Tahmini Ders İçeriği
(Tentative Couse Schedule – Syllabus)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

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: **(3-7 Ekim)** FPGA, Birleşik (Combinational) Devreler, Aritmetik Modüller, Decoder, Encoder, Mux, Verilog HDL

5-6. Hafta: **(10-14, 17-21 Ekim)** Ardışık (Sequential) Devreler, Mandal (Latch), Flip-Flop, Zamanlama (Timing)

Lab Sınavı (265/264L) **(19 Ekim)**

Proje Duyurusu

7. Hafta: **(24-28 Ekim)** Durum Makinaları, Örnek Tasarımlar

8. Hafta: **(31 Ekim – 4 Kasım)** Yazmaçlar (Registers), Sayaçlar (Counters)

Ara Sınav (265/264) **(9 Kasım)**

9. Hafta: **(7-11 Kasım)** Bellekler, FPGA'da Block RAM, OpenRAM

10. Hafta: **(14-18 Kasım)** RTL (Register Transfer Level) ASMD (Algorithmic State Machine and Datapath) Tasarımları

11-12. Hafta: **(21-25 Kasım, 28 Kasım – 2 Aralık)** Boru hattı, FPGA ve ASIC Tasarım Akışları

Final **(7 Aralık)** – Proje Teslimleri **(18 Aralık)**

Ders Puanlaması

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

- **BİL-264**

10 proje (fpga flow) + 35 Ara Sınav (264/265) + 55 Final (Verilog) (264/264L/265)

Extralar (sınav ort > 50): 15 proje (caravel asic flow) + 2 Ödev (10)

- **BİL-265**

10 proje (fpga flow) + 15 1. Ara Sınav (264L/265) + 20 2. Ara Sınav (264/265) + 55 Final (Verilog) (264/264L/265)

Extralar (sınav ort > 50): 15 proje (caravel asic flow) + 4 Ödev (10)

- **BİL-264L**

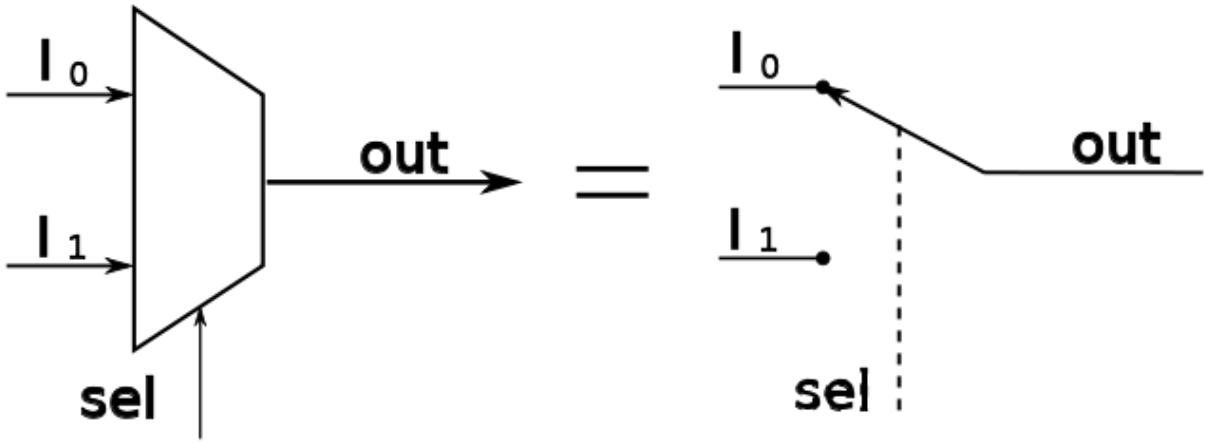
45 Ara Sınav (264L/265) + 55 Final (264/264L/265)

Extralar (sınav ort > 50): 2 Ödev (20)

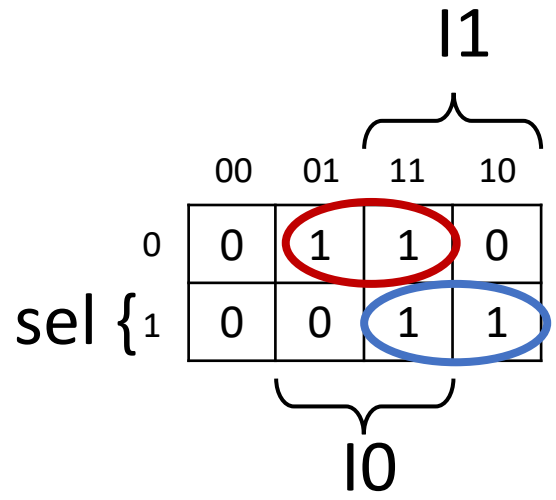
Multiplexer (Çoklayıcı)

"a device that enables the simultaneous transmission of several messages or signals over one communications channel"

"a piece of electronic equipment that can send more than one electrical signal using only one connection"



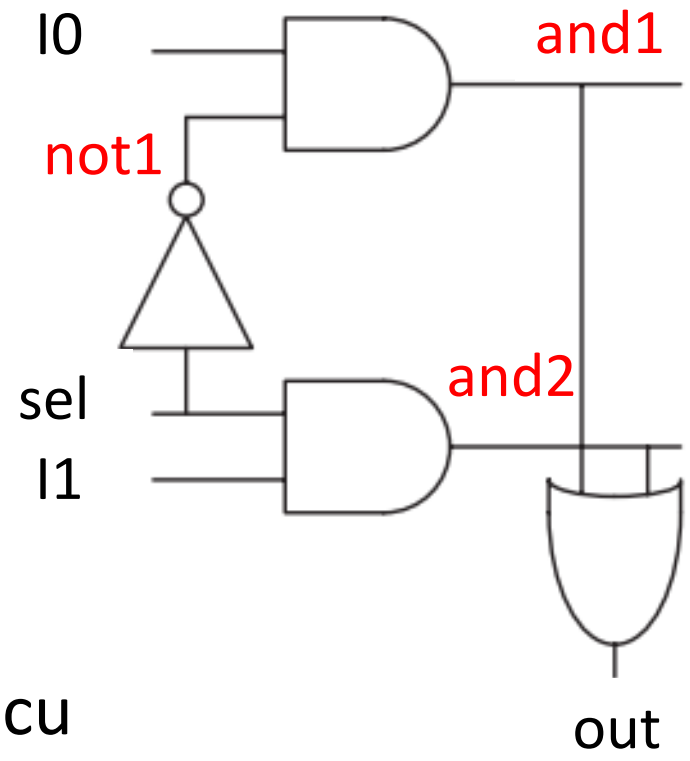
Giriş			Çıkış
sel	I1	I0	out
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



out = sel'I0 + I1sel

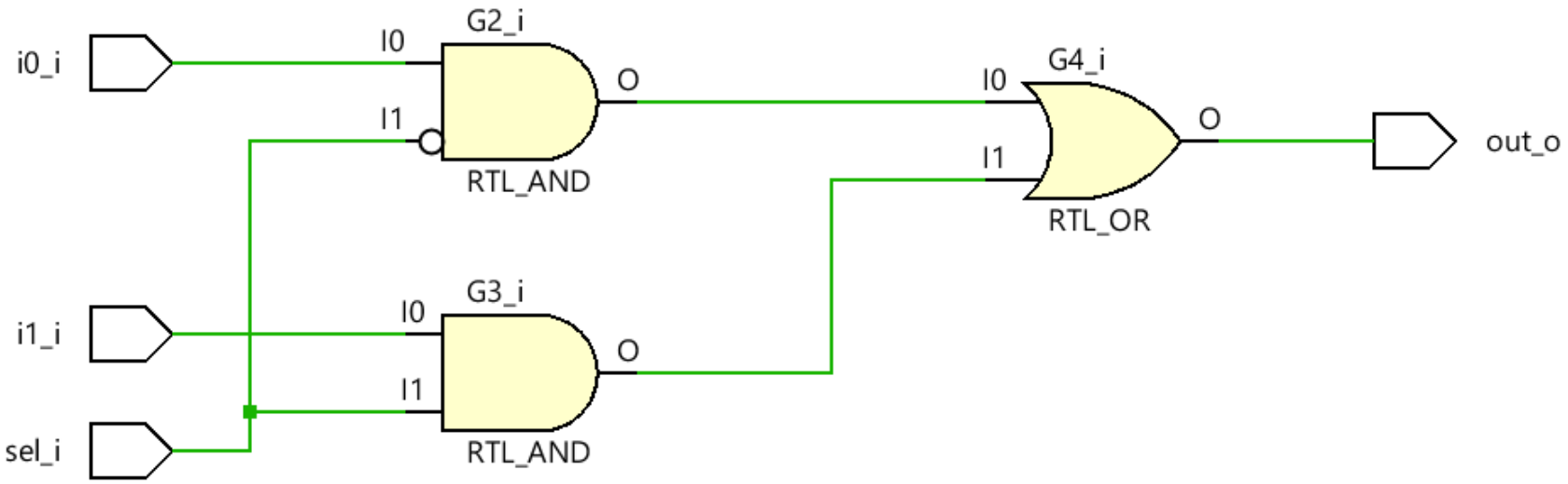
Multiplexer 2x1

out = sel’I0 + I1sel



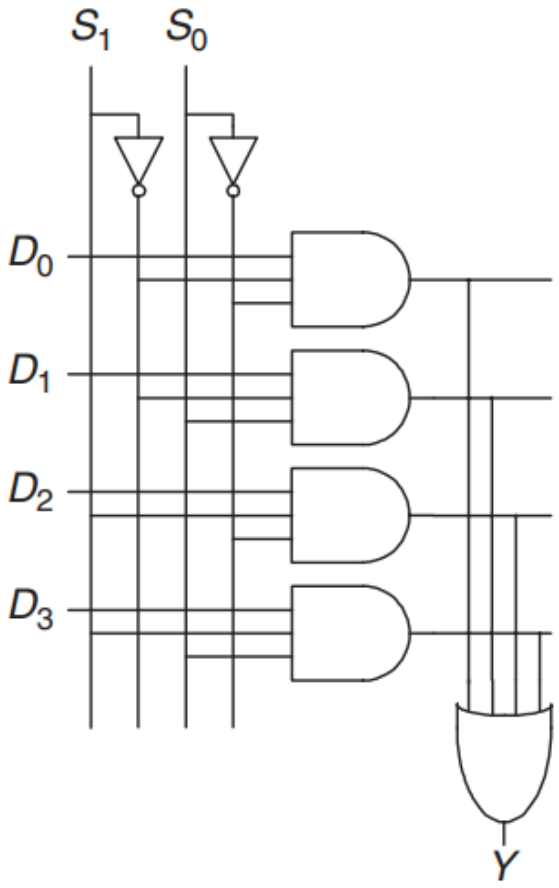

```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |LUT3 |     1|
|2     |IBUF |     3|
|3     |OBUF |     1|
+------+-----+------+
```

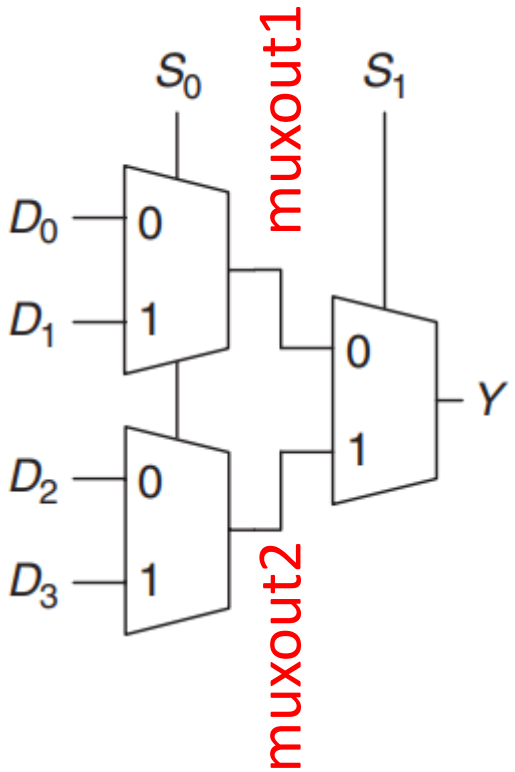
Vivado Elaboration Sonucu




```
module mux_2_1
(
input i0_i,
input i1_i,
input sel_i,
output out_o
);

wire not1, and1, and2;

not G1 (not1,sel_i);
and G2 (and1,i0_i,not1);
and G3 (and2,i1_i,sel_i);
or G4 (out_o,and1,and2);

endmodule
```

Multiplexer 4x1






```
module mux_4_1
(
input i0_i,
input i1_i,
input i2_i,
input i3_i,
input sel0_i,
input sel1_i,
output out_o
);

wire muxout1, muxout2;

mux_2_1 MUX1
(
.i0_i   (i0_i),
.i1_i   (i1_i),
.sel_i  (sel0_i),
.out_o  (muxout1)
);
```

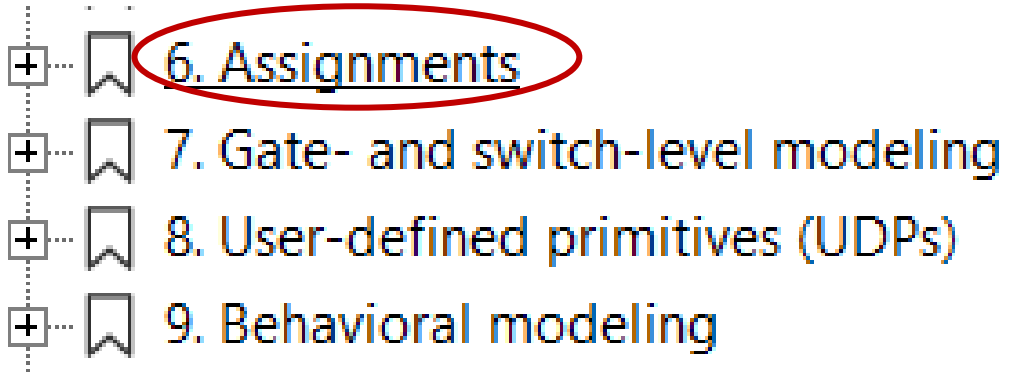
```
mux_2_1 MUX2
(
.i0_i   (i2_i),
.i1_i   (i3_i),
.sel_i  (sel0_i),
.out_o  (muxout2)
);

mux_2_1 MUX3
(
.i0_i   (muxout1),
.i1_i   (muxout2),
.sel_i  (sel1_i),
.out_o  (out_o)
);

endmodule
```

```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |LUT6 |     1|
|2     |IBUF |     6|
|3     |OBUF |     1|
+------+-----+------+
```

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**.

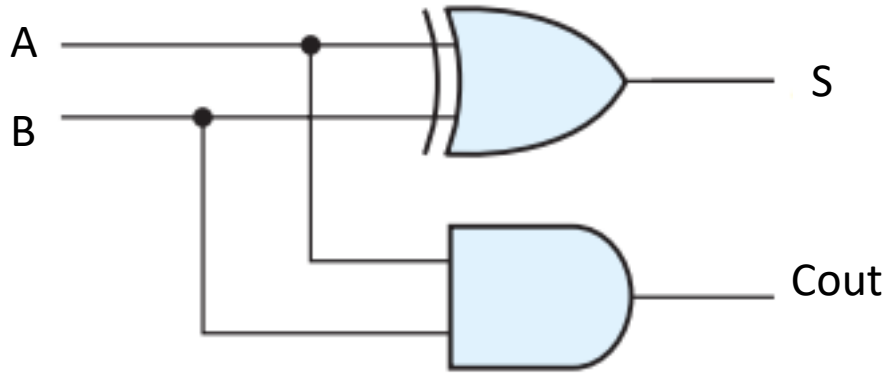


1364-2005 standard

Half Adder – Verilog Assign Keyword

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

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

```
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
```

Statement type	Left-hand side
Continuous assignment	Net (vector or scalar) Constant bit-select of a vector net Constant part-select of a vector net Constant indexed part-select of a vector net Concatenation or nested concatenation of any of the above left-hand side
Procedural assignment	Variables (vector or scalar) Bit-select of a vector reg, integer, or time variable Constant part-select of a vector reg, integer, or time variable Indexed part-select of a vector reg, integer, or time variable Memory word Concatenation or nested concatenation of any of the above left-hand side

wire

reg

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

Half Adder – Verilog Assign Keyword

{} {{}}	Concatenation, replication
unary + unary -	Unary operators
+ - * / **	Arithmetic
%	Modulus
> >= < <=	Relational
!	Logical negation
&&	Logical and
||	Logical or
==	Logical equality
!=	Logical inequality
===	Case equality
!==	Case inequality
~	Bitwise negation
&	Bitwise and
|	Bitwise inclusive or
^	Bitwise exclusive or
^~ or ~^	Bitwise equivalence
&	Reduction and
~&	Reduction nand

|	Reduction or
~|	Reduction nor
^	Reduction xor
~^ or ^~	Reduction xnor
<<	Logical left shift
>>	Logical right shift
<<<	Arithmetic left shift
>>>	Arithmetic right shift
? :	Conditional

```
module half_adder_assign
(
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);

assign s_o      = a_i ^ b_i;
assign cout_o   = a_i & b_i;

endmodule
```

Full Adder – Verilog Assign Keyword

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

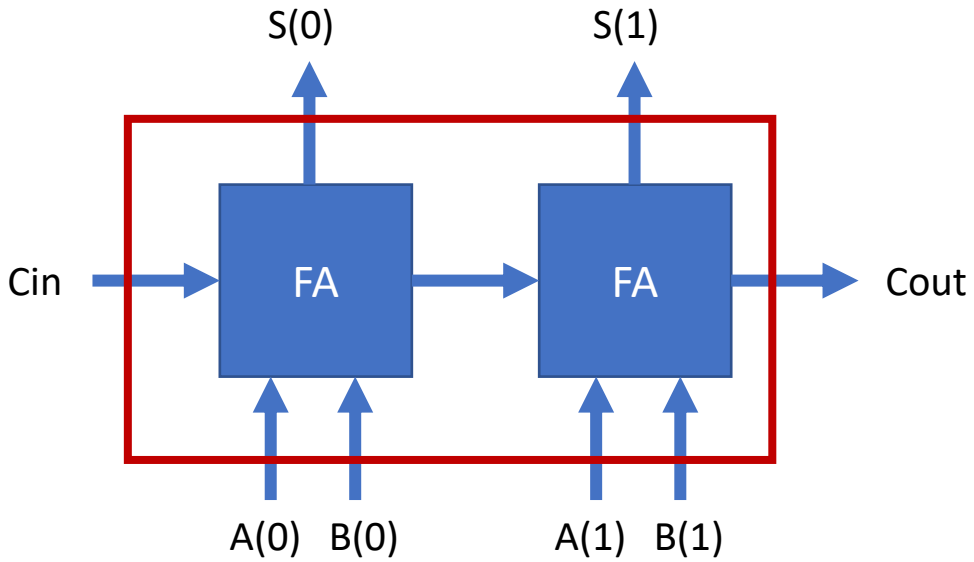
```
module full_adder_assign
(
input a_i,
input b_i,
input cin_i,
output s_o,
output cout_o
);

// 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);

assign {cout_o,s_o} = a_i + b_i + cin_i;

endmodule
```

{} {{}}	Concatenation, replication
unary + unary -	Unary operators
+ - * / **	Arithmetic
%	Modulus
> >= < <=	Relational
!	Logical negation
&&	Logical and
||	Logical or
==	Logical equality
!=	Logical inequality
===	Case equality
!==	Case inequality
~	Bitwise negation
&	Bitwise and
|	Bitwise inclusive or
^	Bitwise exclusive or
^~ or ~^	Bitwise equivalence
&	Reduction and
~&	Reduction nand

2-bit Binary Adder – Verilog Assign Keyword

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

2-bit Binary Adder




```
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
```

```
module binary_adder_2bit_assign
(
input [1:0] a_i,
input [1:0] b_i,
input cin_i,
output [1:0] s_o,
output cout_o
);

assign {cout_o,s_o} = a_i + b_i + cin_i;

endmodule
```

Multiplexer 2x1

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

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

==	Logical equality
? :	Conditional

```
module mux_2_1
(
input i0_i,
input i1_i,
input sel_i,
output out_o
);

wire not1, and1, and2;

not G1 (not1,sel_i);
and G2 (and1,i0_i,not1);
and G3 (and2,i1_i,sel_i);
or G4 (out_o,and1,and2);

endmodule
```

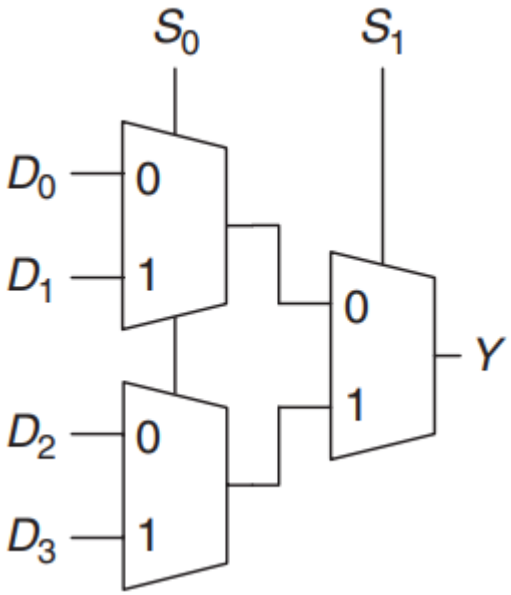
```
module mux_2_1_assign
(
input i0_i,
input i1_i,
input sel_i,
output out_o
);

// not G1 (not1,sel_i);
// and G2 (and1,i0_i,not1);
// and G3 (and2,i1_i,sel_i);
// or G4 (out_o,and1,and2);

assign out_o = (sel_i == 1'b0) ? i0_i : i1_i;

endmodule
```

Multiplexer 4x1

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
module mux_4_1
(
input i0_i,
input i1_i,
input i2_i,
input i3_i,
input sel0_i,
input sel1_i,
output out_o
);

wire muxout1, muxout2;

mux_2_1 MUX1
(
.i0_i   (i0_i),
.i1_i   (i1_i),
.sel_i  (sel0_i),
.out_o  (muxout1)
);
```

```
mux_2_1 MUX2
(
.i0_i   (i2_i),
.i1_i   (i3_i),
.sel_i  (sel0_i),
.out_o  (muxout2)
);

mux_2_1 MUX3
(
.i0_i   (muxout1),
.i1_i   (muxout2),
.sel_i  (sel1_i),
.out_o  (out_o)
);

endmodule
```



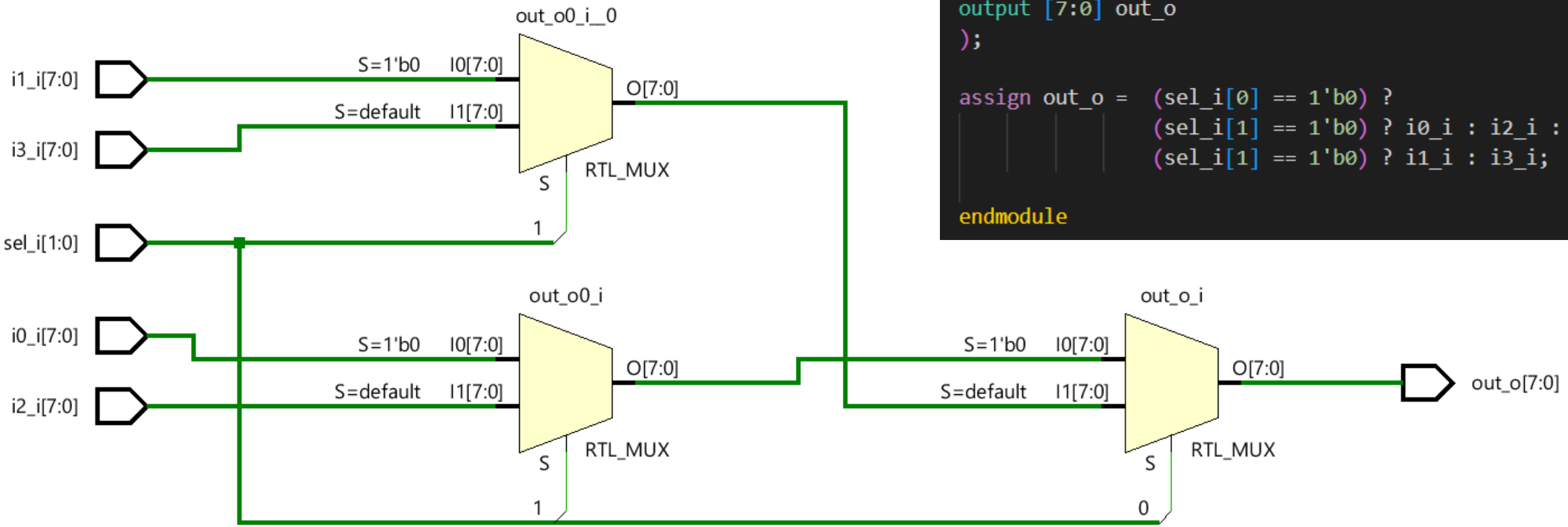
```
module mux_4_1_assign
(
input i0_i,
input i1_i,
input i2_i,
input i3_i,
input sel0_i,
input sel1_i,
output out_o
);

assign out_o =  (sel0_i == 1'b0) ?
                (sel1_i == 1'b0) ? i0_i : i2_i :
                (sel1_i == 1'b0) ? i1_i : i3_i;


endmodule
```

Multiplexer 4x1 |8-bit Genişlik

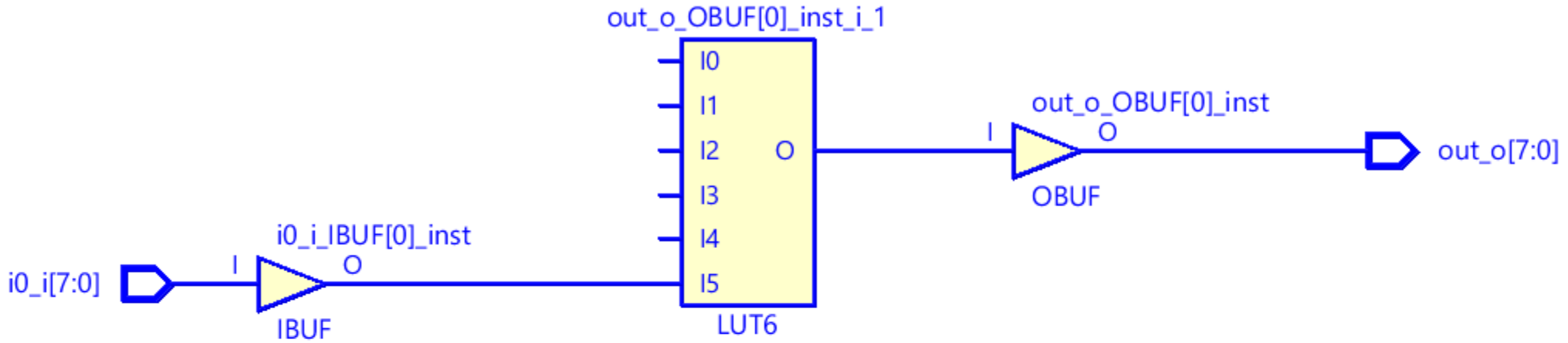
TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
module mux_4_1_8bit_assign
(
input [7:0] i0_i,
input [7:0] i1_i,
input [7:0] i2_i,
input [7:0] i3_i,
input [1:0] sel_i,
output [7:0] out_o
);

assign out_o =  (sel_i[0] == 1'b0) ?
                (sel_i[1] == 1'b0) ? i0_i : i2_i :
                (sel_i[1] == 1'b0) ? i1_i : i3_i;

endmodule
```

VIVADO ELABORATION



Multiplexer 4x1 |8-bit Genişlik



Nets Net Segm

Design Runs | Timing

Unconstrained Paths - NONE - NONE - Setup

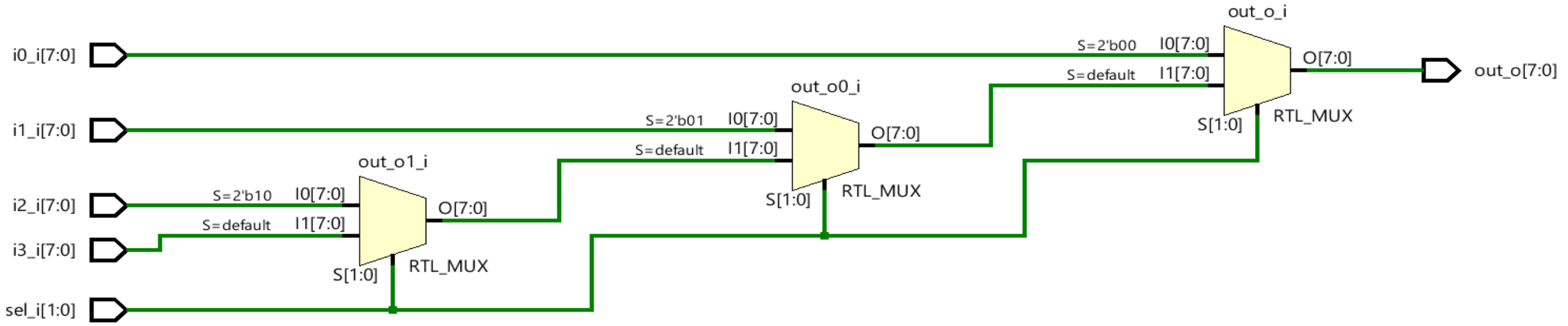
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Excep
Path 1	∞	3	4	1	i0_i[0]	out_o[0]	5.231	3.632	1.599	∞	input port clock		
Path 2	∞	3	4	1	i0_i[1]	out_o[1]	5.231	3.632	1.599	∞	input port clock		
Path 3	∞	3	4	1	i0_i[2]	out_o[2]	5.231	3.632	1.599	∞	input port clock		

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

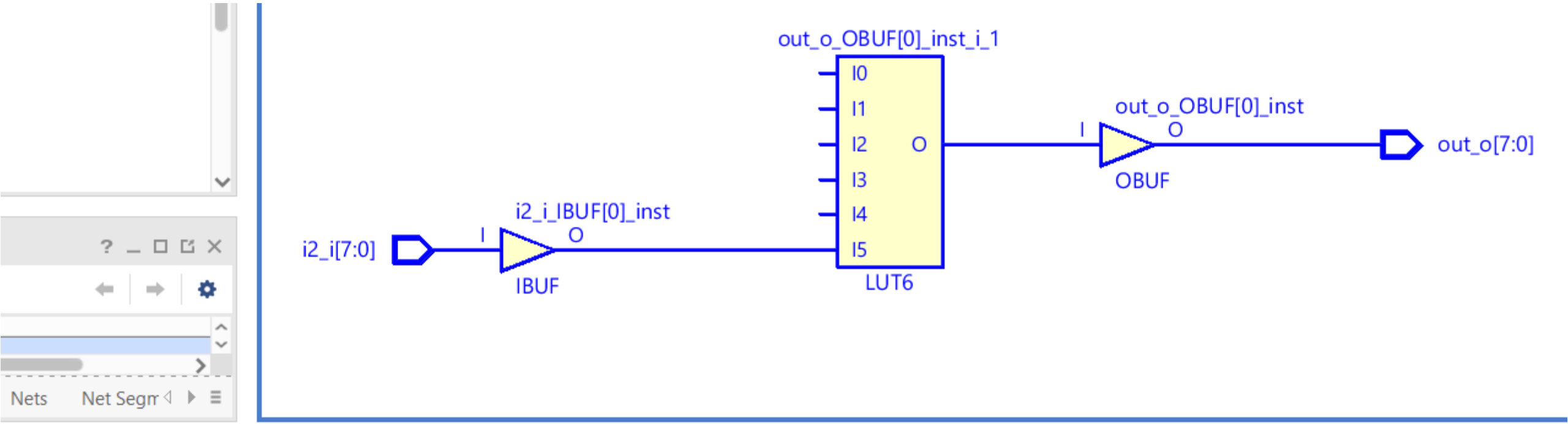
Multiplexer 4x1 |8-bit Genişlik Alternatif Kod

```
module mux_4_1_8bit_assign
(
input [7:0] i0_i,
input [7:0] i1_i,
input [7:0] i2_i,
input [7:0] i3_i,
input [1:0] sel_i,
output [7:0] out_o
);

//assign out_o =  (sel_i[0] == 1'b0) ?
//                (sel_i[1] == 1'b0) ? i0_i : i2_i :
//                (sel_i[1] == 1'b0) ? i1_i : i3_i;

assign out_o =  (sel_i == 2'b00) ? i0_i :
                (sel_i == 2'b01) ? i1_i :
                (sel_i == 2'b10) ? i2_i :
                                   i3_i;

endmodule
```

VIVADO ELABORATION



Multiplexer 4x1 |8-bit Genişlik
Alternatif Kod



Design Runs | Timing

Unconstrained Paths - NONE - NONE - Setup

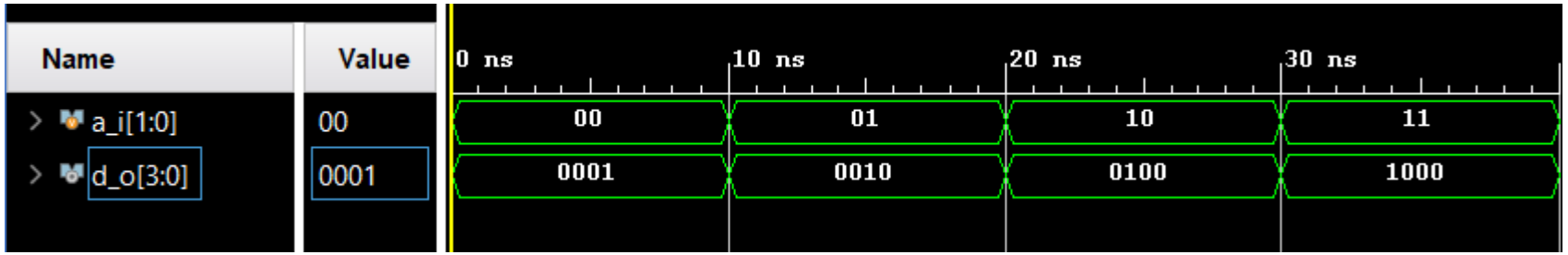
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Excep
Path 1	∞	3	4	1	i2_i[0]	out_o[0]	5.231	3.632	1.599	∞	input port clock		
Path 2	∞	3	4	1	i2_i[1]	out_o[1]	5.231	3.632	1.599	∞	input port clock		
Path 3	∞	3	4	1	i2_i[2]	out_o[2]	5.231	3.632	1.599	∞	input port clock		

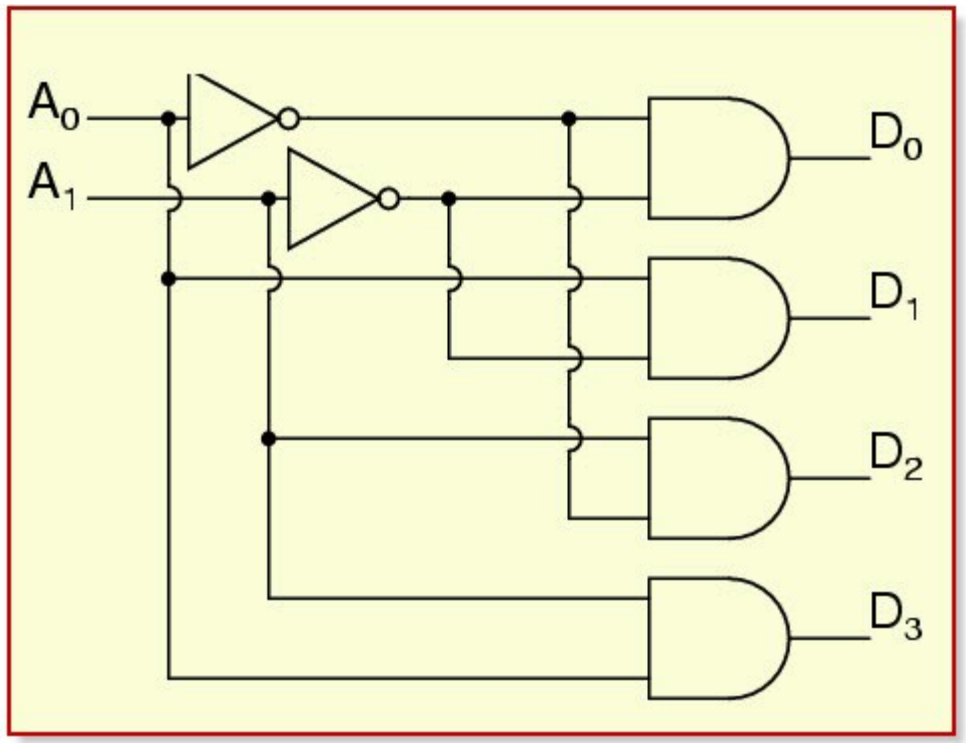
Decoder (Çözücü) 2-4

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

Giriş		Çıkış			
a0	a1	d0	d1	d2	d3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

n giriş, 2^n çıkış sinyali → Daha genel olarak n giriş m çıkış, m <= 2^n Sum-of-Minterm oluşturur



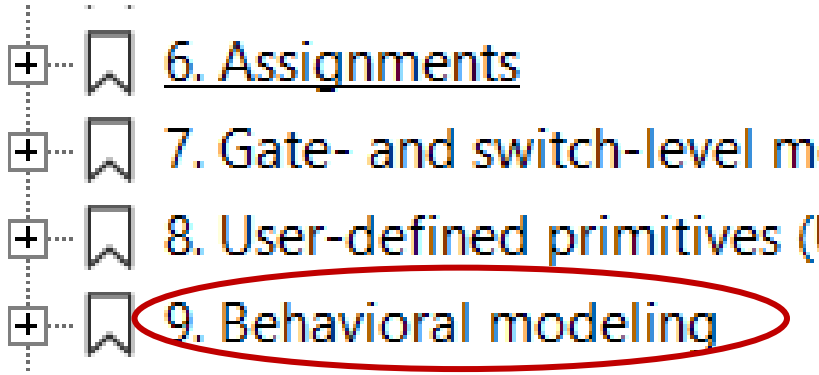



```
module decoder_2_4
(
input [1:0] a_i,
output [3:0] d_o
);


assign d_o =    (a_i == 2'b00) ? 4'b0001 :
                (a_i == 2'b01) ? 4'b0010 :
                (a_i == 2'b10) ? 4'b0100 :
                                 4'b1000;


endmodule
```

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

VERILOG HDL (Hardware Description Language)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

9. Behavioral modeling

1364-2005 standard

The language constructs introduced so far allow hardware to be described at a relatively detailed level. Modeling a circuit with logic gates and continuous assignments reflects quite closely the logic structure of the circuit being modeled; however, these constructs do not provide the power of abstraction necessary for describing complex high-level aspects of a system. The procedural constructs described in this clause are well suited to tackling problems such as describing a microprocessor or implementing complex timing checks.

9.2 Procedural assignments

As described in Clause 6, procedural assignments are used for updating **reg**, **integer**, **time**, **real**, **realtime**, and memory data types. There is a significant difference between procedural assignments and continuous assignments:

— Continuous assignments drive nets and are evaluated and updated whenever an input operand changes value.
— Procedural assignments update the value of variables under the control of the procedural flow constructs that surround them.

The right-hand side of a procedural assignment can be any expression that evaluates to a value. The left-hand side shall be a variable that receives the assignment from the right-hand side. The left-hand side of a procedural assignment can take one of the following forms:

— **reg, integer, real, realtime,** or **time** data type: an assignment to the name reference of one of these data types.
— Bit-select of a **reg, integer,** or **time** data type: an assignment to a single bit that leaves the other bits untouched.
— Part-select of a **reg, integer,** or **time** data type: a part-select of one or more contiguous bits that leaves the rest of the bits untouched.
— Memory word: a single word of a memory.
— Concatenation or nested concatenation of any of the above: a concatenation or nested concatenation of any of the previous four forms. Such specification effectively partitions the result of the right-hand expression and assigns the partition parts, in order, to the various parts of the concatenation or nested concatenation.

Half Adder – Verilog Behavioral Modeling (Davranışsal Modelleme)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

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

always @(a_i,b_i) begin
    {cout_o,s_o} = a_i + b_i;
end

endmodule
```

1364-2005 standard

6.2 Procedural assignments

The primary discussion of procedural assignments is in 9.2. However, a description of the basic ideas in this clause highlights the differences between continuous assignments and procedural assignments.

As stated in 6.1, continuous assignments drive nets in a manner similar to the way gates drive nets. The expression on the right-hand side can be thought of as a combinatorial circuit that drives the net continuously. In contrast, procedural assignments put values in variables. The assignment does not have duration; instead, the variable holds the value of the assignment until the next procedural assignment to that variable.

Procedural assignments occur within procedures such as **always**, **initial** (see 9.9), **task**, and **function** (see Clause 10) and can be thought of as "triggered" assignments. The trigger occurs when the flow of execution in the simulation reaches an assignment within a procedure. Reaching the assignment can be controlled by conditional statements. Event controls, delay controls, **if** statements, **case** statements, and looping statements can all be used to control whether assignments are evaluated. Clause 9 gives details and examples.

```
Synthesis (4 errors)
  [Synth 8-2576] procedural assignment to a non-register cout_o is not permitted [half_adder_behav.v:32] (1 more like this)
    [Synth 8-2576] procedural assignment to a non-register s_o is not permitted [half_adder_behav.v:32]
  [Synth 8-2132] illegal expression in target [half_adder_behav.v:32]
  [Common 17-69] Command failed: Vivado Synthesis failed
```

ÇÖZÜM ???

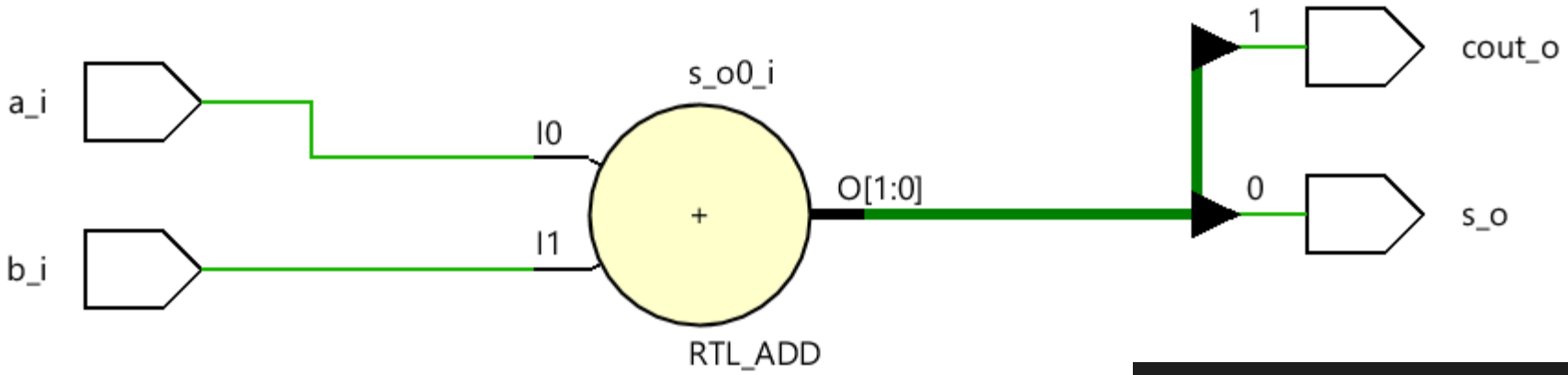
1364-2005 standard

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
blocking_assignment ::= (From A.6.2)
        variable_lvalue = [ delay_or_event_control ] expression
delay_control ::= (From A.6.5)
          # delay_value
        | # ( mintypmax_expression )
delay_or_event_control ::=
          delay_control
        | event_control
        | repeat ( expression ) event_control
event_control ::=
          @ hierarchical_event_identifier
        | @ ( event_expression )
        | @*
        | @ (*)
event_expression ::=
          expression
        | posedge expression
        | negedge expression
        | event_expression or event_expression
        | event_expression , event_expression
variable_lvalue ::= (From A.8.5)
          hierarchical_variable_identifier [ { [ expression ] } [ range_expression ] ]
        | { variable_lvalue { , variable_lvalue } }
```

Syntax 9-1—Syntax for blocking assignments

Half Adder – Verilog Behavioral Modeling (Davranışsal Modelleme)



2 YÖNTEM VAR

```
module half_adder_behav
(
input wire a_i,
input wire b_i,
output reg s_o,
output reg cout_o
);

always @(a_i,b_i) begin
    {cout_o,s_o} = a_i + b_i;
end

endmodule
```

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

reg s,cout;

always @(a_i,b_i) begin
    {cout,s} = a_i + b_i;
end

assign cout_o   = cout;
assign s_o      = s;

endmodule
```

Multiplexer – Verilog Behavioral Modeling (Davranışsal Modelleme)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
module mux_2_1
(
input i0_i,
input i1_i,
input sel_i,
output out_o
);

wire not1, and1, and2;

not G1 (not1,sel_i);
and G2 (and1,i0_i,not1);
and G3 (and2,i1_i,sel_i);
or G4 (out_o,and1,and2);

endmodule
```

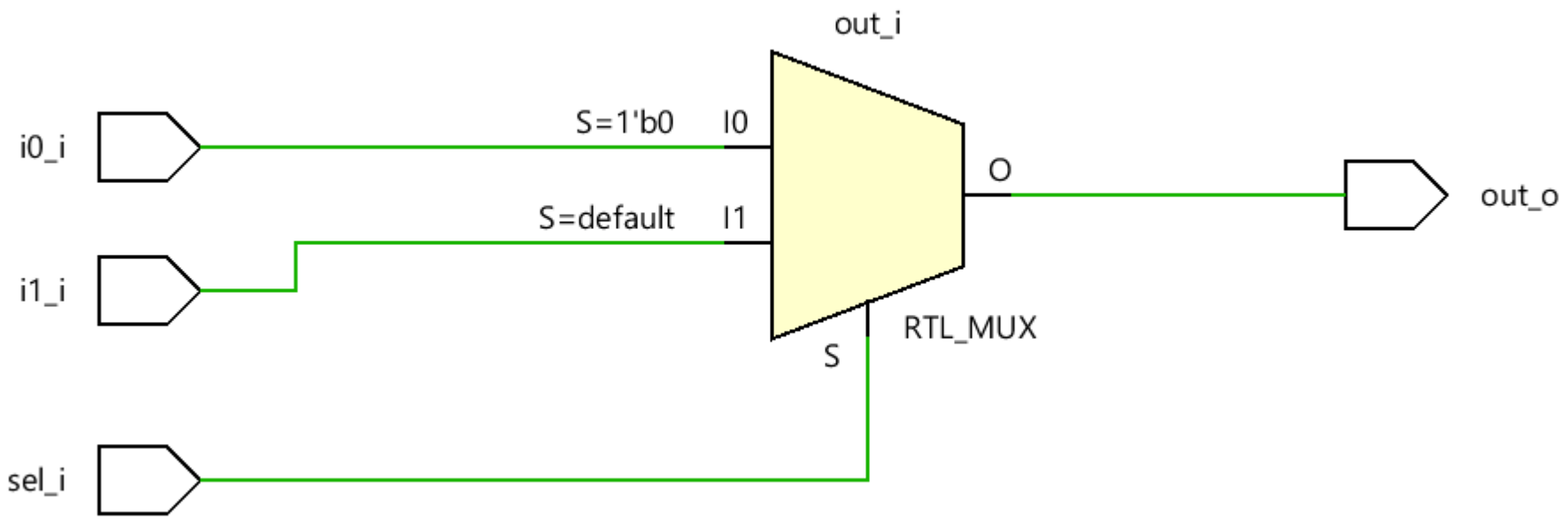
```
module mux_2_1_assign
(
input i0_i,
input i1_i,
input sel_i,
output out_o
);

// not G1 (not1,sel_i);
// and G2 (and1,i0_i,not1);
// and G3 (and2,i1_i,sel_i);
// or G4 (out_o,and1,and2);

assign out_o = (sel_i == 1'b0) ? i0_i : i1_i;

endmodule
```

```
module mux_2_1_behav
(
input i0_i,
input i1_i,
input sel_i,
output out_o
);

reg out;

always@(i0_i or i1_i or sel_i) begin
    if (sel_i == 1'b0) begin
        out = i0_i;
    end
    else begin
        out = i1_i;
    end
end

assign out_o = out;

endmodule
```

Half Adder – Verilog Behavioral Modeling (Davranışsal Modelleme)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

VIVADO ELABORATION



Decoder (Çözücü) 2-4

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
module decoder_2_4
(
input [1:0] a_i,
output [3:0] d_o
);


assign d_o =    (a_i == 2'b00) ? 4'b0001 :
                (a_i == 2'b01) ? 4'b0010 :
                (a_i == 2'b10) ? 4'b0100 :
                                 4'b1000;


endmodule
```

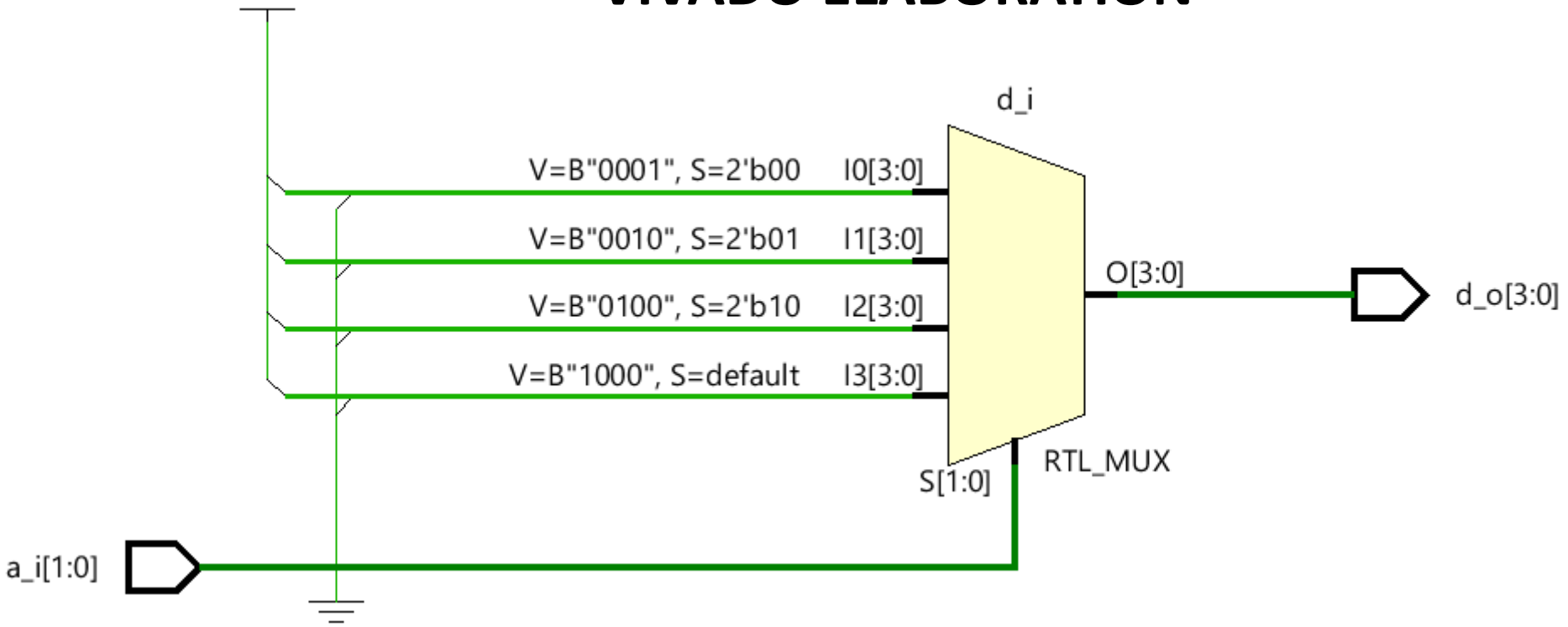
```
module decoder_2_4_behav
(
input [1:0] a_i,
output [3:0] d_o
);

reg [3:0] d;

always @(a_i) begin
    case (a_i)
        2'b00   : d = 4'b0001;
        2'b01   : d = 4'b0010;
        2'b10   : d = 4'b0100;
        default : d = 4'b1000;
    endcase
end

assign d_o = d;

endmodule
```

Giriş		Çıkış			
a0	a1	d0	d1	d2	d3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Decoder (Çözücü) 2-4

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

VIVADO ELABORATION

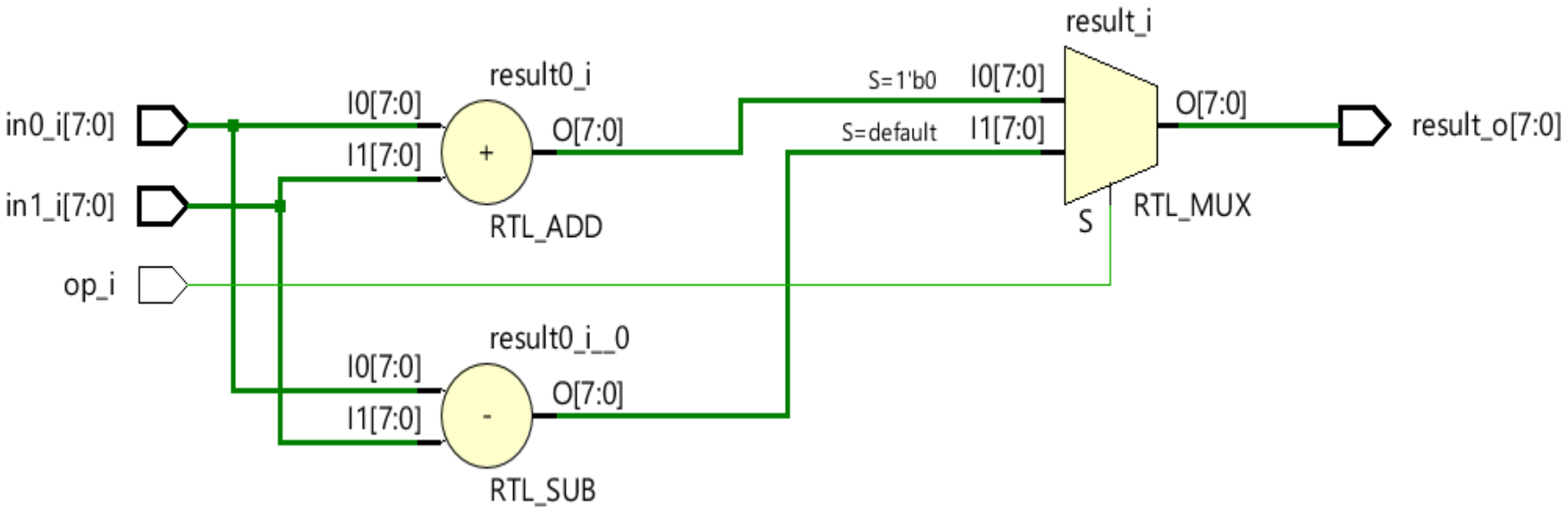


8-bit Toplayıcı/Çıkarıcı (Adder/Subtractor)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
Report Cell Usage:
+------+-------+------+
|      |Cell   |Count |
+------+-------+------+
|1     |CARRY4 |     2|
|2     |LUT1   |     1|
|3     |LUT3   |     7|
|4     |IBUF   |    17|
|5     |OBUF   |     8|
+------+-------+------+
```

VIVADO ELABORATION




```
module add_sub_8bit
(
input [7:0] in0_i,
input [7:0] in1_i,
input op_i,
output [7:0] result_o
);

reg [7:0] result;

always @(in0_i, in1_i, op_i) begin
    if (op_i == 1'b0) begin
        result = in0_i + in1_i;
    end
    else begin
        result = in0_i - in1_i;
    end
end

assign result_o = result;

endmodule
```

8-bit Toplayıcı/Çıkarıcı (Adder/Subtractor)

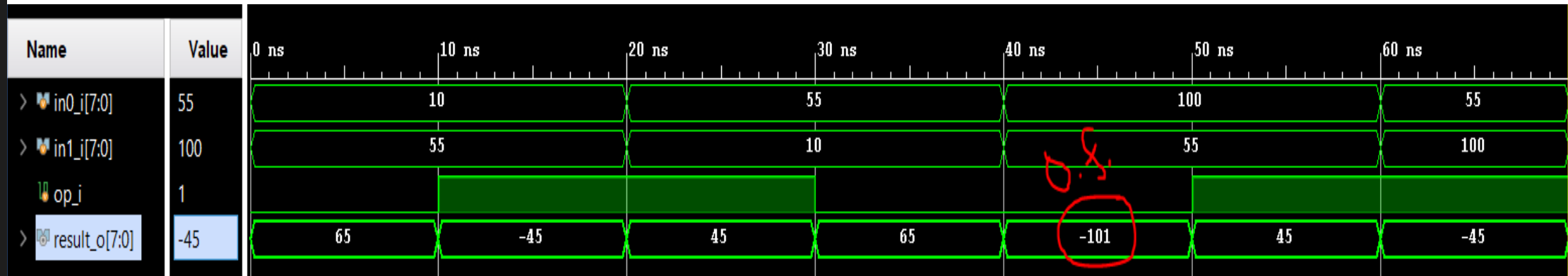
TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
initial begin
in0_i = 8'd10;
in1_i = 8'd55;
op_i = 1'b0;
#10;
in0_i = 8'd10;
in1_i = 8'd55;
op_i = 1'b1;
#10;
in0_i = 8'd55;
in1_i = 8'd10;
op_i = 1'b1;
#10;
in0_i = 8'd55;
in1_i = 8'd10;
op_i = 1'b0;
#10;
in0_i = 8'd100;
in1_i = 8'd55;
op_i = 1'b0;
#10;
in0_i = 8'd100;
in1_i = 8'd55;
op_i = 1'b1;
#10;
in0_i = 8'd55;
in1_i = 8'd100;
op_i = 1'b1;
#10;
$finish;
end
```

8-bit → -128 < … < 127

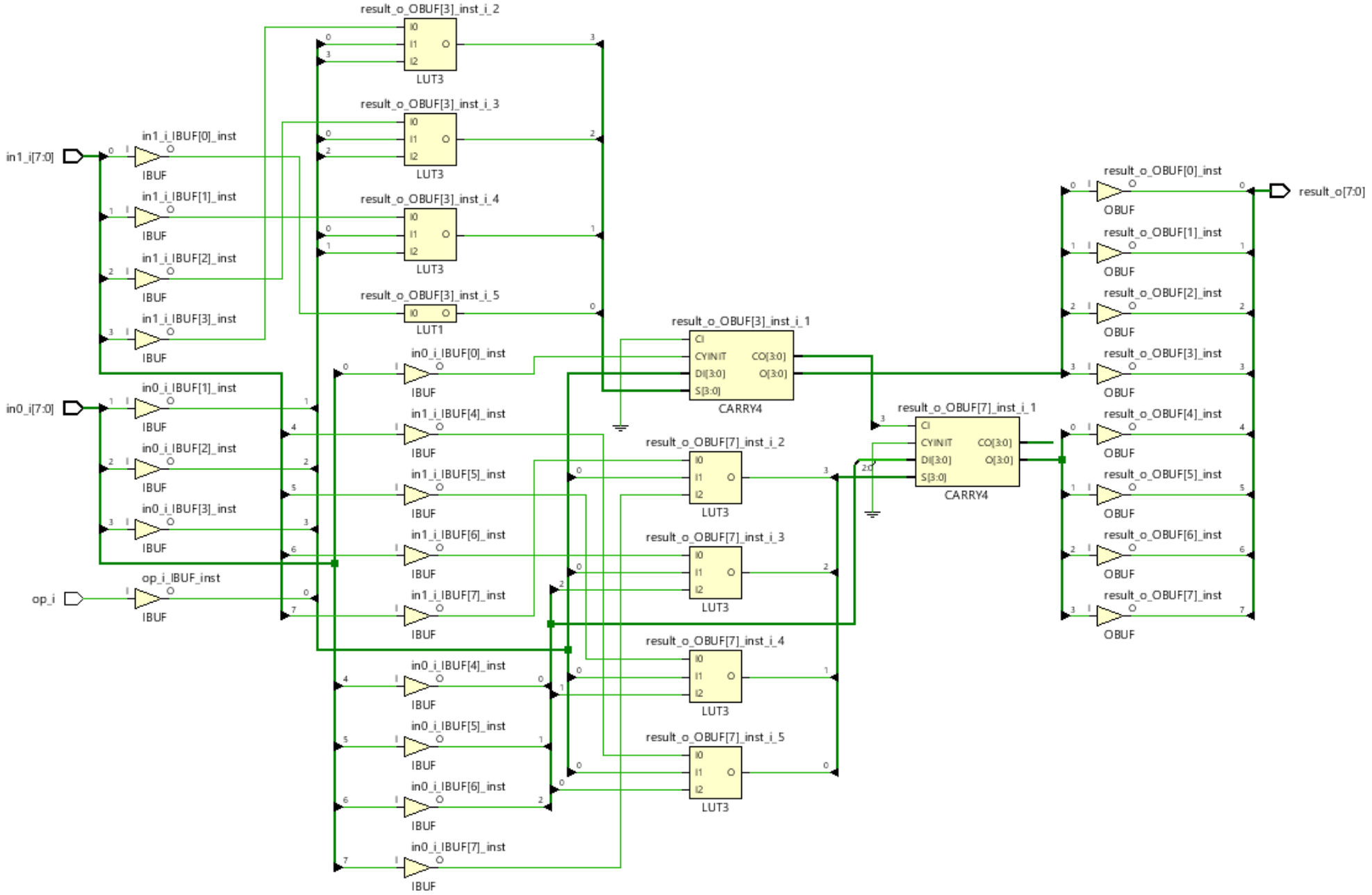
155: 10011011 = -128+16+8+2+1 = -101

Overflow (Taşma)



8-bit Toplayıcı/Çıkarıcı (Adder/Subtractor)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

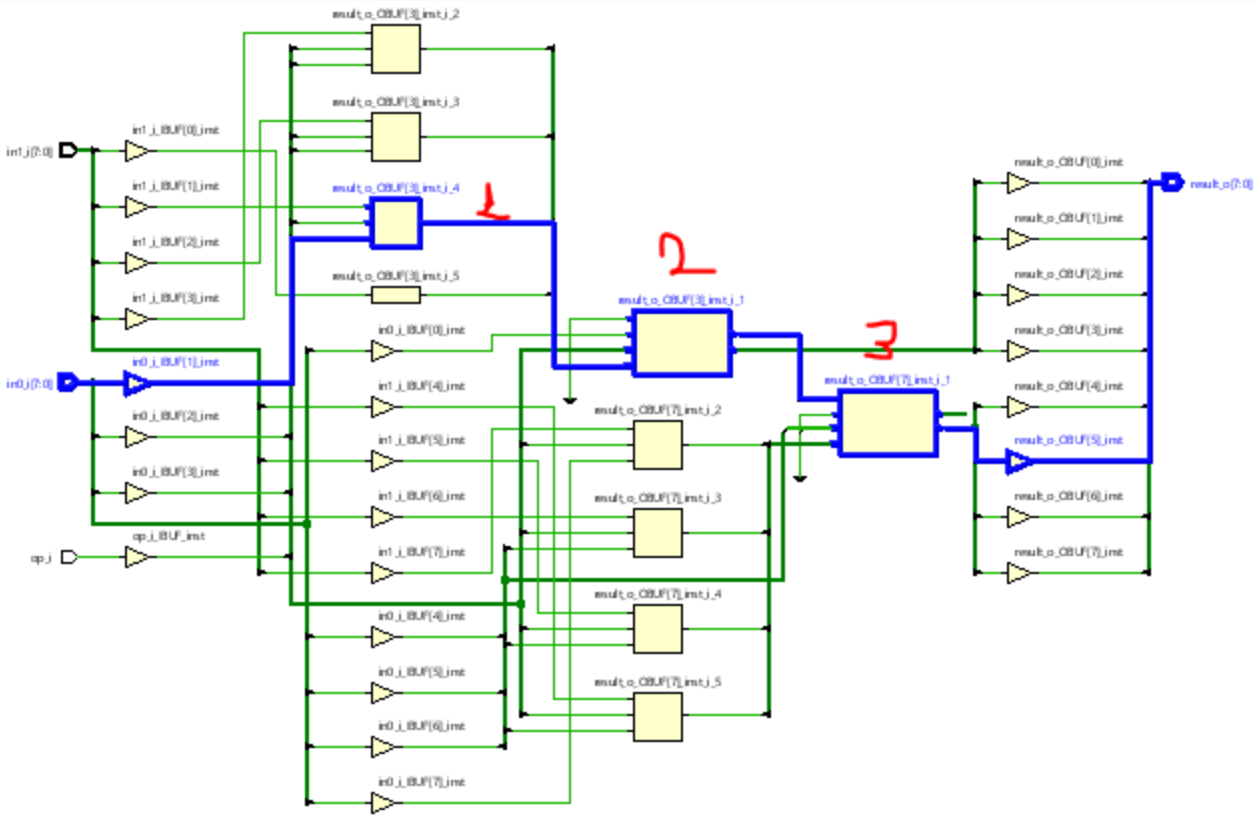


8-bit Toplayıcı/Çıkarıcı (Adder/Subtractor)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

Project Summary | Device | Schematic

35 Cells | 25 I/O Ports | 66 Nets



Timing

Unconstrained Paths - NONE - NONE - Setup

Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Des
∞	5	6	2	in0_i[1]	result_o[5]	6.317	4.709	1.608	∞	input port clock	

always @* | always @(*)

TOBB ETÜ
Ekonomi ve Teknoloji Üniversitesi

```
module add_sub_8bit
(
input [7:0] in0_i,
input [7:0] in1_i,
input op_i,
output [7:0] result_o
);

reg [7:0] result;

//always @(in0_i, in1_i, op_i) begin
always @(*) begin
    if (op_i == 1'b0) begin
        result = in0_i + in1_i;
    end
    else begin
        result = in0_i - in1_i;
    end
end

assign result_o = result;

endmodule
```