encoder, Mux, Verilog HDL

Lab Sınavı

5-6. Hafta: Ardışık (Sequential) Devreler, Mandal (Latch), Flip-Flop, Zamanlama (Timing)

Proje Duyurusu

7. Hafta: Durum Makinaları, Örnek Tasarımlar

8. Hafta: Yazmaçlar (Registers), Sayaçlar (Counters)

Ara Sınav

9. Hafta: Bellekler, FPGA'da Block RAM, OpenRAM

10. Hafta: RTL (Register Transfer Level) ASMD (Algorithmic State Machine and Datapath) Tasarımları

11-12. Hafta: Boru hattı, FPGA ve ASIC Tasarım Akışları

Final – Proje Teslimleri

FPGA Nedir?

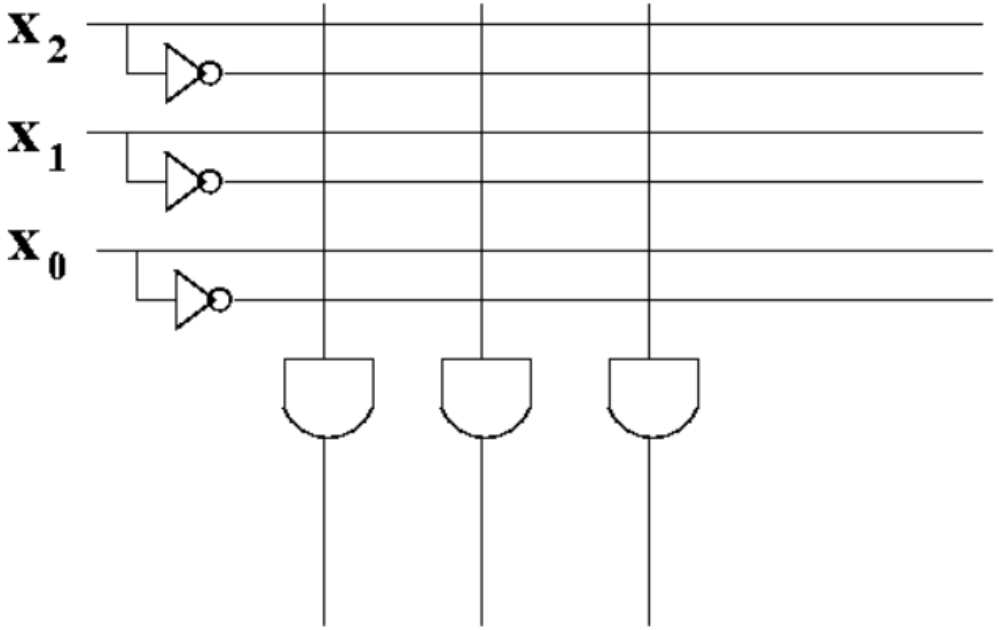
Field Programmable Gate Array – Sahada Programlanabilir Kapı Dizisi

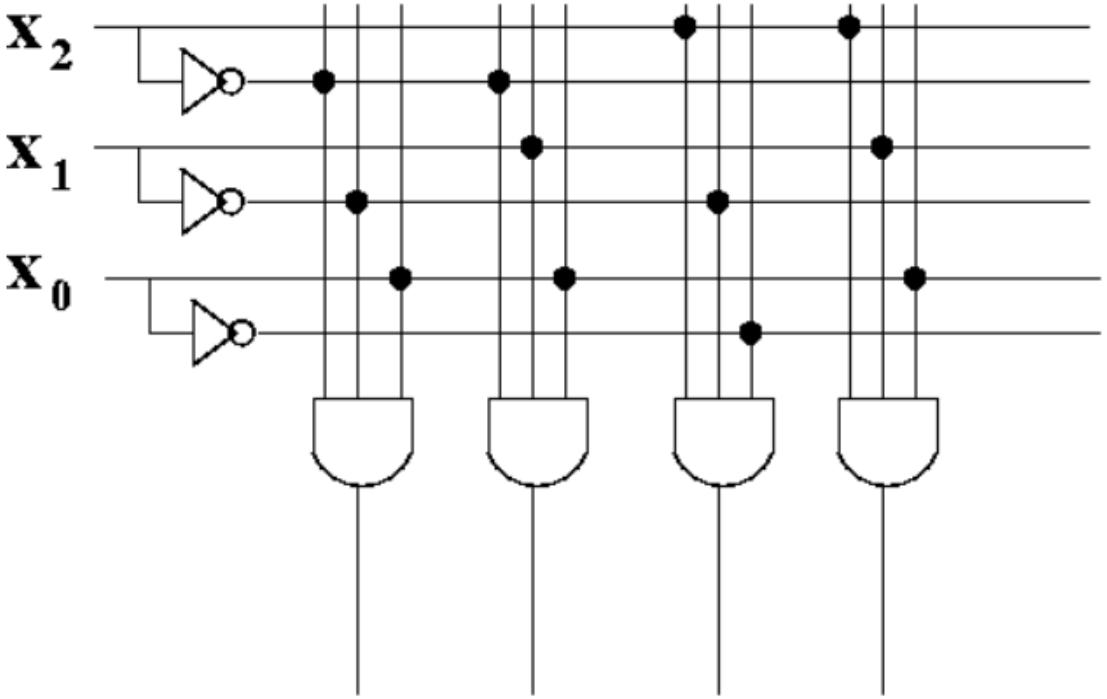
FPGA'dan (1985) önce programlanabilir kapı dizisi olarak ne vardı?

PLA (Programmable Logic Array): Programlanabilir AND ve OR kapıları

Sum of Products (SoP) formda Bool fonksiyonları tasarlanabiliyor

x_2	x_1	x_0	z_1	z_0
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1

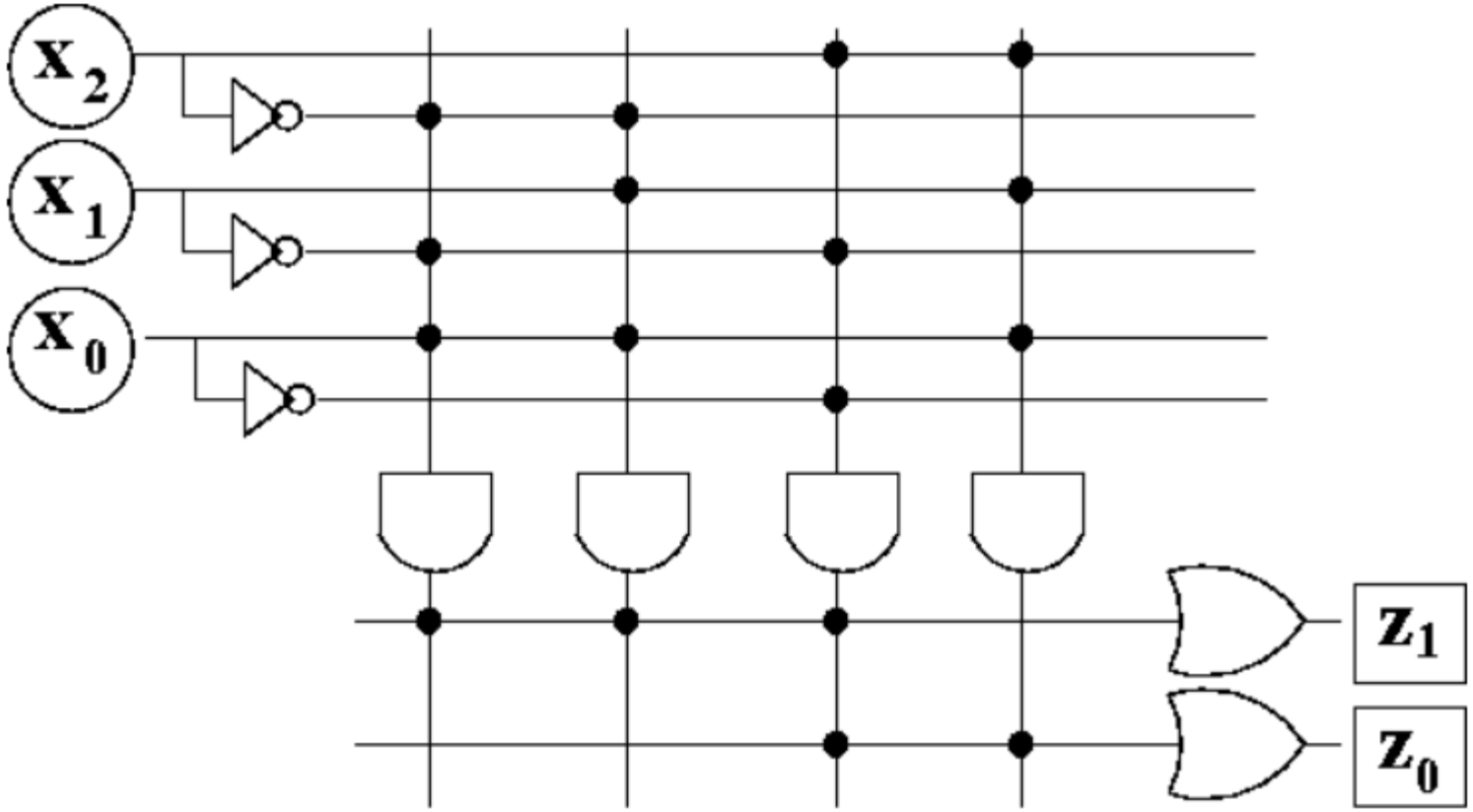




PLA (Programmable Logic Array)

AND ve OR kapılarının girişleri programlanabiliyor (Programmable AND – Programmable OR)

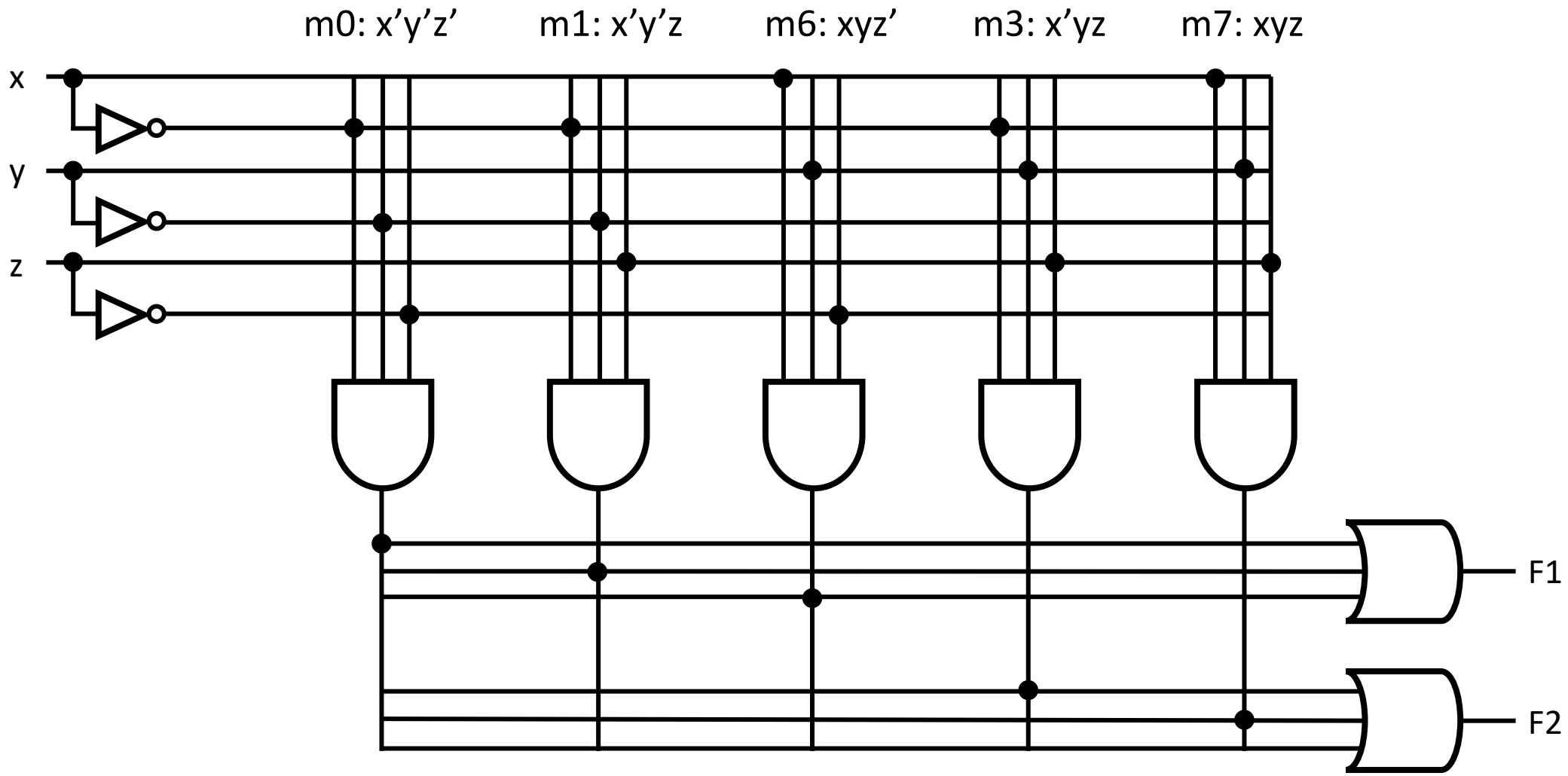
x_2	x_1	x_0	z_1	z_0
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	1
1	0	1	0	0
1	1	0	0	0
1	1	1	0	1



PLA (Programmable Array Logic)

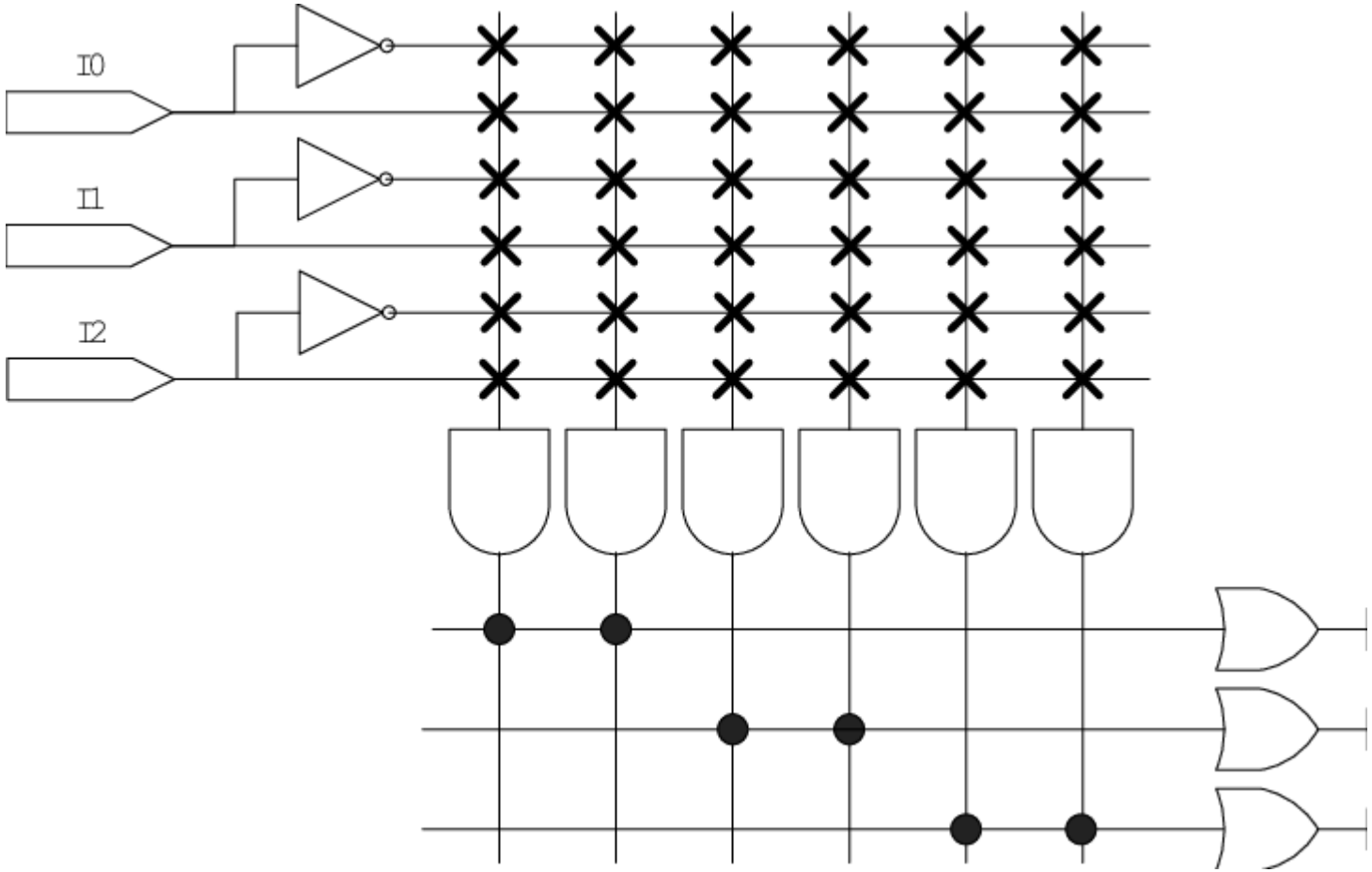
F1 = m0+m1+m6 F2 = m3+m7

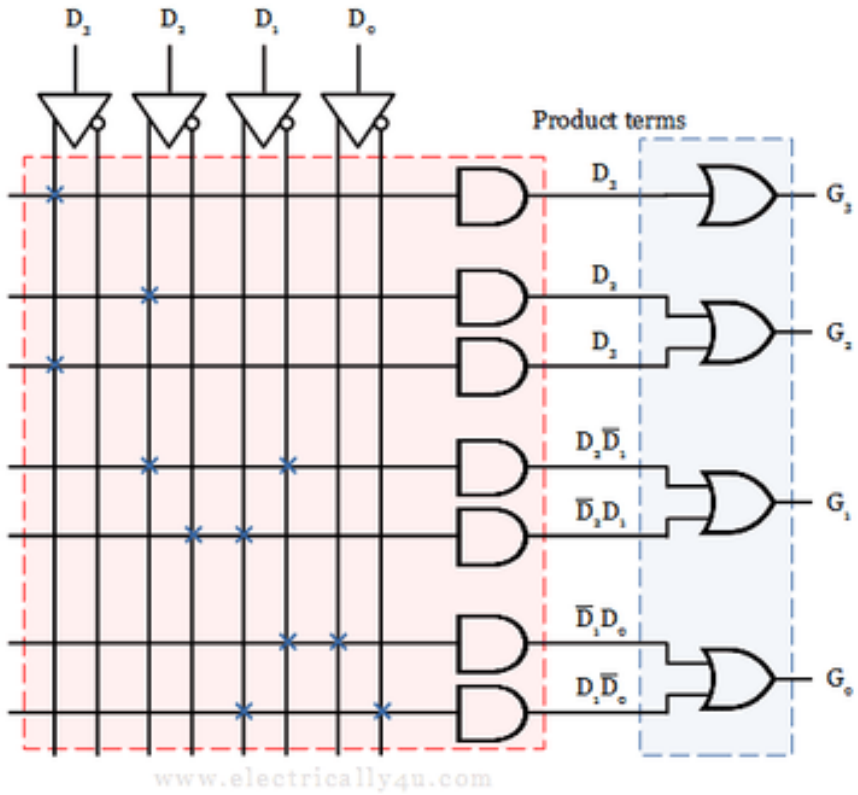
x,y,z giriş sinyalleri olan Bool fonksiyonunu, 5 adet INV 3 adet AND3 ve 2 adet OR3 kapısına sahip PLA ile yapın



PAL (Programmable Array Logic)

AND kapılarının girişleri programlanabiliyor ama OR kapılarının girişleri programlanamıyor (Programmable AND – Fixed OR)





CPLD (Complex Programmable Logic Device)

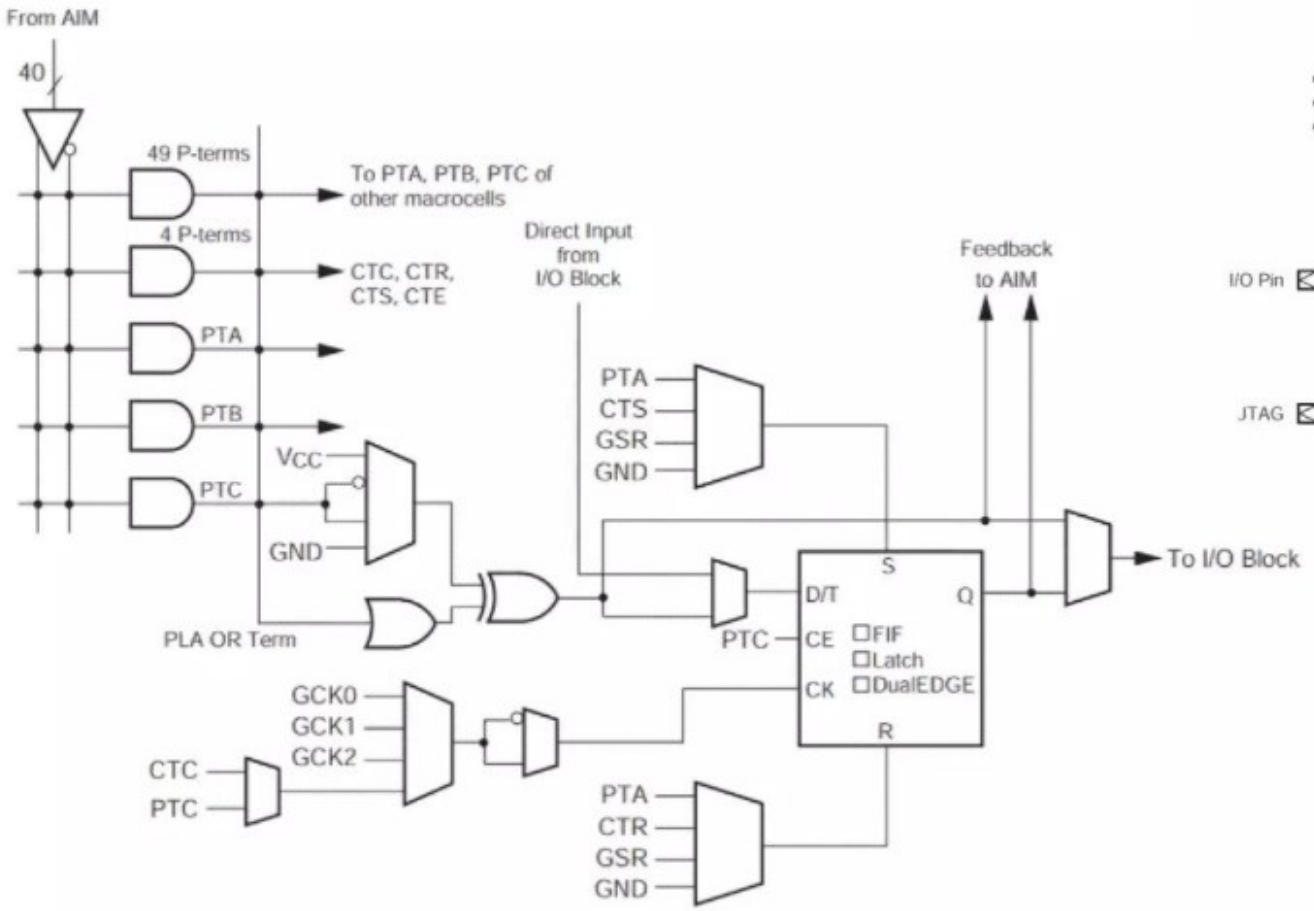


Figure 3: **CoolRunner-II CPLD Macrocell**

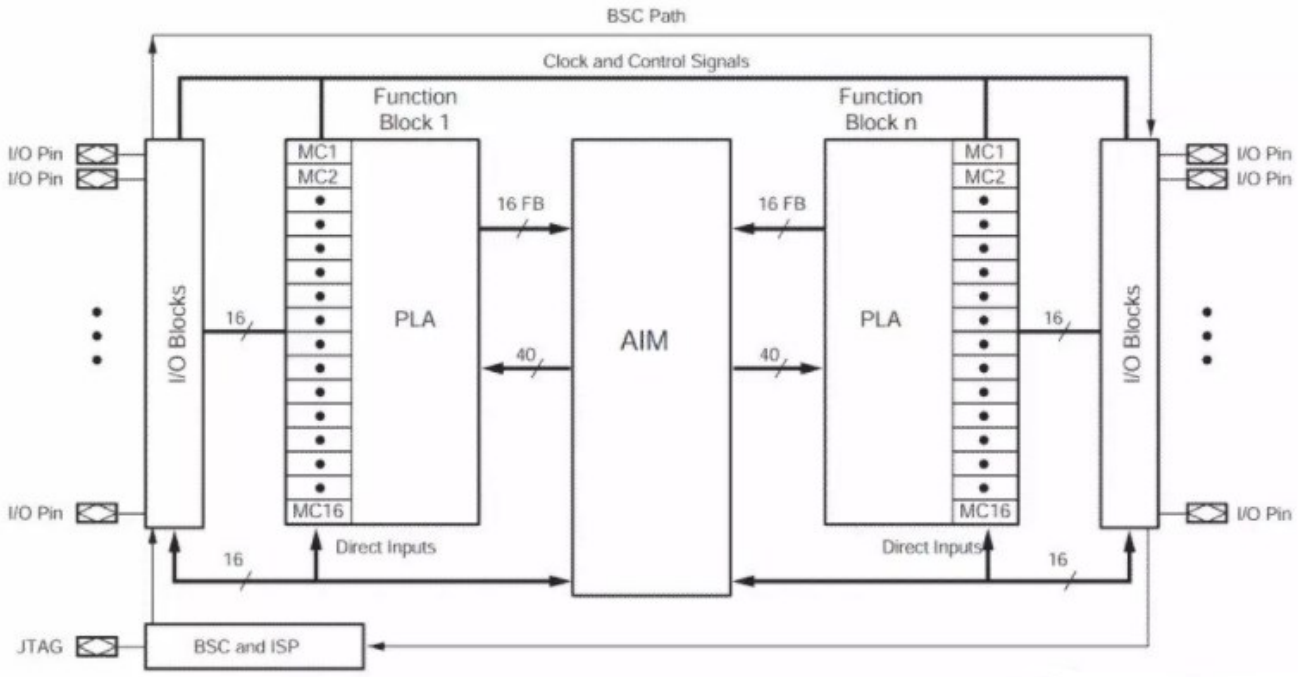
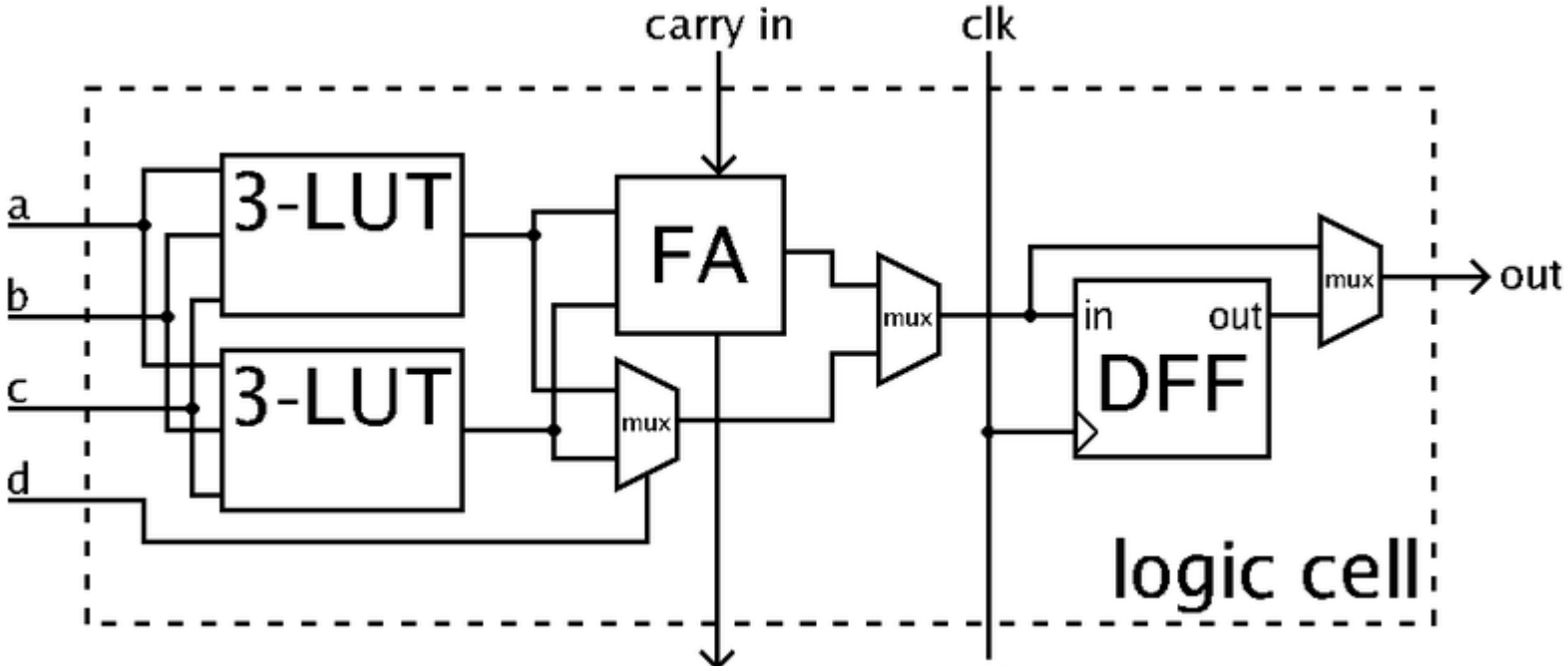
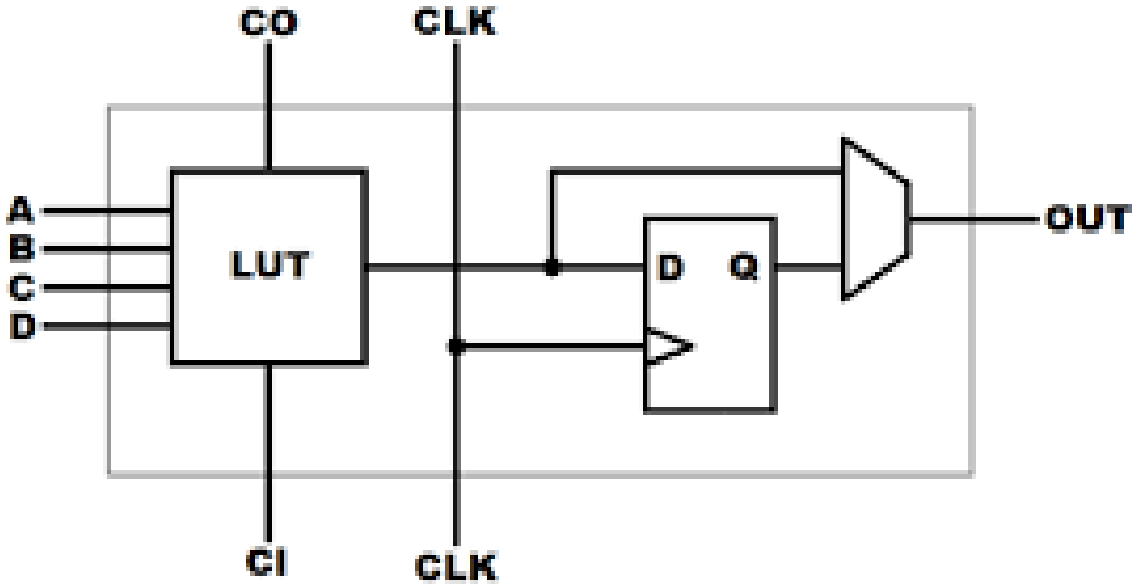


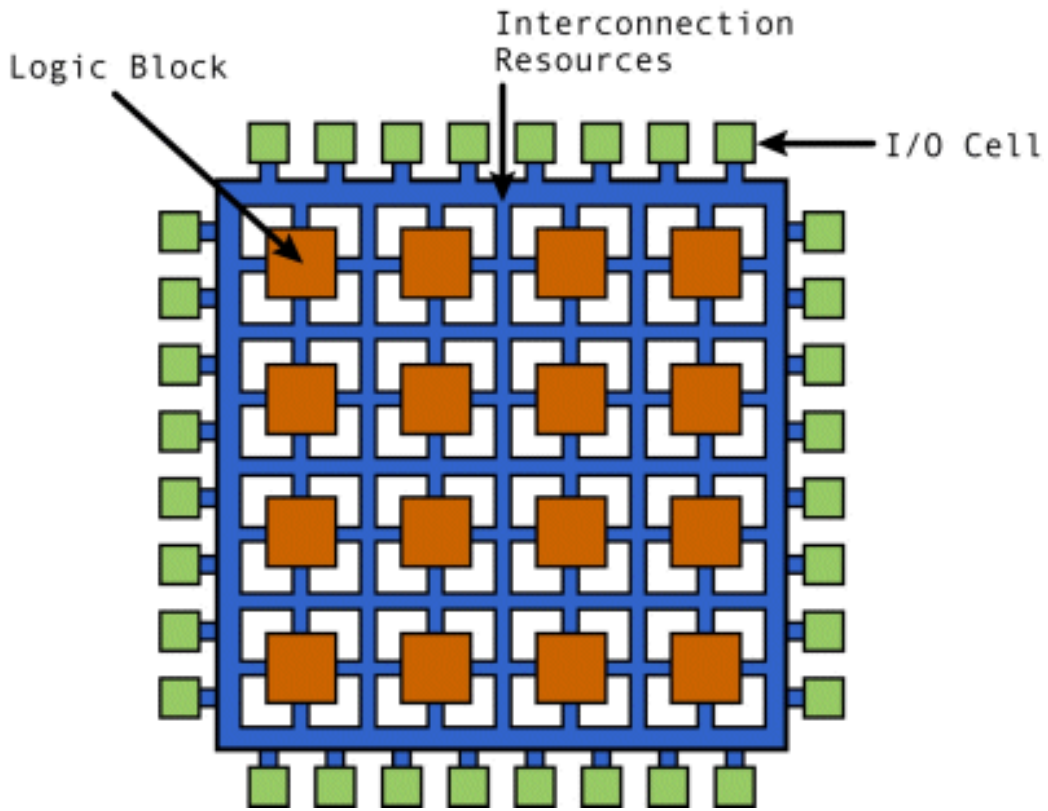
Figure 1: **CoolRunner-II CPLD Architecture**

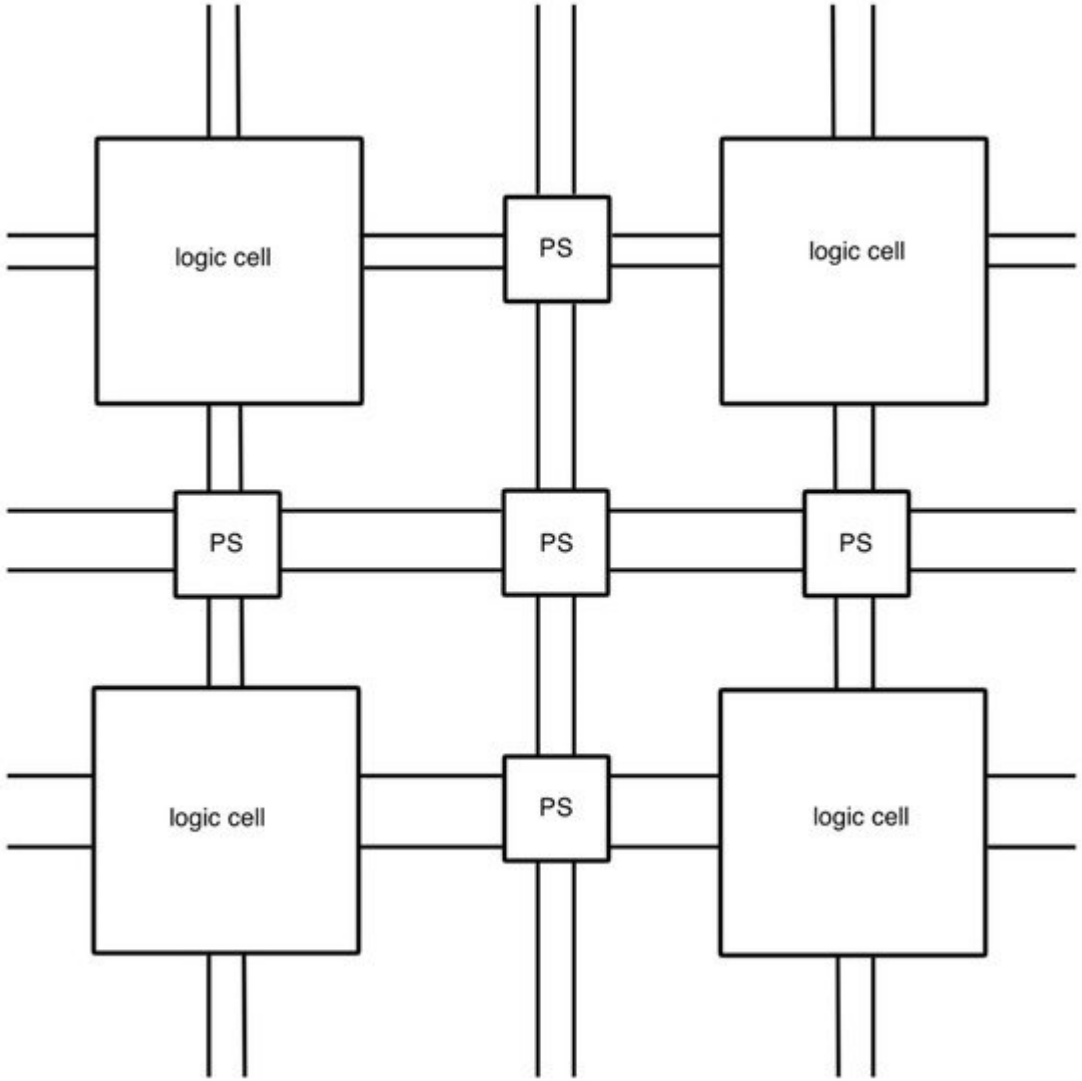
FPGA Mimarisi



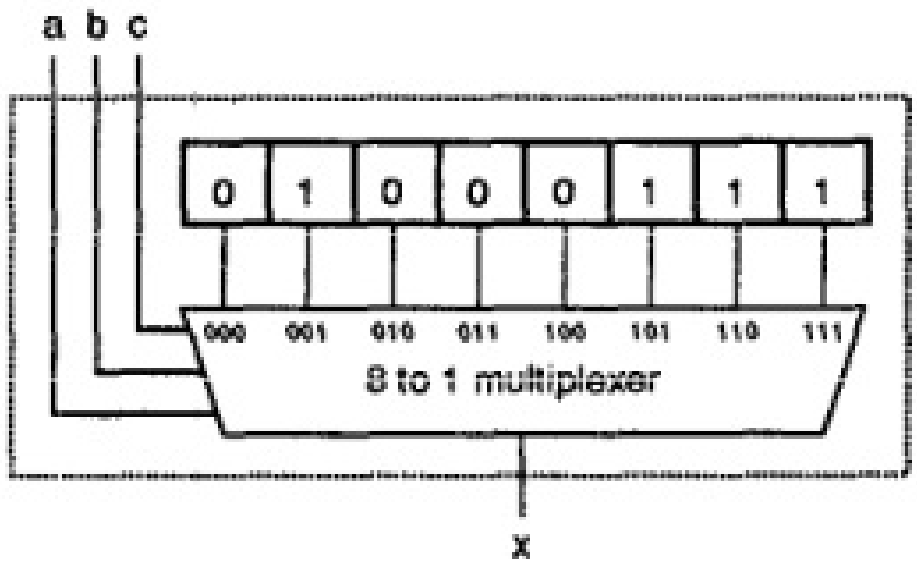


FPGA Mimarisi





FPGA Yapı Taşı: Look-up Table (LUT)



b) 3-Input LUT

Giriş			Çıkış
a	b	c	x
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Herhangi bir Bool fonksiyonu, bir LUT ile gerçekleştirilebilir.

Örnek olarak 3 girişli bu doğruluk tablosu 8-bit'lik bir LUT ile gerçekleştirilmiştir.

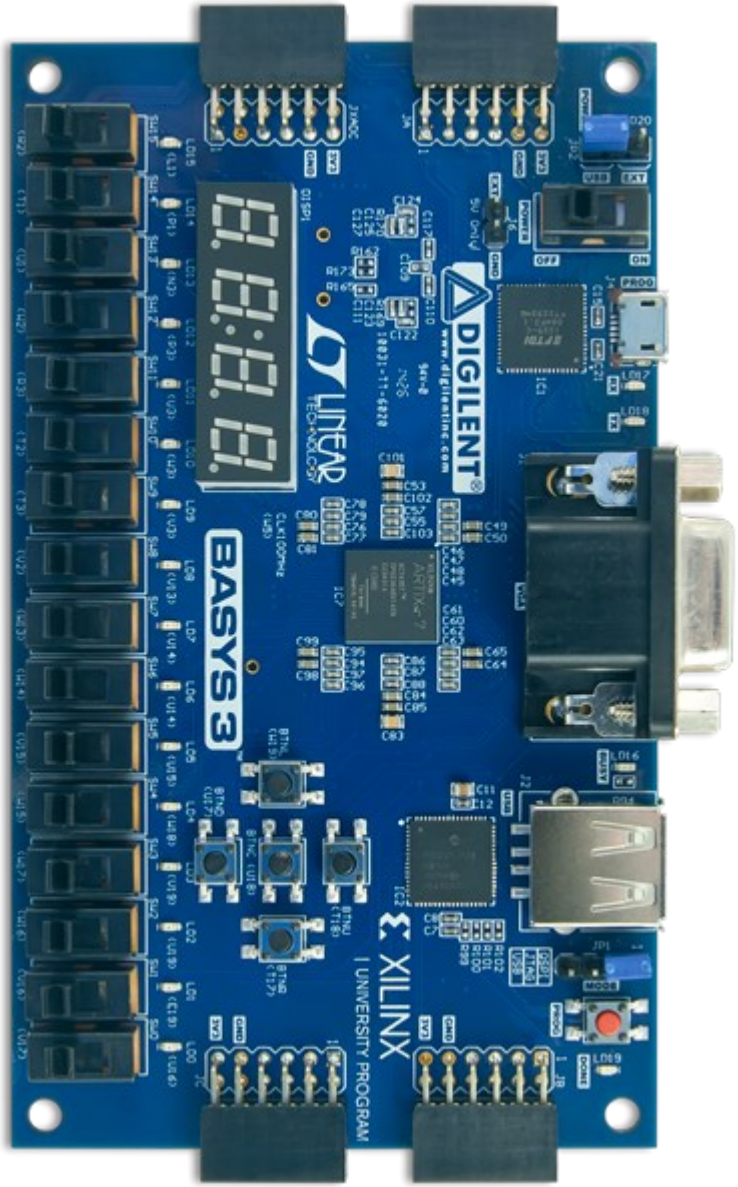
FPGA konfigürasyonu sırasında, kullanılan her bir LUT içerisindeki RAM'in içeriği, gerçekleştirilecek olan fonksiyonun doğruluk tablosuna göre başlangıç değerleri şeklinde yüklenir.

FPGA kaynak kullanımı açısından, Bool fonksiyonunun önemi yoktur, LUT ile herhangi bir fonksiyon tanımlanabilir.

Fonksiyon giriş sinyalleri, RAM içeriğinin adres bitleri olarak işlev görür.

Combinational (sıralı) devreler LUT ile gerçekleşir.

BASYS3 FPGA GELİŞTİRME KARTI



Artix-7 35T features include:

- 33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops);
- 1,800 Kbits of fast block RAM;
- Five clock management tiles, each with a phase-locked loop (PLL);
- 90 DSP slices;
- Internal clock speeds exceeding 450MHz;
- On-chip analog-to-digital converter (XADC).

The Basys 3 also offers an improved collection of ports and peripherals, including:

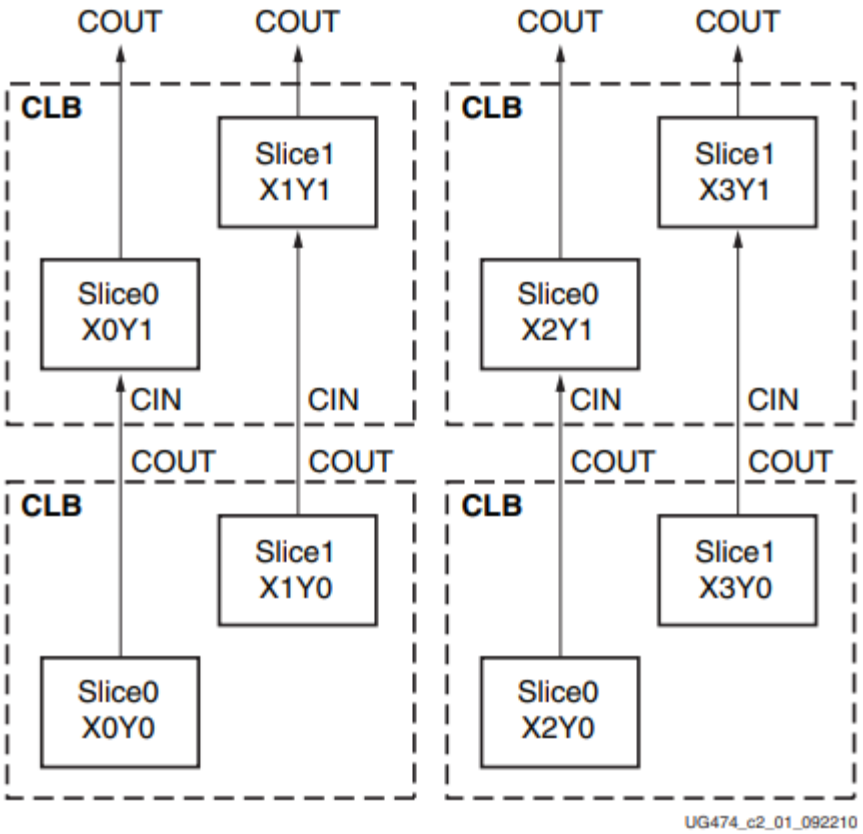
- 16 user switches
- 16 user LEDs
- 5 user pushbuttons
- 4-digit 7-segment display
- Three Pmod ports
- Pmod for XADC signals
- 12-bit VGA output
- USB-UART Bridge
- Serial Flash
- Digilent USB-JTAG port for FPGA programming and communication
- USB HID Host for mice, keyboards and memory sticks

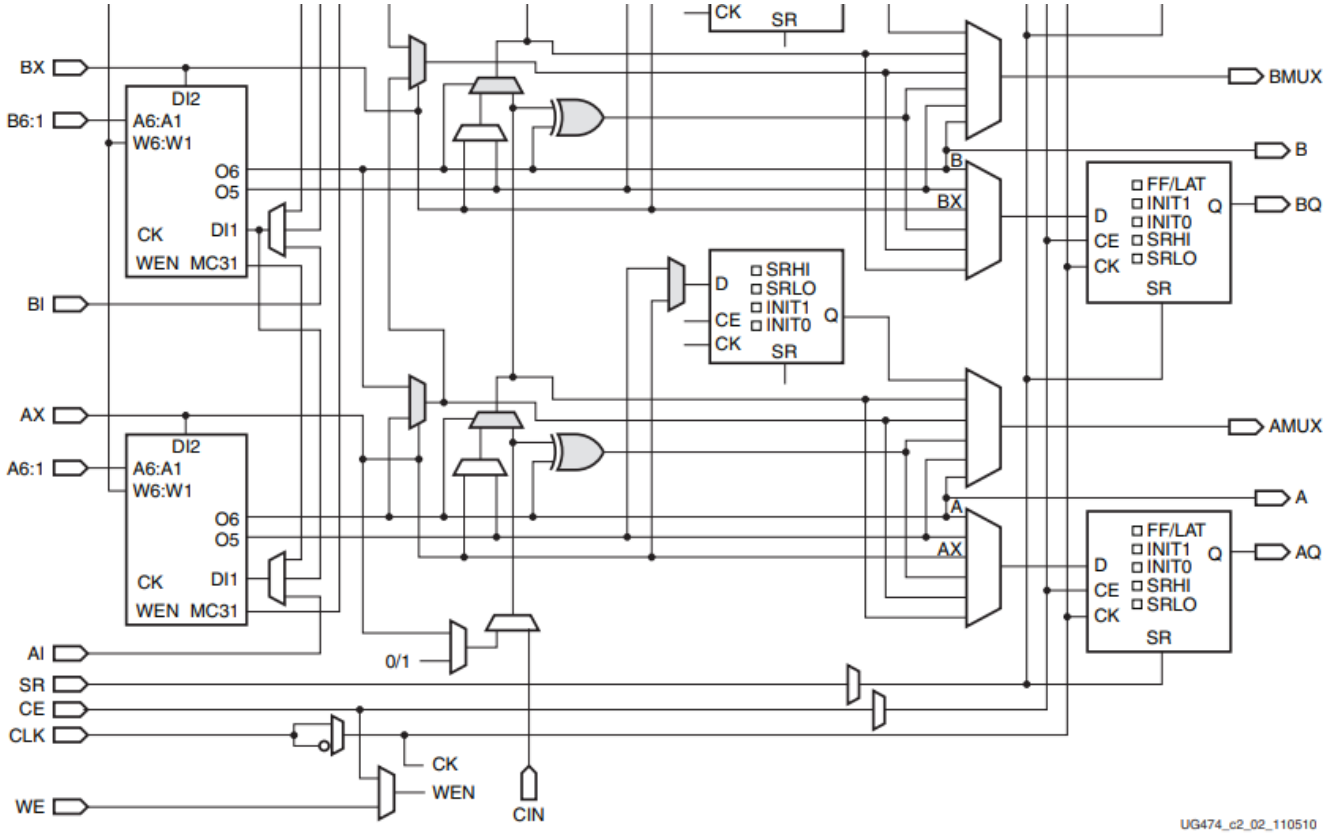
XILINX 7 SERIES FPGA ARCHITECTURE

Device	Slices[1]	SLICEL	SLICEM	6-input LUTs	Distributed RAM (Kb)	Shift Register (Kb)	Flip-Flops
7A15T	2,600[2]	1,800	800	10,400	200	100	20,800
7A35T	5,200[2]	3,600	1,600	20,800	400	200	41,600

Notes:

1. Each 7 series FPGA slice contains four LUTs and eight flip-flops; only SLICEMs can use their LUTs as distributed RAM or SRLs.





VERILOG HDL (Hardware Description Language)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

1984: Phil Moorby & Prabhu Goel @Gateway Design Automation

1990: Cadence Design System acquired Gateway

1991: Cadence made Verilog documentation open to public through OVI (Open Verilog International, now Accellera)

1995: Verilog was submitted to the IEEE and became standard 1364-1995

2001: Upgraded version of 1995 → 1364-2001

2005: Last version of Verilog → 1364-2005

2005: SystemVerilog → 1800-2005

SystemVerilog, Verilog dilinin bir üst kümesidir ve Verilog dilinin özelliklerini de kapsamaktadır.

VERILOG HDL (Hardware Description Language)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

Verilog ile “sentezlenebilir” sayısal devreler tasarlanabildiği gibi, “sentezlenemeyen” simülasyon amaçlı scriptler de yazılabilir.

Sentezlenebilir sayısal devreler, sentezlenemeyen testbench simülasyon scriptleri ile doğrulanırlar.

Biz bu ders kapsamında Verilog-2005 (IEEE 1364-2005) standardına uygun olarak kod geliştireceğiz.

Proje kapsamında Openlane açık-kaynak aracı ile tape-out planlanmaktadır. Openlane içerisinde bulunan Yosys sentezleyici aracı Verilog-2005 destekler.

IEEE

IEEE Standard for Verilog® Hardware Description Language

IEEE Computer Society

Sponsored by the
Design Automation Standards Committee

The Behavioral Task Force had the following membership:

Steven Sharp, Cadence Design Systems, Inc., *Chair*

Kurt Baty, WFSDB Consulting
Stefen Boyd, Boyd Technology
Shalom Bresticker, Intel Corporation
Dennis Brophy, Mentor Graphics Corporation
Cliff Cummings, Sunburst Design, Inc.
Steven Dovich, Cadence Design Systems, Inc.
Tom Fitzpatrick, Mentor Graphics Corporation
Ronald Goodstein, First Shot Logic Simulation and Design
Keith Gover, Mentor Graphics Corporation
Mark Hartoog, Synopsys, Inc.
Ennis Hawk, Jeda Technologies
Atsushi Kasuya, Jeda Technologies
Jay Lawrence, Cadence Design Systems, Inc.
Francoise Martinolle, Cadence Design Systems, Inc.
Kathryn McKinley, Cadence Design Systems, Inc.
Michael McNamara, Verisity, Ltd.
Don Mills, LCDM Engineering
Mehdi Mohtashemi, Synopsys, Inc.
Karen Pieper, Synopsys, Inc.
Brad Pierce, Synopsys, Inc.
Dave Rich, Mentor Graphics Corporation
Steven Sharp, Cadence Design Systems, Inc.
Alec Stanculescu, Fintronic, USA
Stuart Sutherland, Sutherland HDL, Inc.
Gordon Vreugdenhil, Mentor Graphics Corporation

At the time this standard was completed, the IEEE P1364 Working Group had the following membership:

Johny Srouji, IBM, *IEEE SystemVerilog Working Group Chair*
Tom Fitzpatrick, Mentor Graphics Corporation, *Chair*
Neil Korpusik, Sun Microsystems, Inc., *Co-chair*
Stuart Sutherland, Sutherland HDL, Inc., *Editor*
Shalom Bresticker, Intel Corporation, *Editor through February 2005*

VERILOG HDL (Hardware Description Language)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

- Gate-level modeling using instantiations of predefined and user-defined primitive gates.
- Dataflow modeling using continuous assignment statements with the keyword **assign**.
- Behavioral modeling using procedural assignment statements with the keyword **always**.

1364-2005 standard

XILINX VIVADO DESIGN SUITE

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

Default Part

Choose a default Xilinx part or board for your project.

https://digilent.com/reference/software/vivado/board-files

Parts | **Boards**

Reset All Filters

Update Board Repositories

Vendor: All Name: All Board Rev: Latest

Search:

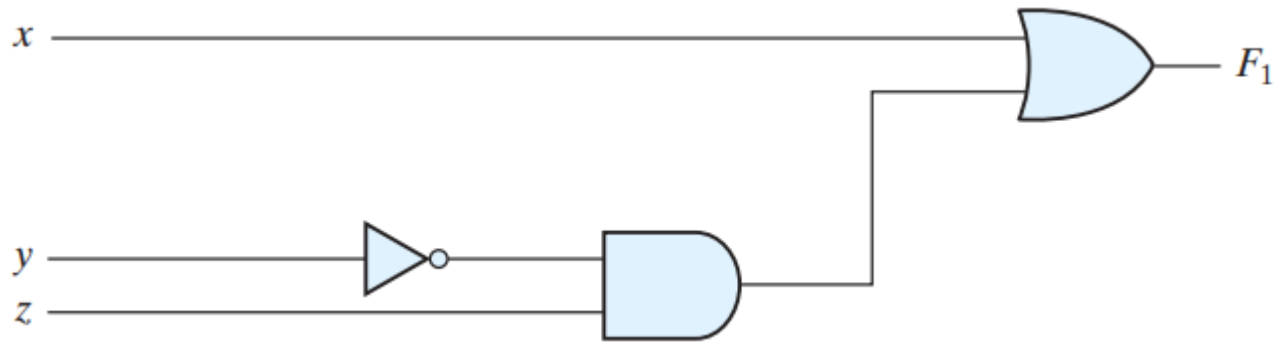
Display Name	Preview	Vendor	File Version	Part	I/
Basys3		digilentinc.com	1.2	xc7a35tcpg236-1	23
Nexys4 DDR		digilentinc.com	1.1	xc7a100tcsg324-1	32
ZedBoard Zynq Evaluation and Development Kit Add Daughter Card Connections		em.avnet.com	1.4	xc7z020clg484-1	48
Artix-7 AC701 Evaluation Platform Add Daughter Card Connections		xilinx.com	1.4	xc7a200tfbg676-2	67
Kintex UltraScale+ KCU116 Evaluation Platform		xilinx.com	1.4	xcku5p-ffvb676-2-e	67

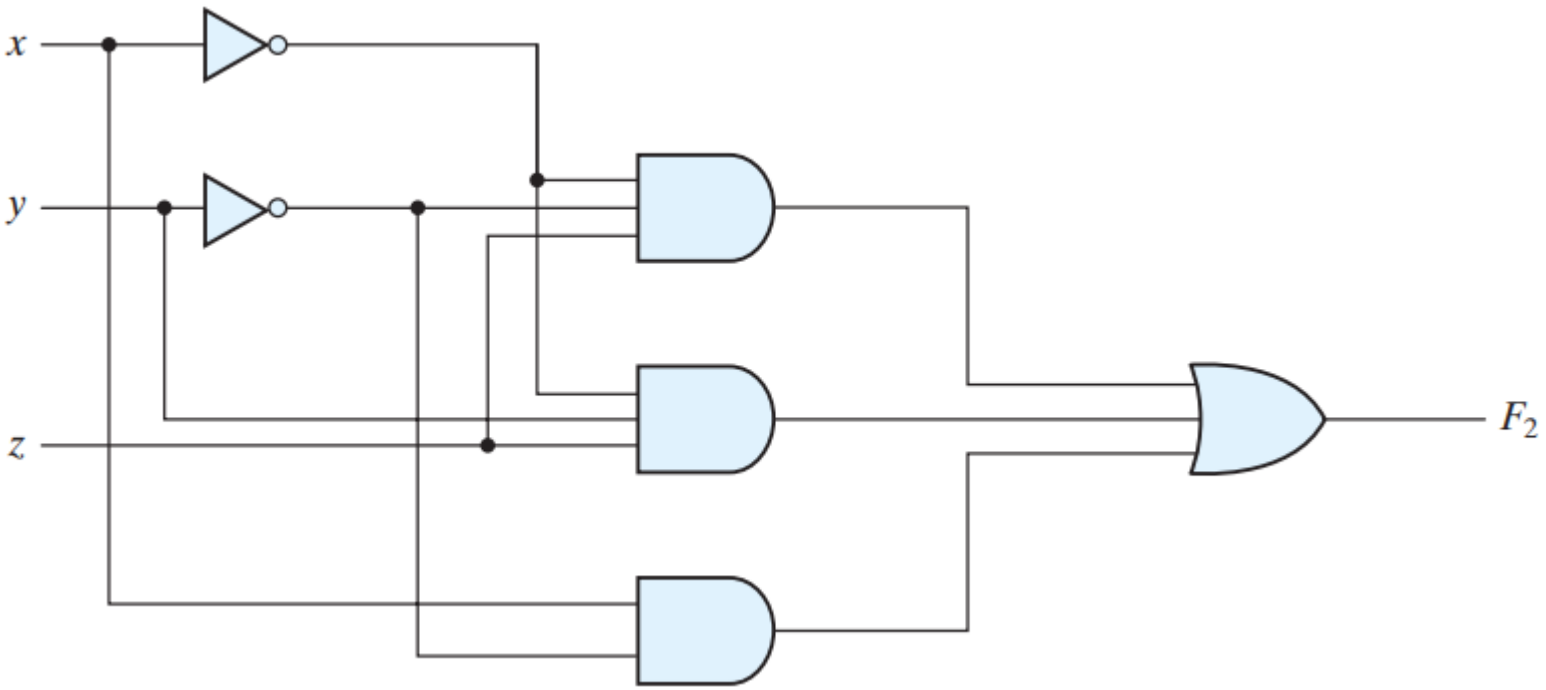
Bool Fonksiyonları

$F1 = x + y'z$

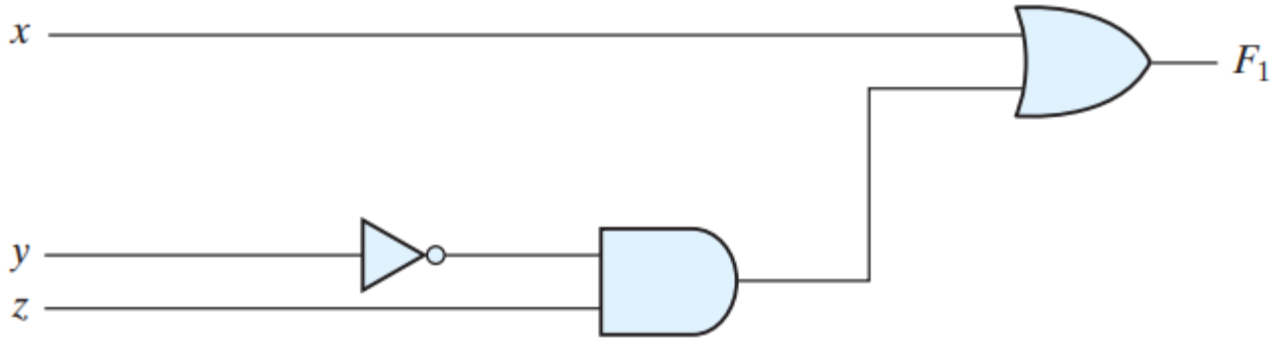
$F2 = x'y'z + x'yz + xy'$

F1 & F2				
Giriş			Çıkış	
x	y	z	F1	F2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0



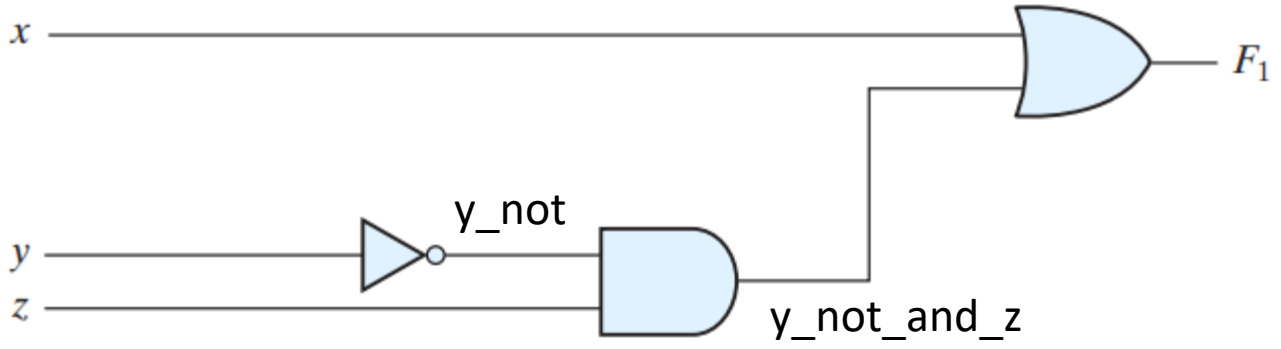


VERILOG HDL ÖRNEK-1



n_input gates	n_output gates	Three-state gates	Pull gates	MOS switches	Bidirectional switches
and	buf	bufif0	pulldown	cmos	rtran
nand	not	bufif1	pullup	nmos	rtranif0
nor		notif0		pmos	rtranif1
or		notif1		rcmos	tran
xnor				rnmos	tranif0
xor				rpmos	tranif1

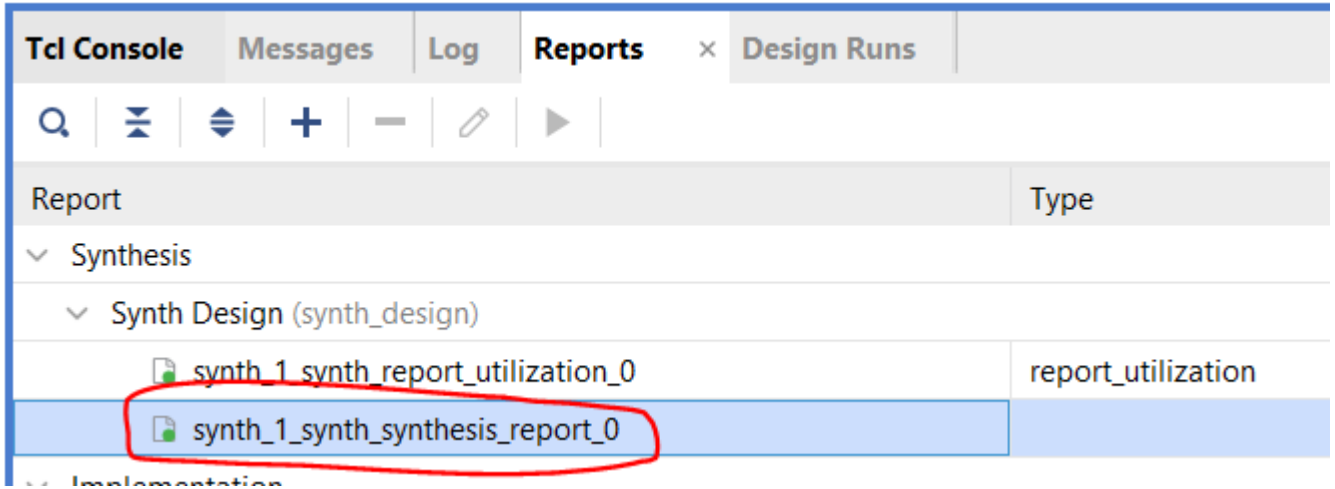
VERILOG HDL ÖRNEK-1 (Gate Level)



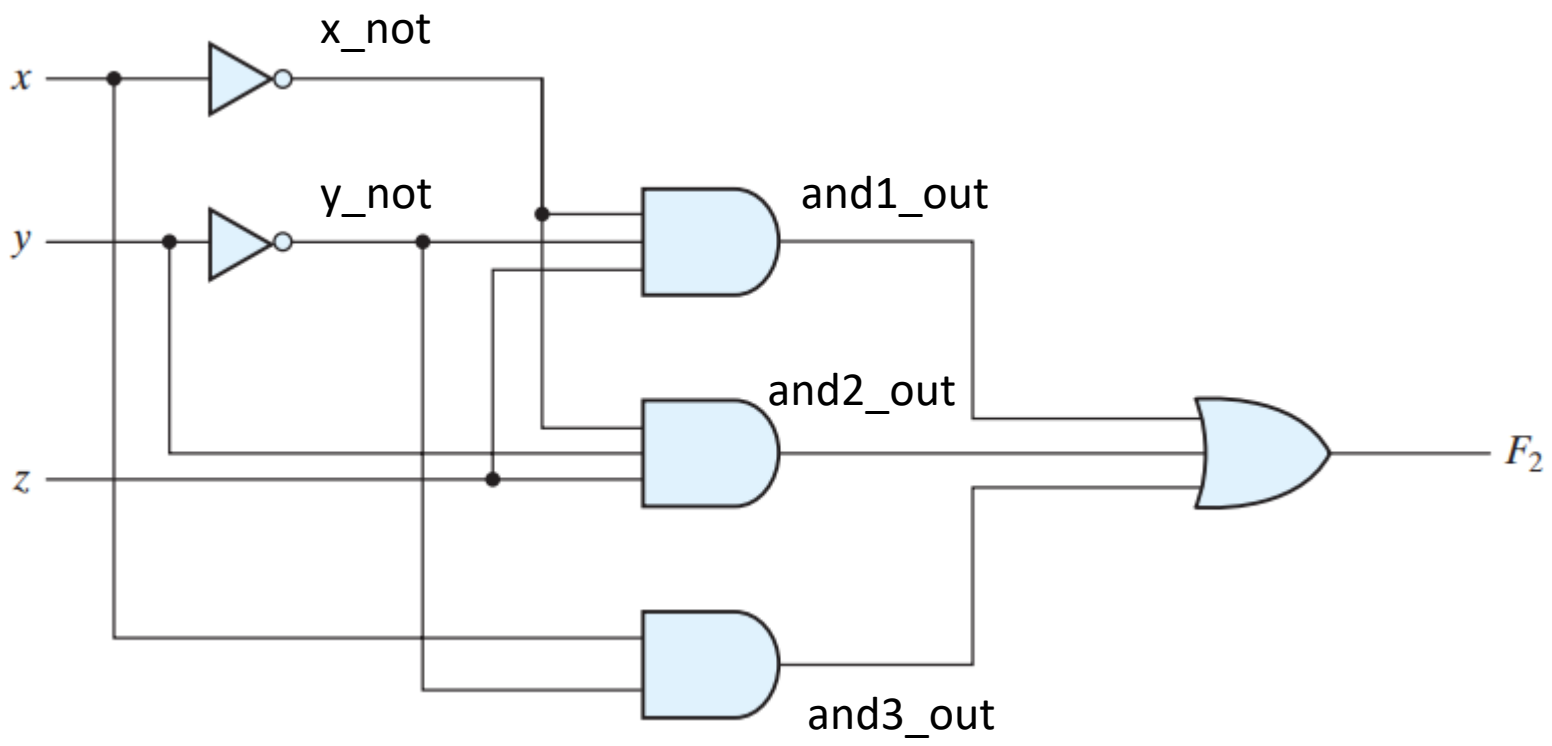

```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |LUT3 |     1|
|2     |IBUF |     3|
|3     |OBUF |     1|
+------+-----+------+
```

```
module ornek1
(
input x,
input y,
input z,
output F1
);

wire y_not;
wire y_not_and_z;

not G1 (y_not,y);
and G2 (y_not_and_z,y_not,z);
or G3 (F1,x,y_not_and_z);

endmodule
```

ismin önemi yok !!!



VERILOG HDL ÖRNEK-2 (Gate Level)




```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |LUT3 |     1|
|2     |IBUF |     3|
|3     |OBUF |     1|
+------+-----+------+
```

```
module ornek2
(
input x,
input y,
input z,
output F2
);

wire x_not;
wire y_not;
wire and1_out;
wire and2_out;
wire and3_out;

not G1 (x_not,x);
not G2 (y_not,y);
and G3 (and1_out,x_not,y_not,z);
and G4 (and2_out,x_not,y,z);
and G5 (and3_out,x,y_not);
or G6 (F2,and1_out,and2_out,and3_out);

endmodule
```

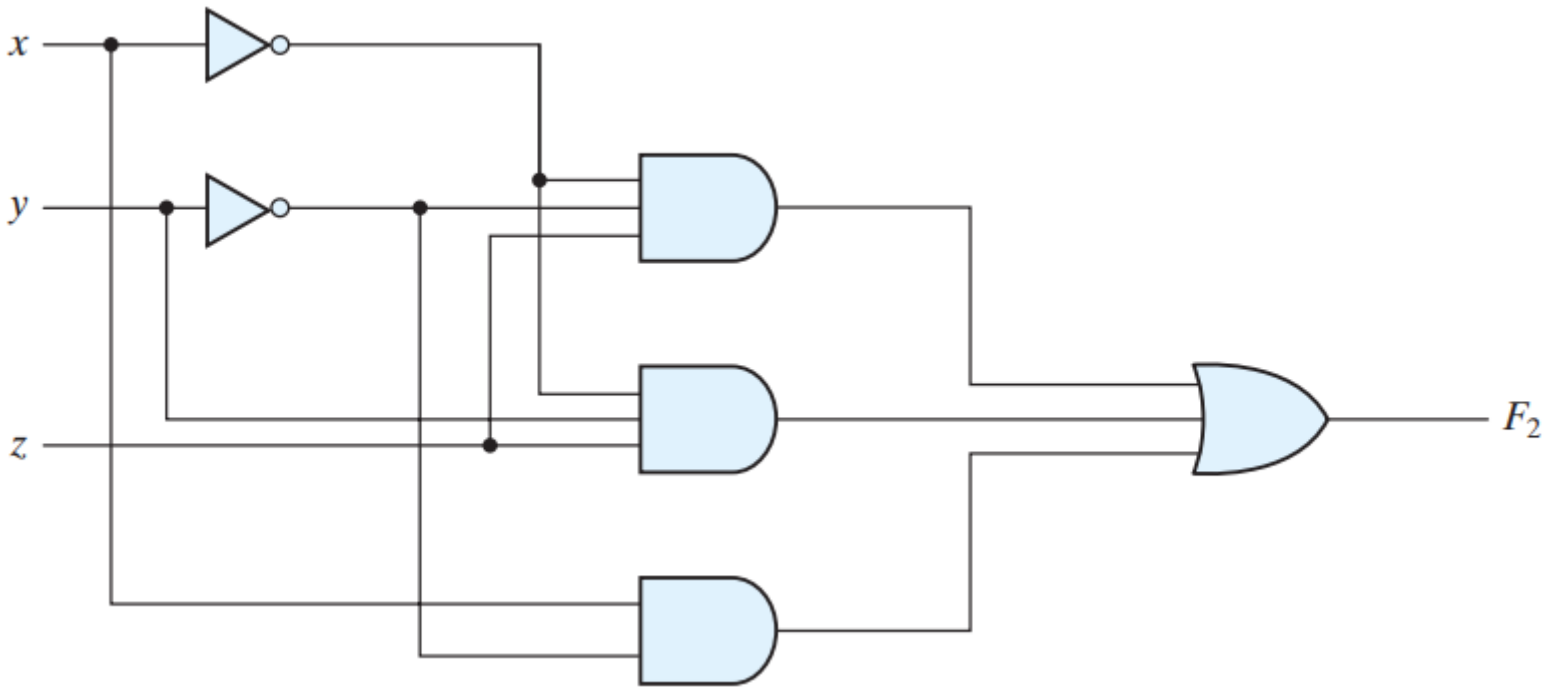
Bool Fonksiyonları - Sadeleştirme

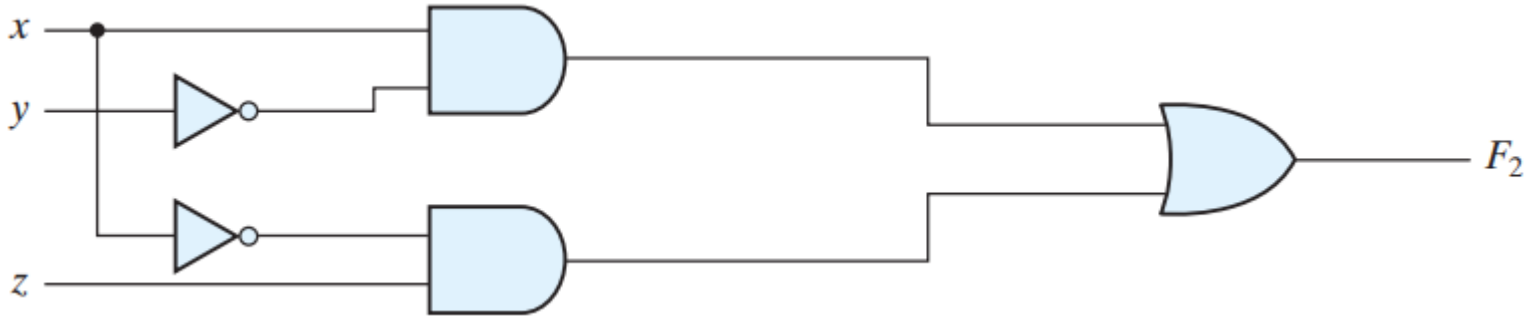
F2 = x’y’z + x’yz + xy’

F2 = x’z(y’+y) + xy’

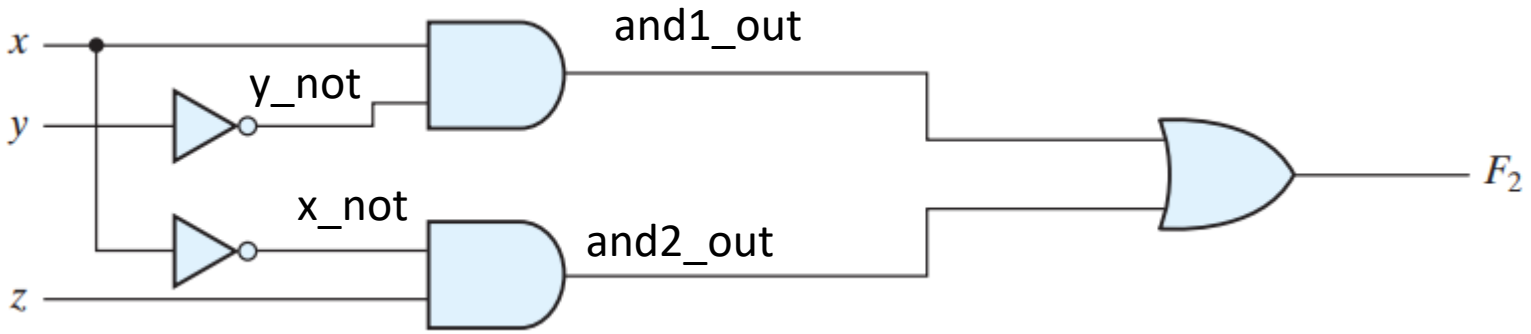
F2 = x’z + xy’

Fonksiyonda sadeleştirme daha az kapı kullanımına, yani daha hızlı, daha az alana sahip, daha ucuz ve daha az güç tüketen devre demektir





VERILOG HDL ÖRNEK-3 (Gate Level)

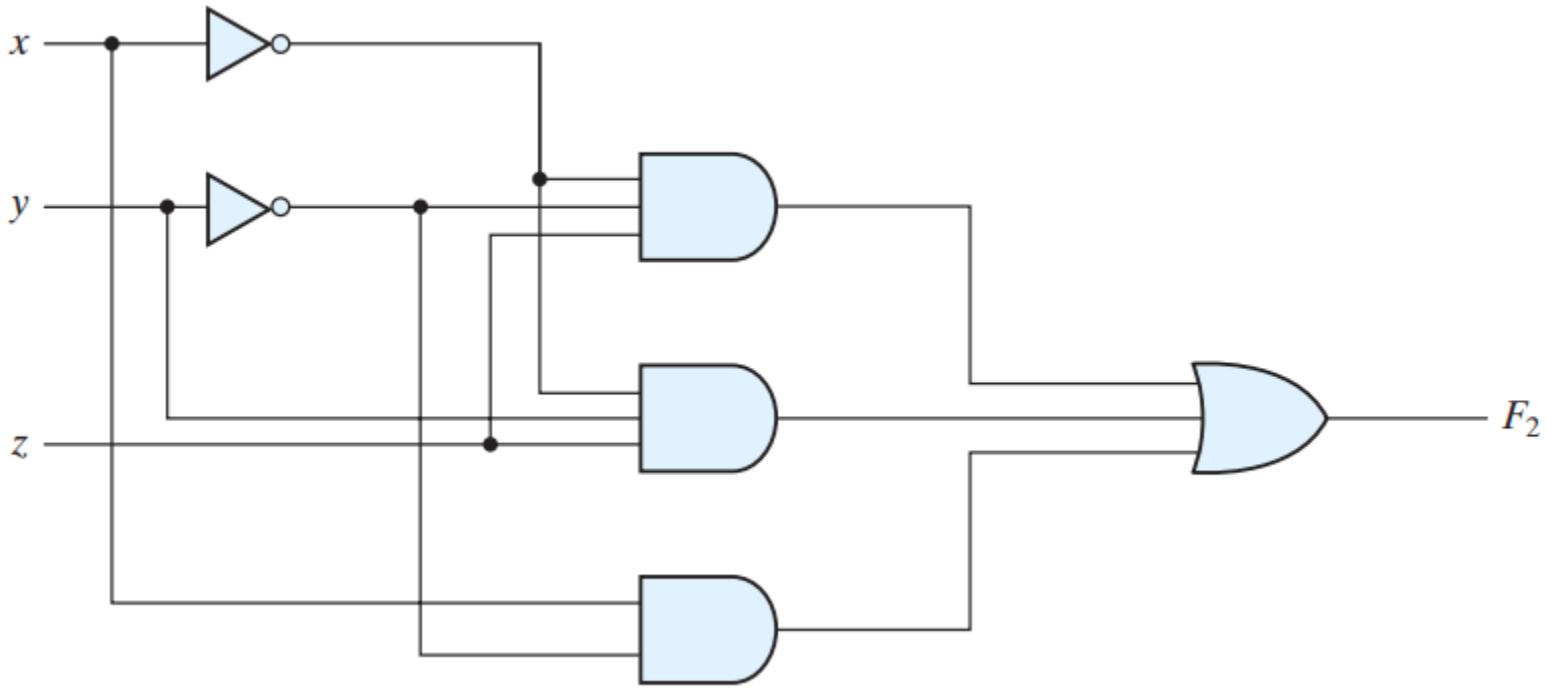


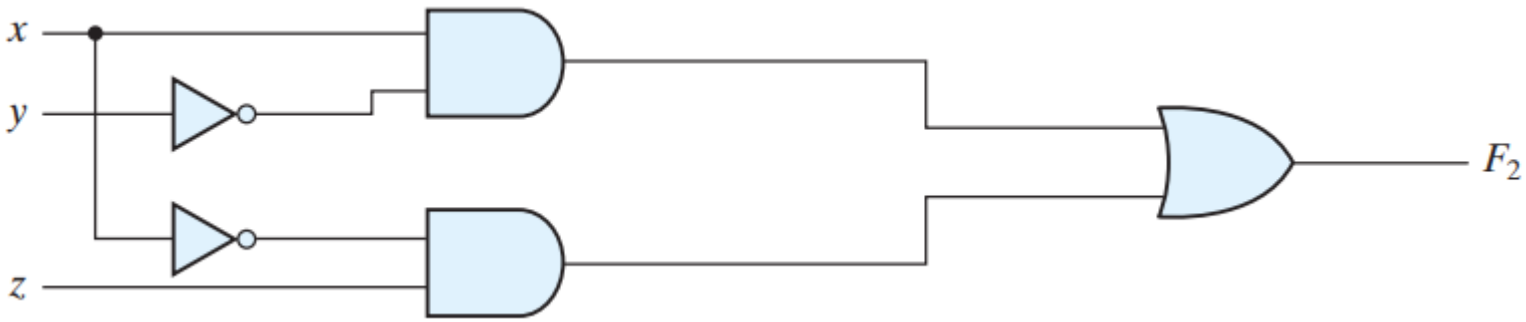
“Fonksiyonda sadeleştirme daha az kapı kullanımına, yani daha hızlı, daha az alana sahip, daha ucuz ve daha az güç tüketen devre demektir”

```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |LUT3 |     1|
|2     |IBUF |     3|
|3     |OBUF |     1|
+------+-----+------+
```

```verilog
module ornek3
(
input x,
input y,
input z,
output F2
);

wire x_not;
wire y_not;
wire and1_out;
wire and2_out;

not G1 (x_not,x);
not G2 (y_not,y);
and G3 (and1_out,x,y_not);
and G4 (and2_out,x_not,z);
or G5 (F2,and1_out,and2_out);

endmodule
```

SENTEZLEYİCİ ARAÇLARIN OPTİMİZASYONU





Vivado sentezleyici, F2 fonksiyonunu 3 girişli bir LUT ile gerçekleştirmektedir.

FPGA'larda, kapılar değil, LUT kullanımı önemlidir.

ASIC'te ise standard kütüphanede kapılar önemlidir. Ama yine ASIC için sentezleyiciler de fonksiyonel sadeleştirme ile optimizasyon yaparlar.

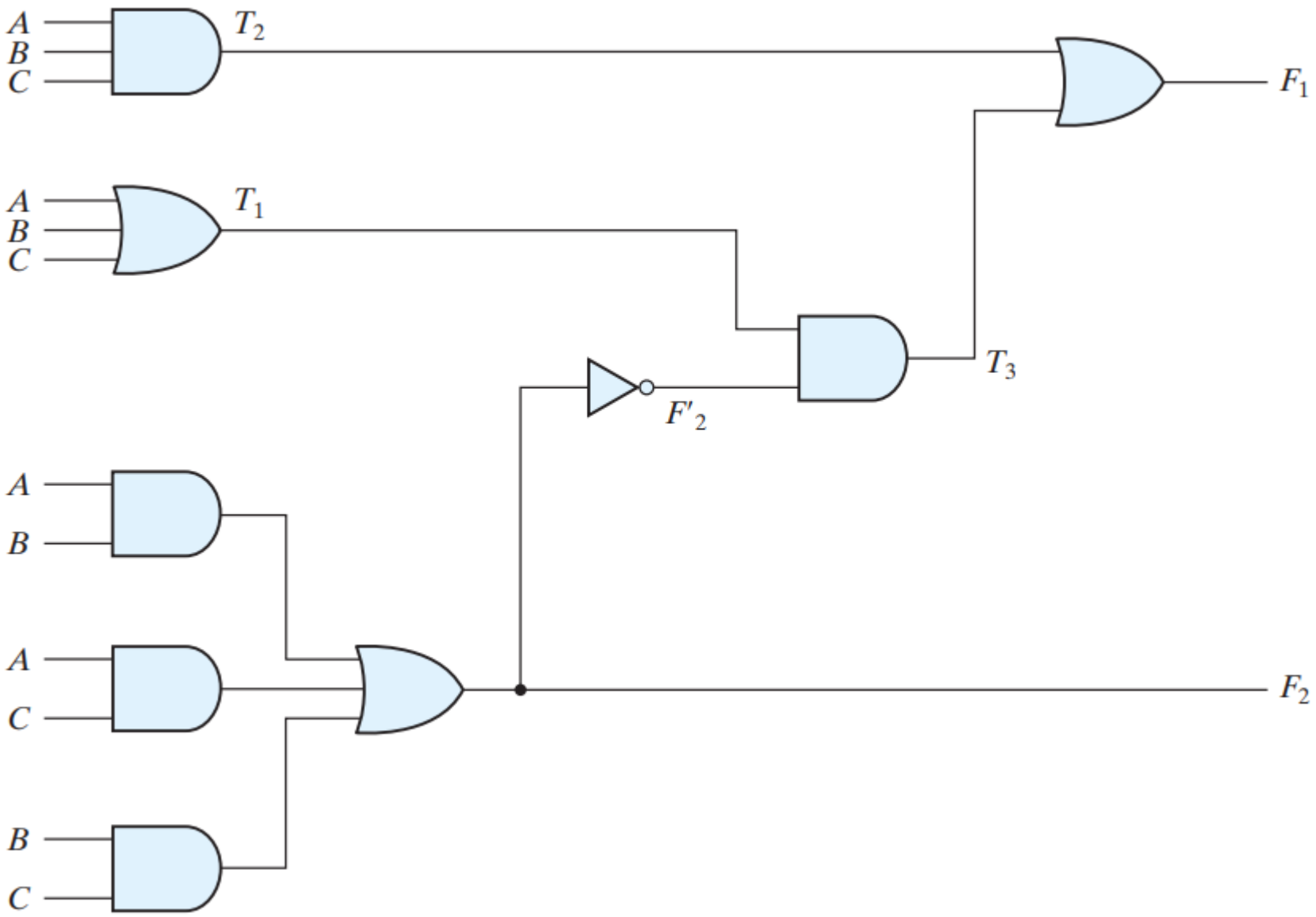
COMBINATIONAL (BİRLEŞİK) DEVRELER

Combinational (birleşik): Devrenin çıkış sinyal(ler)i, yalnızca o anki giriş sinyal(ler)inin değerine bağlıdır. Devrede durum (state) belirten bir bellek (memory) bulunmamaktadır. Giriş sinyali değişir değişmez çıkış sinyaline etki eder.

Sequential (sıralı): Devrenin çıkış sinyal(ler)i, giriş sinyal(ler)inin ve devrede durum (state) belirten bellek (memory) değerine bağlıdır. Giriş sinyalinin değişimi her zaman çıkış sinyaline etki etmez, devrenin durumuna bağlıdır.

NOT: Derste şu ana kadar gördüğümüz bütün fonksiyonlar ve devreler combinational devreydi.

COMBINATIONAL DEVRE ANALİZİ



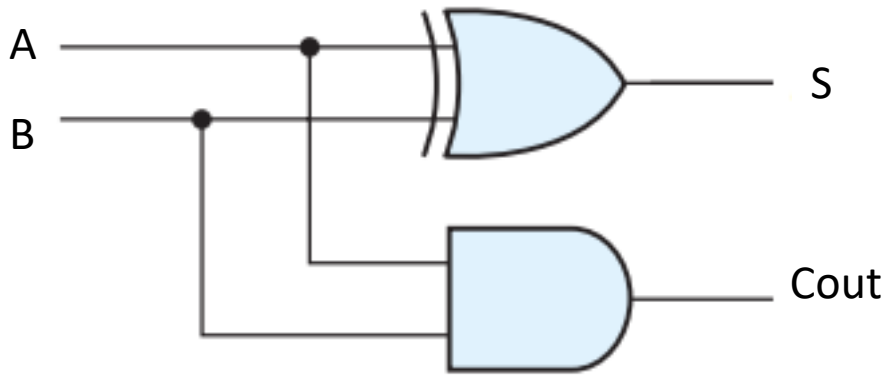
F1 & F2				
Giriş			Çıkış	
A	B	C	F1	F2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

COMBINATIONAL DEVRE TASARIMI – Half Adder

Giriş		Çıkış	
A	B	S	Cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S = A'B+AB' → S = A xor B

Cout = AB

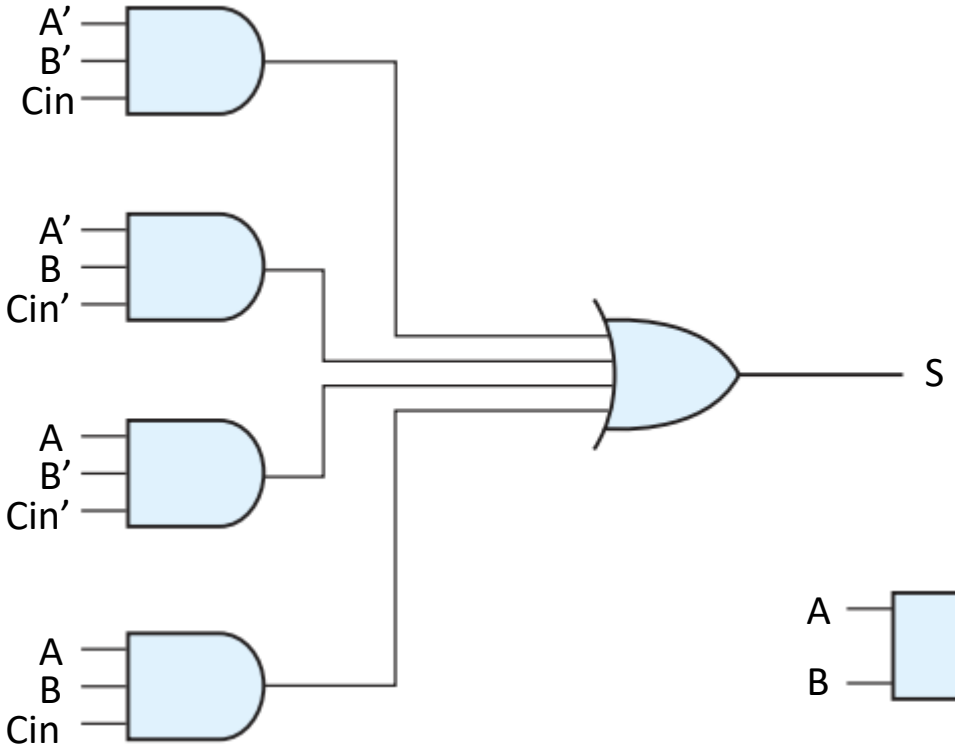


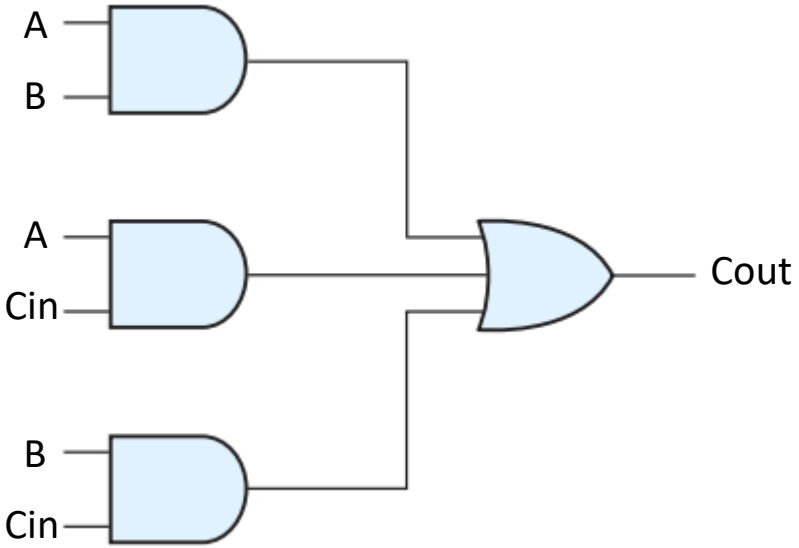

```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |LUT2 |     2|
|2     |IBUF |     2|
|3     |OBUF |     2|
+------+-----+------+
```

```
module half_adder
(
input a_i,
input b_i,
output s_o,
output cout_o
);

xor G1 (s_o,a_i,b_i);
and G2 (cout_o,a_i,b_i);


endmodule
```

COMBINATIONAL DEVRE TASARIMI – Full Adder

Giriş			Çıkış	
A	B	Cin	S	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





S = A'B'Cin+A'BCin'+AB'Cin'+ABCin → Cin'(A'B+AB')+Cin(A'B'+AB)
→ Cin'(A^B)+Cin(A^B)' → A^B^C

Cout = AB+ACin+BCin

COMBINATIONAL DEVRE TASARIMI – Full Adder

Giriş			Çıkış	
A	B	Cin	S	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

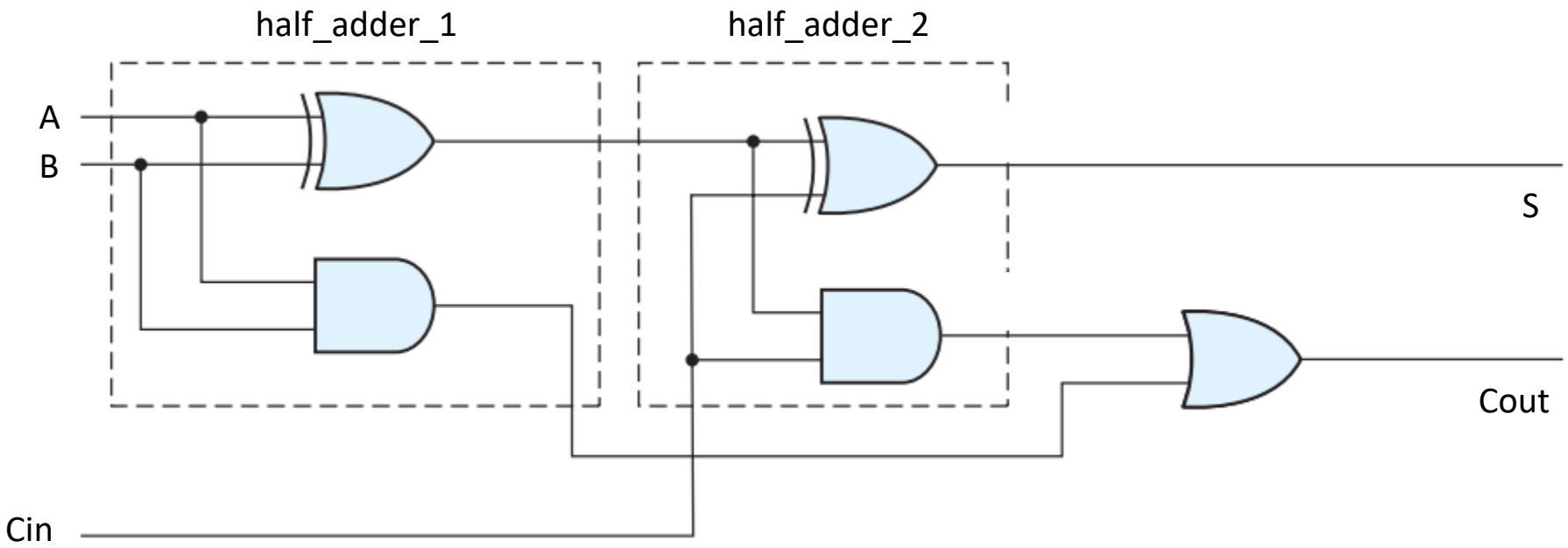
```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |LUT3 |     2|
|2     |IBUF |     3|
|3     |OBUF |     2|
+------+-----+------+
```

```
module full_adder
(
input a_i,
input b_i,
input cin_i,
output s_o,
output cout_o
);

wire and1, and2, and3;

xor G1 (s_o,a_i,b_i,cin_i);
and G2 (and1,a_i,b_i);
and G3 (and2,a_i,cin_i);
and G4 (and3,b_i,cin_i);
or G5 (cout_o,and1,and2,and3);

endmodule
```

S = A'B'Cin+A'BCin'+AB'Cin'+ABCin → Cin'(A'B+AB')+Cin(A'B'+AB)
→ Cin'(A^B)+Cin(A^B)' → A^B^C

Cout = AB+ACin+BCin

Full Adder – Hiyerarşik Tasarım



- **full_adder_hier** (full_adder_hier.v) (2)
  - HA1 : half_adder (half_adder.v)
  - HA2 : half_adder (half_adder.v)

```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |LUT3 |     2|
|2     |IBUF |     3|
|3     |OBUF |     2|
+------+-----+------+
```

```
module full_adder_hier
(
input a_i,
input b_i,
input cin_i,
output s_o,
output cout_o
);

wire ha1_s, ha1_cout, ha2_cout;

half_adder HA1
(
.a_i    (a_i),
.b_i    (b_i),
.s_o    (ha1_s),
.cout_o (ha1_cout)
);

half_adder HA2
(
.a_i    (ha1_s),
.b_i    (cin_i),
.s_o    (s_o),
.cout_o (ha2_cout)
);

or G1 (cout_o,ha2_cout,ha1_cout);

endmodule
```

Verilog – Scalar vs Vector

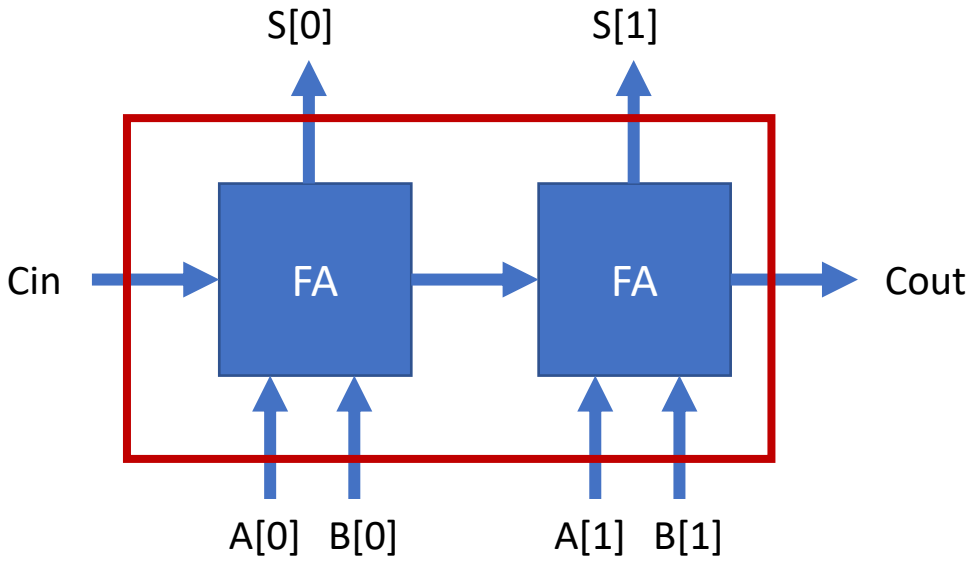
```
module full_adder_hier
(
input a_i,
input b_i,
input cin_i,
output s_o,
output cout_o
);

wire ha1_s, ha1_cout, ha2_cout;

half_adder HA1
(
.a_i    (a_i),
.b_i    (b_i),
.s_o    (ha1_s),
.cout_o (ha1_cout)
);

half_adder HA2
(
.a_i    (ha1_s),
.b_i    (cin_i),
.s_o    (s_o),
.cout_o (ha2_cout)
);

or G1 (cout_o,ha2_cout,ha1_cout);

endmodule
```

scalar

```
module binary_adder_2bit
(
input [1:0] a_i,
input [1:0] b_i,
input cin_i,
output [1:0] s_o,
output cout_o
);
```

vector

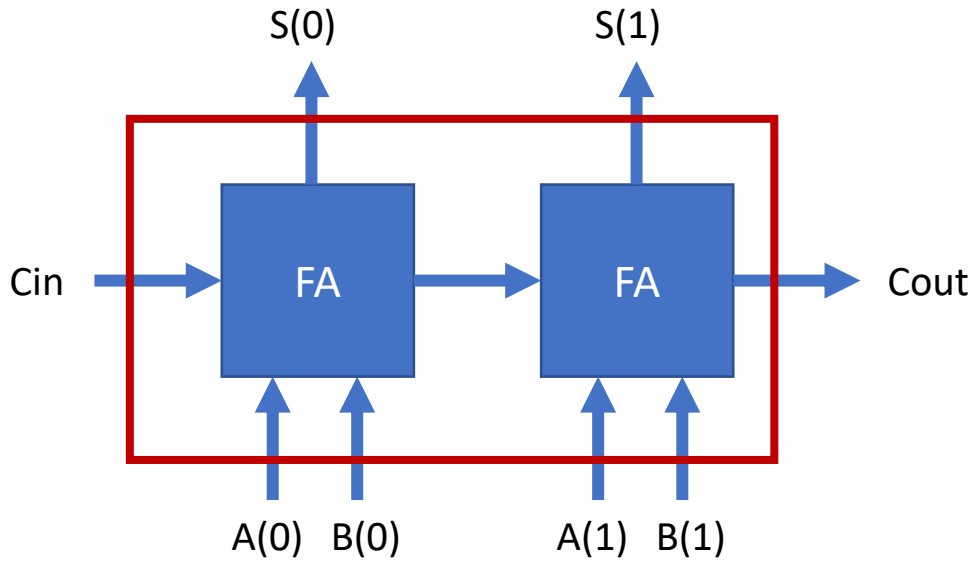
2-bit Binary Adder



2-bit Binary Adder

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

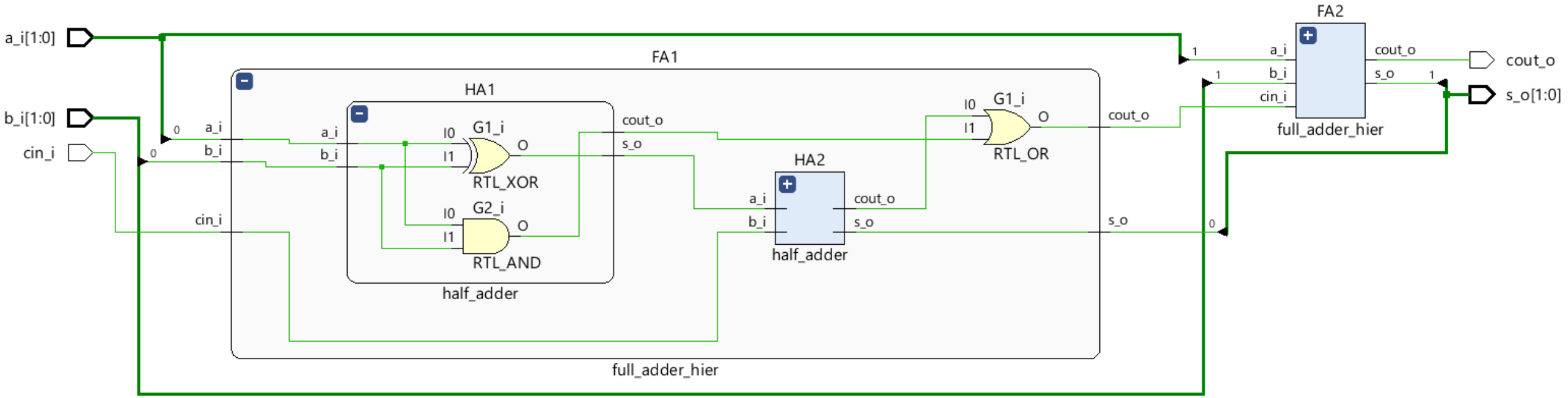
2-bit Binary Adder



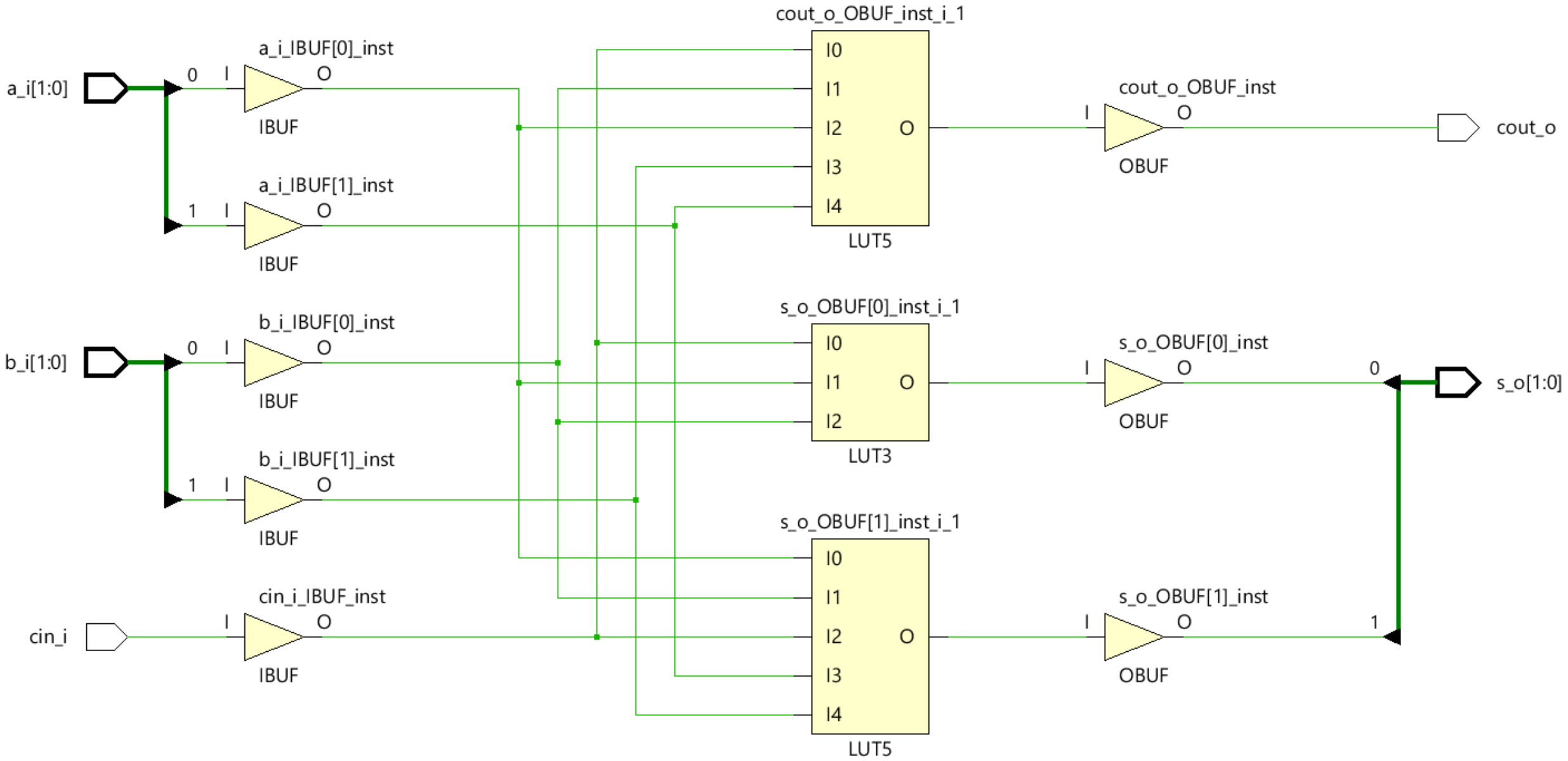

```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |LUT3 |     1|
|2     |LUT5 |     2|
|3     |IBUF |     5|
|4     |OBUF |     3|
+------+-----+------+
```

```
module binary_adder_2bit
(
input [1:0] a_i,
input [1:0] b_i,
input cin_i,
output [1:0] s_o,
output cout_o
);

wire cout_fa1;

full_adder_hier FA1
(
.a_i    (a_i[0]),
.b_i    (b_i[0]),
.cin_i  (cin_i),
.s_o    (s_o[0]),
.cout_o (cout_fa)
);

full_adder_hier FA2
(
.a_i    (a_i[1]),
.b_i    (b_i[1]),
.cin_i  (cout_fa),
.s_o    (s_o[1]),
.cout_o (cout_o)
);

endmodule
```

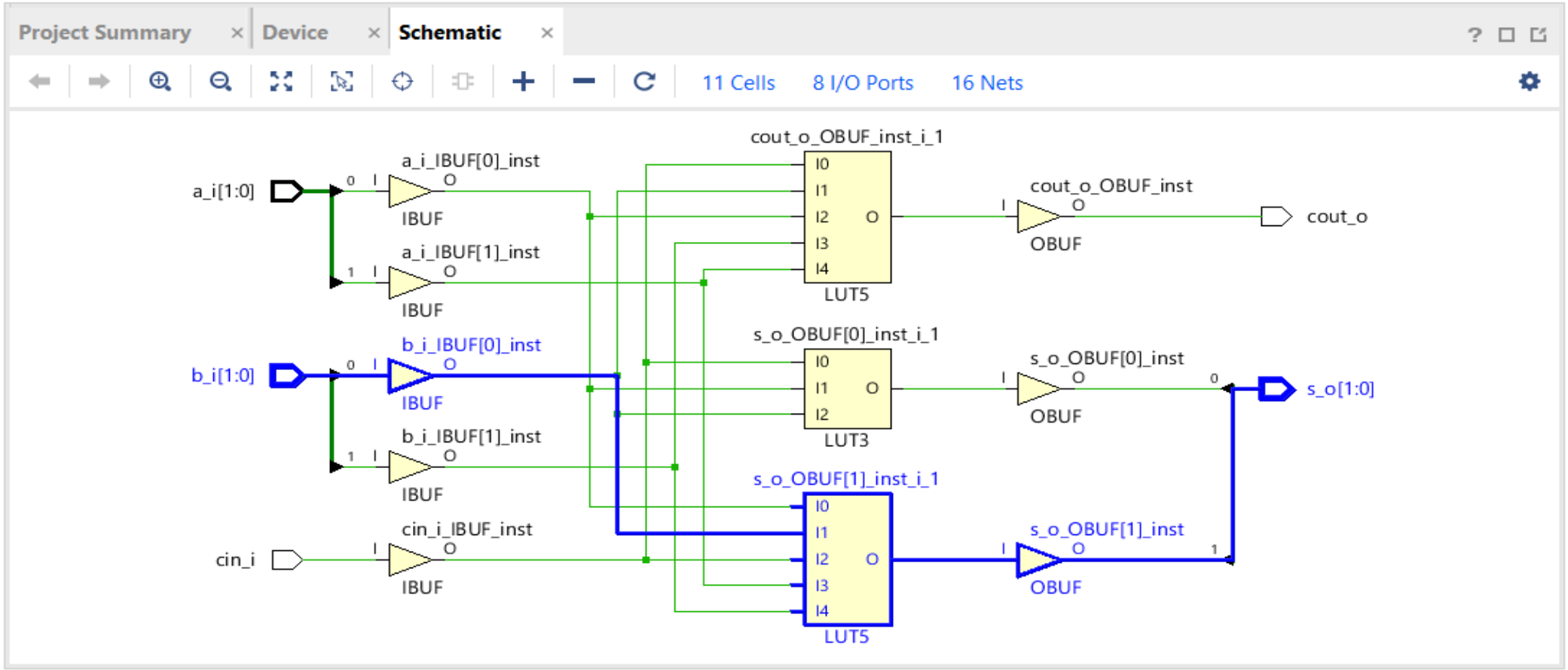
2-bit Binary Adder – Vivado Elaborated Design



2-bit Binary Adder – Vivado Synthesized Design



2-bit Binary Adder – Report Timing Summary

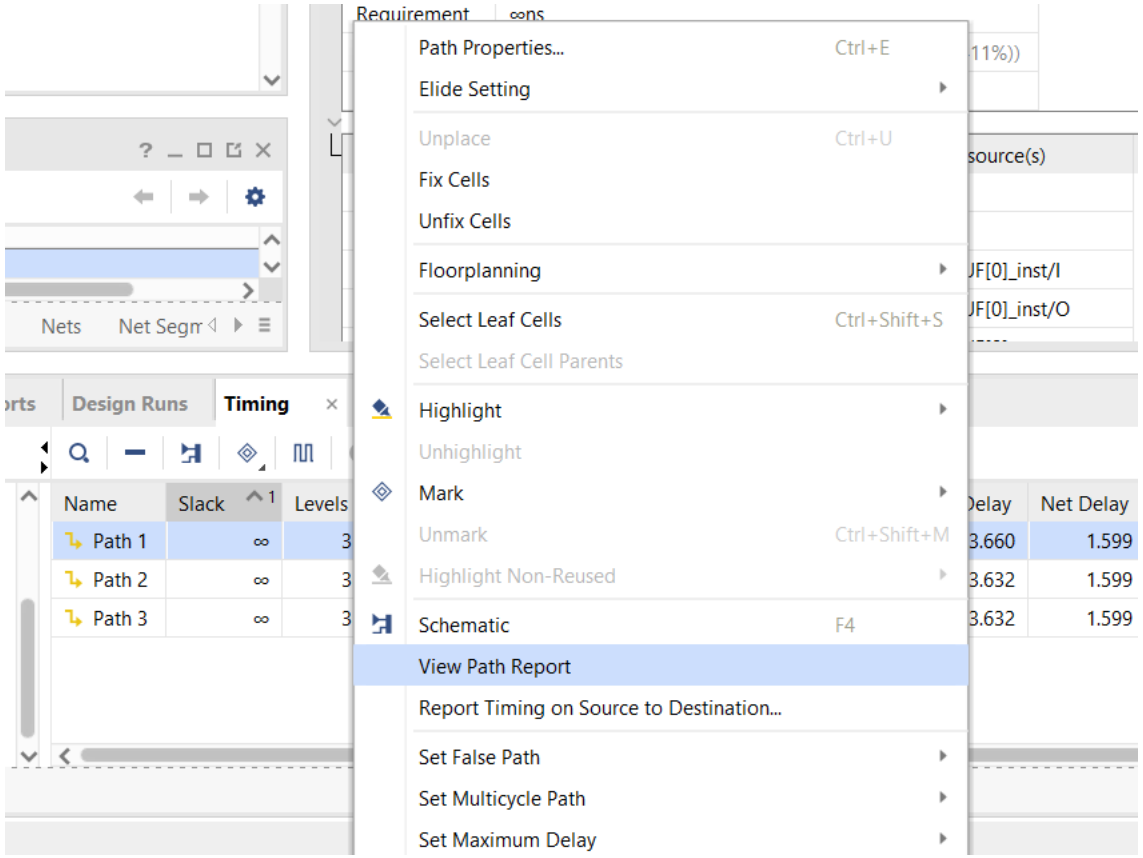


Design Runs | Timing

Unconstrained Paths - NONE - NONE - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	∞	3	4	3	b_i[0]	s_o[1]	5.259	3.660	1.599	∞	input port clock		
Path 2	∞	3	4	2	a_i[1]	cout_o	5.231	3.632	1.599	∞	input port clock		
Path 3	∞	3	4	3	b_i[0]	s_o[0]	5.231	3.632	1.599	∞	input port clock		

Path Delays



Project Summary | Device | Schematic | **Path 1 - timing_1**

Summary

Name	Path 1
Slack	∞ns
Source	b_i[0] (input port)
Destination	s_o[1] (output port)
Path Group	(none)
Path Type	Max at Slow Process Corner
Requirement	∞ns
Data P...Delay	5.259ns (logic 3.660ns (69.589%) route 1.599ns (30.411%))
Logic Levels	3 (IBUF=1 LUT5=1 OBUF=1)

Data Path

Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
	(r) 0.000	0.000		b_i[0]
net (fo=0)	0.000	0.000		b_i[0]
				b_i_IBUF[0]_inst/I
IBUF (Prop_ibuf_I_O)	(r) 0.923	0.923		b_i_IBUF[0]_inst/O
net (fo=3, unplaced)	0.800	1.722		b_i_IBUF[0]
				s_o_OBUF[1]_inst_i_1/I1
LUT5 (Pro...t5_I1_O)	(r) 0.152	1.874		s_o_OBUF[1]_inst_i_1/O
net (fo=1, unplaced)	0.800	2.674		s_o_OBUF[1]
				s_o_OBUF[1]_inst/I
OBUF (Pr...buf_I_O)	(r) 2.585	5.259		s_o_OBUF[1]_inst/O
net (fo=0)	0.000	5.259		s_o[1]
				s_o[1]

Verilog Testbench

Why `reg` in testbench for inputs of DUT? Generating input stimulus is one of the primary tasks of a testbench. Unless you want to tie the input to a static value, you will have to

1. Change the value of inputs at different time-stamps
2. Hold the value of inputs stable between time-stamps

```
initial begin
   #10 reset = 0;
   #20 reset = 1; // apply reset for few cycles
   #40 reset = 0;
end
```

In the code shown above, reset changes value at 3 different time stamps and you want the value of reset to held stable between those three intervals. Right?

Now, recall what is the data type available in Verilog for holding values — `reg`

Why `wire` in testbench for outputs of DUT? Another major task of a testbench is to monitor the ouputs of DUT. Since testbench is going to sample the value of outputs, therefore, only a connection is required. There is no need to hold value of outputs. Again, recall what is the data type for establishing a continuous assignment — `wire`

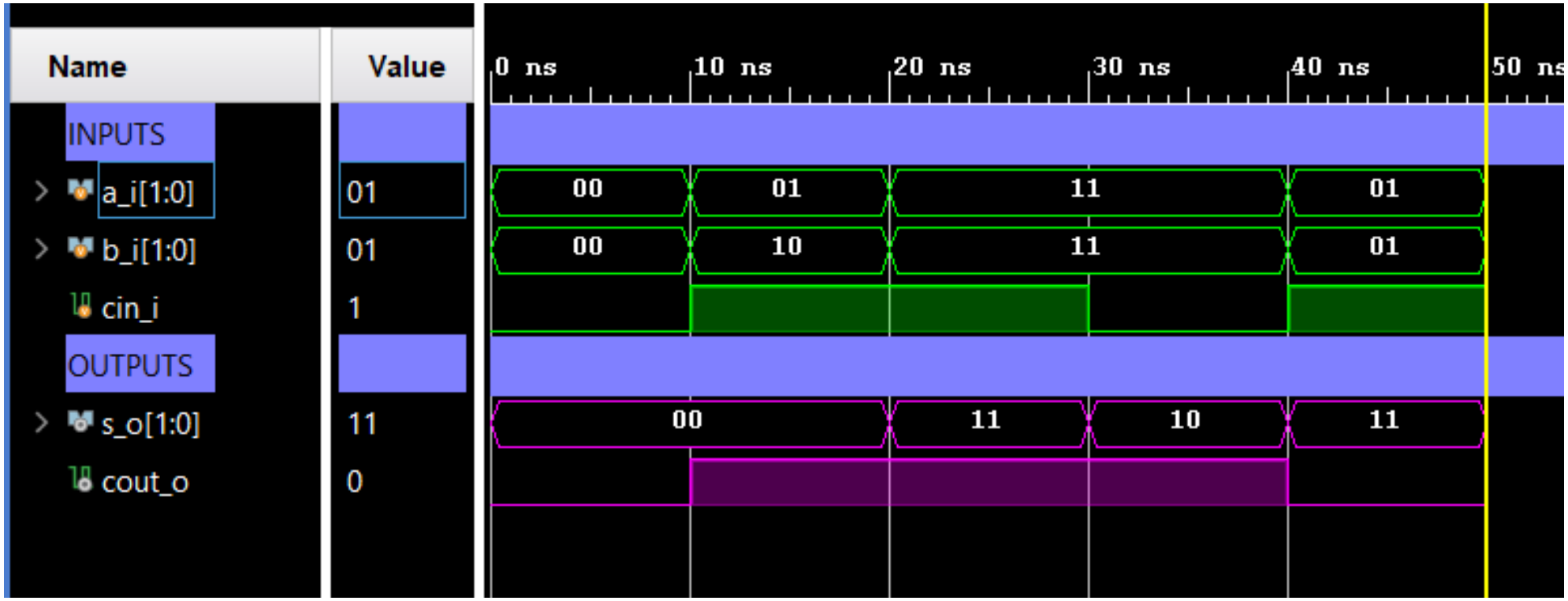
Why `reg` in DUT for outputs? Because DUT needs to drive these outputs and hold them stable between value change.

Why `wire` in DUT for inputs? Because DUT needs to just sample the inputs and not hold their values.

Simple!

```
module tb_binary_adder_2bit;

reg [1:0] a_i,b_i;  // define inputs of DUT as reg
reg cin_i;
wire [1:0] s_o;     // define outputs of DUT as wire
wire cout_o;

binary_adder_2bit DUT   // Design Under Test
(
.a_i    (a_i   ),
.b_i    (b_i   ),
.cin_i  (cin_i ),
.s_o    (s_o   ),
.cout_o (cout_o)
);
```

Verilog Testbench




```
initial begin
    a_i     = 2'b00;
    b_i     = 2'b00;
    cin_i   = 1'b0;
    #10;
    a_i     = 2'b01;
    b_i     = 2'b10;
    cin_i   = 1'b1;
    #10;
    a_i     = 2'b11;
    b_i     = 2'b11;
    cin_i   = 1'b1;
    #10;
    a_i     = 2'b11;
    b_i     = 2'b11;
    cin_i   = 1'b0;
    #10;
    a_i     = 2'b01;
    b_i     = 2'b01;
    cin_i   = 1'b1;
    #10;
    $finish;
end

endmodule
```

Kodlar ve Ders Notları Github Hesabımda