



Introduction to Embedded Systems, Lecture 3

Digital circuits design with Verilog. Combinatorial logic

Dmitry Zaitsev

<http://daze.ho.ua>

Combinatorial logic

- specify functioning by truth table
- synthesize DNF for each bit of result
- minimize DNFs
- draw scheme using gates
- ✓ sometimes we use restricted set of gates
- ✓ sometimes we start with specification using formulae

Standard forms of Boolean algebra

- Disjunctive normal form

$$(A \wedge \neg B \wedge \neg C) \vee (\neg D \wedge E \wedge F)$$

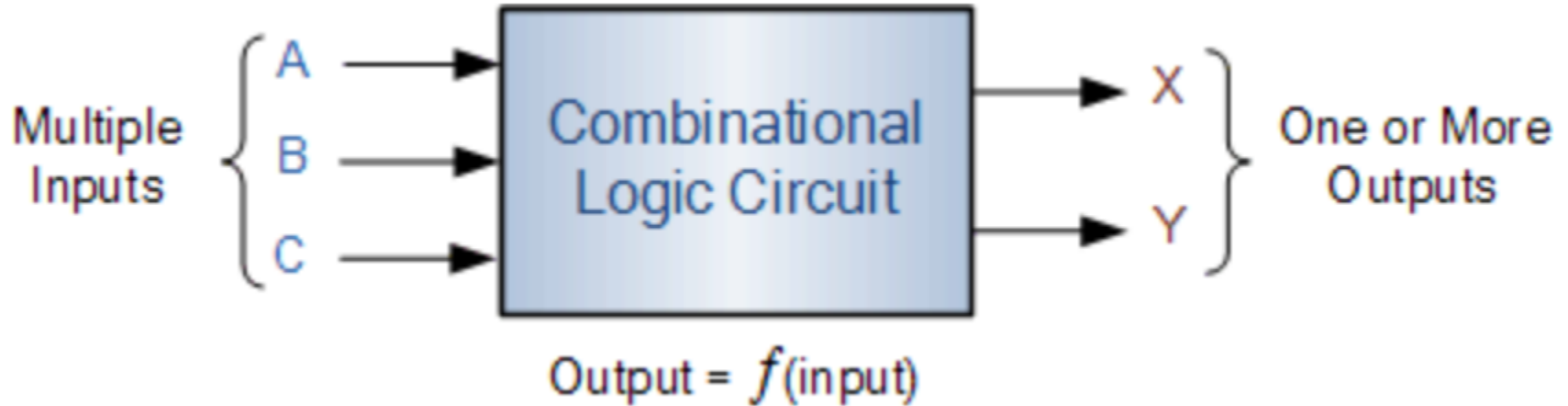
- Conjunctive normal form

$$(A \vee \neg B \vee \neg C) \wedge (\neg D \vee E \vee F)$$

- Polynomial of Zhegalkin – algebraic normal form








$$1 \oplus A \oplus ABD.$$

Combinational Logic Circuits

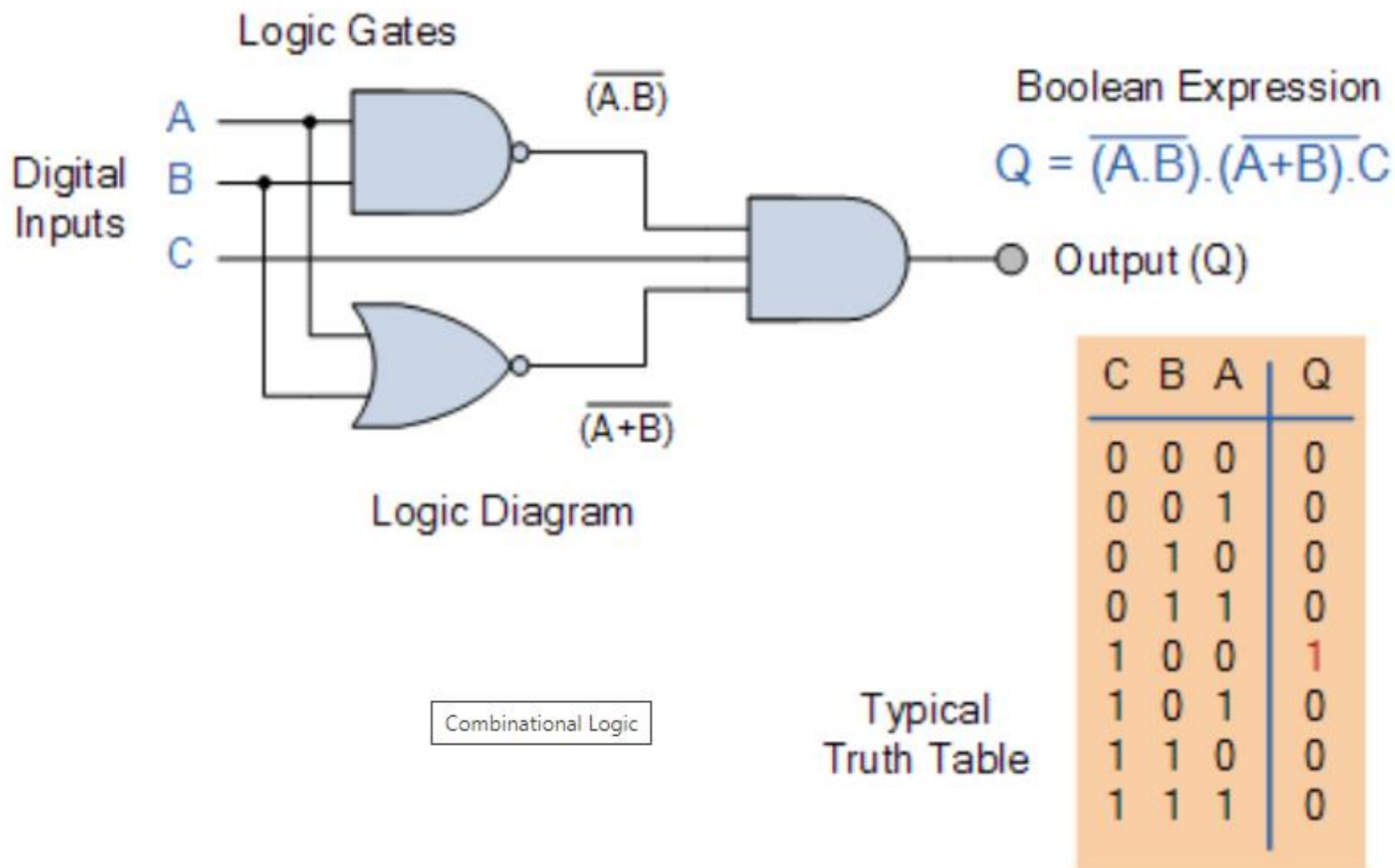


The outputs of **Combinational Logic Circuits** are only determined by the logical function of their current input state

Logic gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR																																																																																																
Alg. Expr.	\overline{A}	AB	\overline{AB}	$A+B$	$\overline{A+B}$	$A\oplus B$	$\overline{A\oplus B}$																																																																																																
Symbol																																																																																																							
Truth Table	<table><tr><th>A</th><th>X</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	X	0	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	0	0	1	0	1	0	0	1	1	1	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	1	0	1	1	1	0	1	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	1	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	0	<table><tr><th>B</th><th>A</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	1
A	X																																																																																																						
0	1																																																																																																						
1	0																																																																																																						
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					

Logic diagram



Combinational Logic Circuit

```
graph TD; A[Combinational Logic Circuit] --> B[Arithmetic & Logical Functions]; A --> C[Data Transmission]; A --> D[Code Converters]; B --> B1[Adders<br/>Subtractors<br/>Comparators<br/>PLD's]; C --> C1[Multiplexers<br/>Demultiplexers<br/>Encoders<br/>Decoders]; D --> D1[Binary<br/>BCD<br/>7-segment];
```

Arithmetic &
Logical Functions

Adders
Subtractors
Comparators
PLD's

Data
Transmission

Multiplexers
Demultiplexers
Encoders
Decoders

Code
Converters

Binary
BCD
7-segment

Verilog

- hardware description language (HDL)
- design and verification of digital circuits
- syntax similar to C
- standardized as IEEE 1364
- tools: ModelSim (siemens), VeriLogger, Verilator etc

Verilog features

- specify schemes with modules and formulae
- hierarchical design: using modules inside modules
- automatic graphical layout of scheme using gates
- automated simulation and testing
- final generation of technological specifications to produce chips

Verilog module

A Verilog module is a building block that defines a design or testbench component, by defining the building block's ports and internal behaviour. Higher-level modules can embed lower-level modules to create hierarchical designs. Different Verilog modules communicate with each other through Verilog port.

Defining a Verilog Module

- Keyword *module* to begin the definition
- Identifier that is the name of the module
- Optional list of parameters
- Optional list of ports (to be addressed more deeply in a future article)
- Module item
- Keyword *endmodule* to end the definition

Half adder

- One bit adder
- Input:
 - bit x
 - bit y
- Output ($s=x+y$):
 - bit s – sum
 - bit c – carry

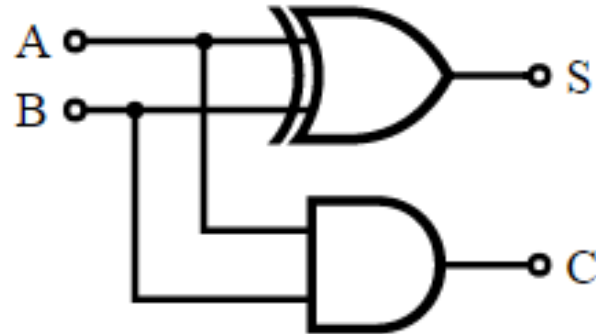
$$\begin{array}{r} 1 \\ + \\ 1 \\ \hline 10 \\ \text{CS} \end{array}$$

Half adder design

Half Adder Truth Table			
A	B	Carry	Sum
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

$$S = A \oplus B$$

$$C = A \wedge B$$



Half adder module

```
module half_adder1
(  input x,
    input y,
    output s,
    output c  );

    assign s = x^y;
    assign c = x&y;

endmodule
```

Verilog online: <http://digitaljs.tilk.eu>

Verilog Module for Design and Te x Half Adder - Nandland x DigitalJS Online x +

← → ↻ ⚠ Not secure | digitaljs.tilk.eu 🔍 📄 ☆ 🛡 🧩 🖨 👤 ⋮

🔊 ⏸ ⏮ ⏭ ⏪ ⏩ ⌚ 3205 📁 💾 🔗

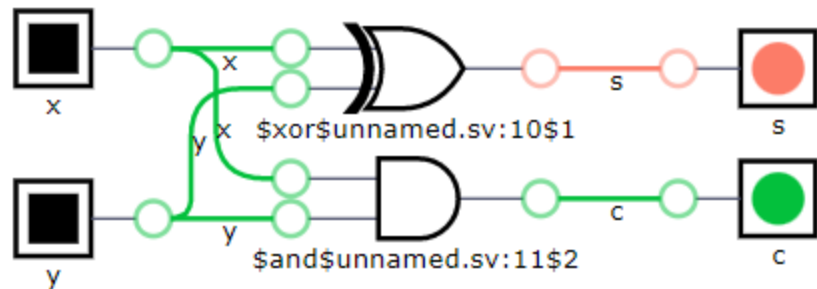
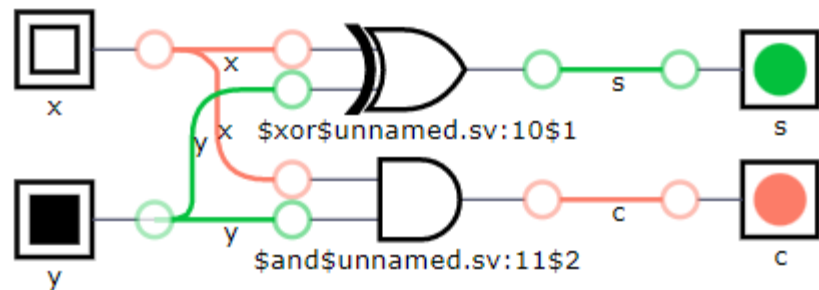
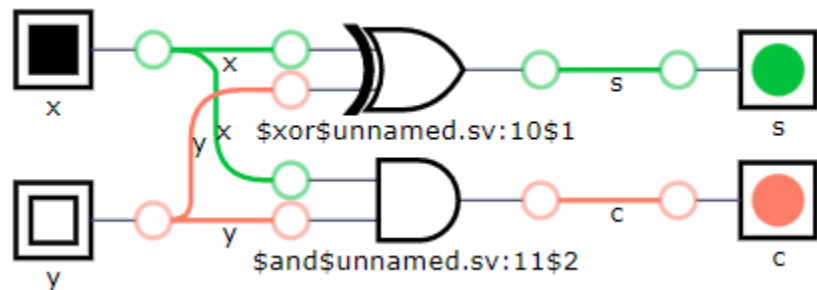
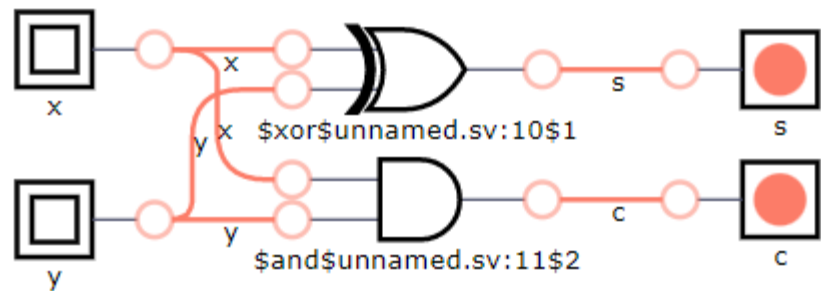
Setup I/O unnamed.sv 🗑

```
1 module half_adder1
2 (
3   input x,
4   input y,
5
6   output s,
7   output c
8 );
9
10 assign s = x^y;
11 assign c = x&y;
12
13 endmodule
14
```

Synthesize and simulate!

Windows Taskbar: Type here to search, 65°F Partly sunny, 5:44 PM, 4/25/2023

Test half adder



Full adder

- One bit adder
- Input:
 - bit x
 - bit y
 - input carry c_{in}
- Output ($s=x+y+c_{in}$):
 - bit s – sum
 - bit c_{out} – output carry

$$\begin{array}{r} 1 c_{in} \\ + \\ 1 x \\ + \\ 1 y \\ \hline 11 \\ c_s \end{array}$$

Full adder design

x	y	c_in	c_out	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$s = (x \oplus y) \oplus c_{in}$$

$$c_{out} = ((x \oplus y) \wedge c_{in}) \oplus (x \wedge y)$$

```
module fulladder1
( input x,
  input y,
  input c_in,

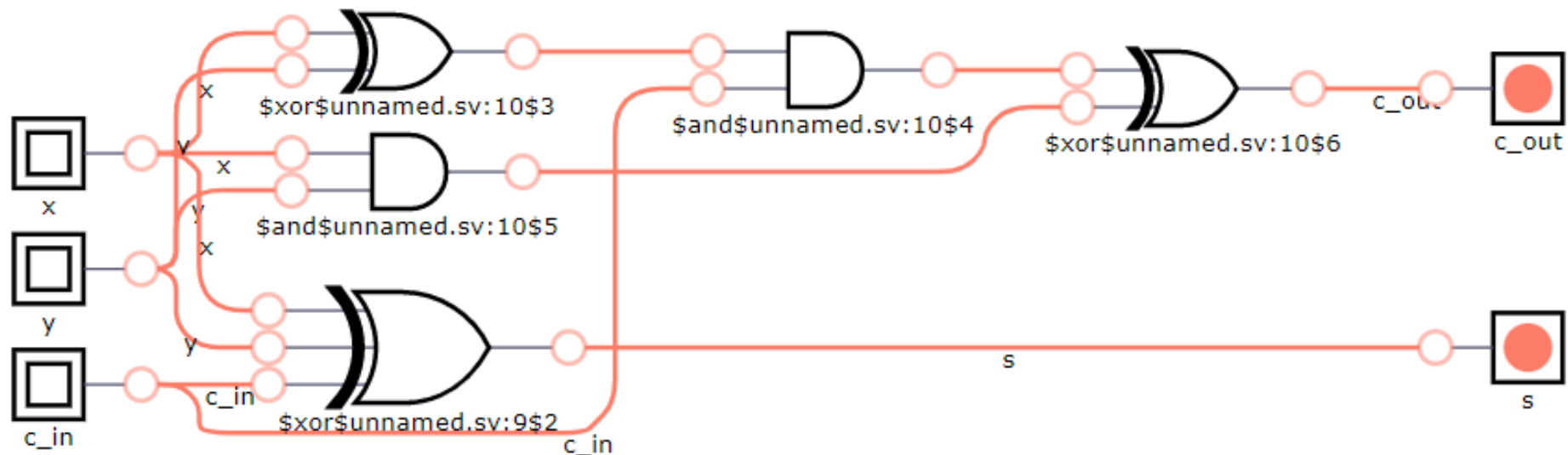
  output s,
  output c_out );

  assign s = (x^y) ^ c_in;
  assign c_out = ((x^y) & c_in) ^ (x&y);

endmodule
```

Full adder module

Full adder scheme



Hierarchical design

- specify a bus as an array of wires
- specify components
- use components
- connect input and output wires of components

Four bits sequential adder

- idea: connect 4 full adders sequentially
- c_out of an adder to c_in of the next adder
- c_in of the first feed with 0
- use c_out of the last as the resulting carry
- example:

```
    1001
    1101
  -----
    10110
```

Four bits sequential adder module

```
module Adder4(  
    input [3:0] a,  
    input [3:0] b,  
    output [3:0] sum,  
    output carry  
);  
    wire cin;  
  
    assign cin = 1'b0;  
    fulladder1 s0( .x( a[0] ), .y( b[0]), .c_in( cin ), .s( sum[0]), .c_out( ripple0 ) );  
    fulladder1 s1( .x( a[1] ), .y( b[1]), .c_in( ripple0 ), .s( sum[1]), .c_out( ripple1 ) );  
    fulladder1 s2( .x( a[2] ), .y( b[2]), .c_in( ripple1 ), .s( sum[2]), .c_out( ripple2 ) );  
    fulladder1 s3( .x( a[3] ), .y( b[3]), .c_in( ripple2 ), .s( sum[3]), .c_out( carry ) );  
  
endmodule
```

Automatic layout

DigitalJS Online

Not secure | digitaljs.tilk.eu

535

Setup I/O unnamed.sv

```
1 module fulladder1
2 (
3   input x,
4   input y,
5   input c_in,
6
7   output s,
8   output c_out
9 );
10
11 assign s = (x^y) ^ c_in;
12 assign c_out = ((x^y) & c_in
13
14 endmodule
15
16 module Adder4(
17   input [3:0] a,
18   input [3:0] b,
19   output [3:0] sum,
20   output carry
21 );
22 wire cin;
23
24 assign cin = 1'b0;
25 fulladder1 s0(.x( a[0] )
26 fulladder1 s1(.x( a[1] )
27 fulladder1 s2(.x( a[2] )
```

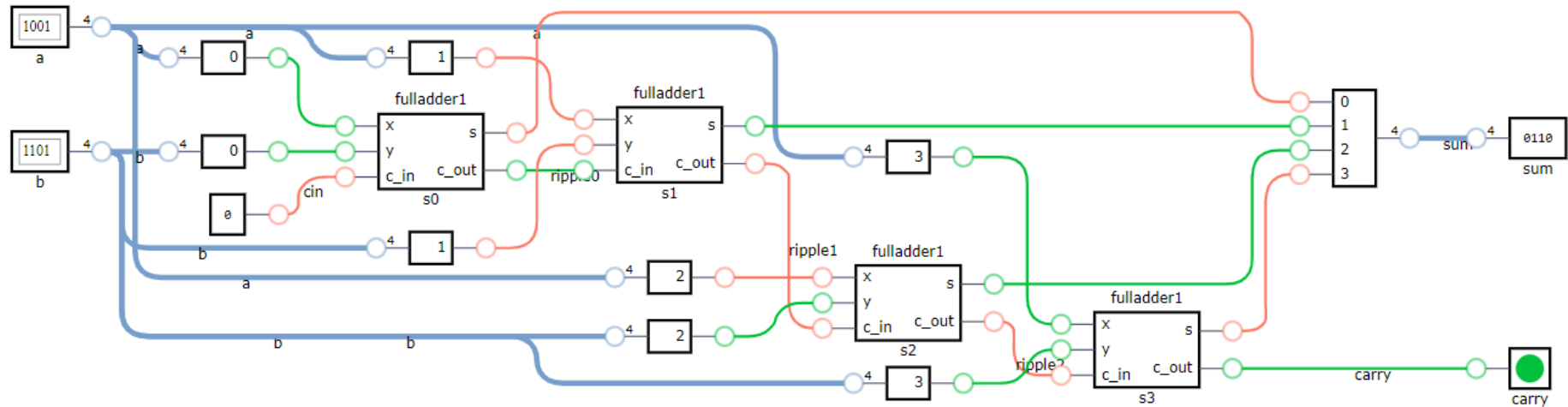
Synthesize and simulate!

Activate Windows
Go to Settings to activate Windows.

Type here to search

66°F Sunny 2:01 PM 4/26/2023

Manual testing



Show components

DigitalJS Online

Not secure | digitaljs.tilk.eu

3601

fulladder1 s1

Activate Windows
Go to Settings to activate Windows.

Type here to search

66°F Mostly sunny 2:42 PM 4/26/2023

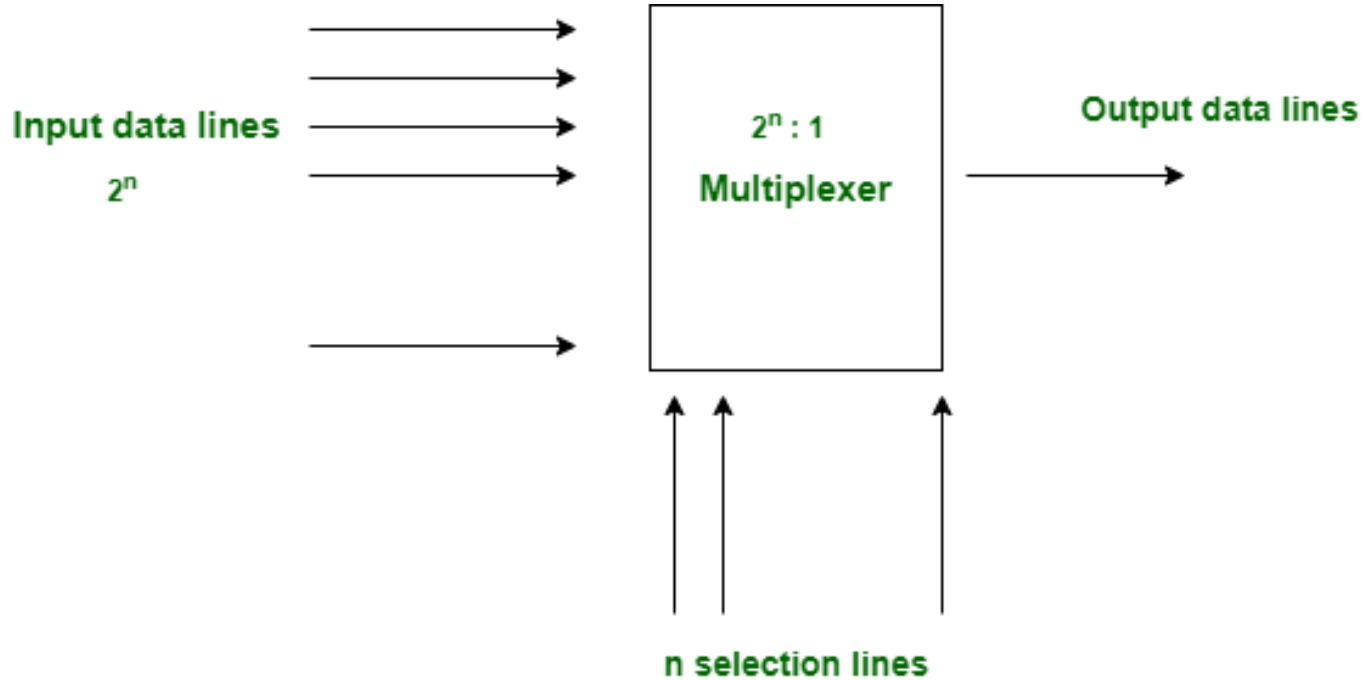
Pros and contras sequential composition

- Pros:
 - simplicity, regularity of design
- Contras:
 - long time of work, total delay grows linearly with respect to number of bits

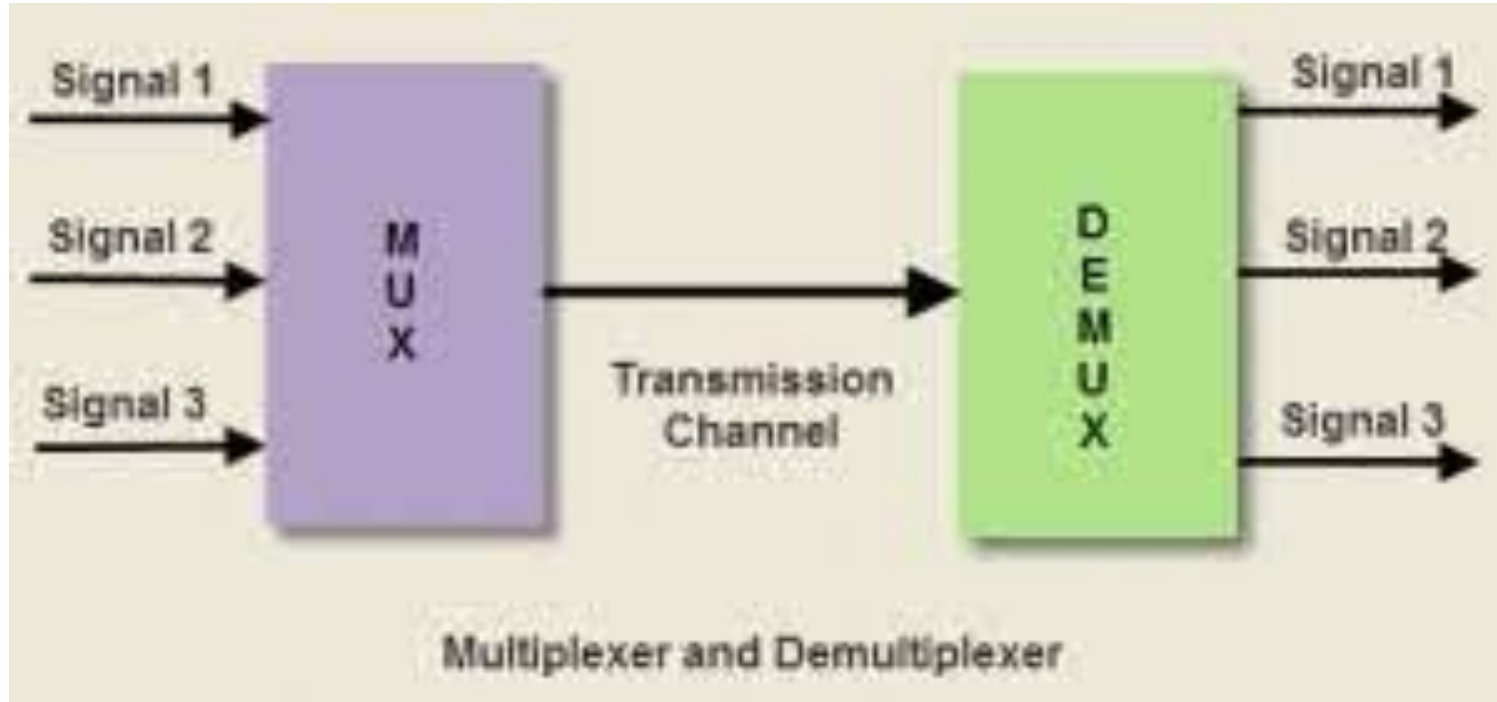
Overview of basic combinatorial circuits

- Adder: $x+y$
- Subtractor: $x-y$ (addition with 2's complement)
- Comparator: $x < y$ (sign of subtraction)
- Multiplexer: selects an input from several inputs
- Demultiplexer: switch single input between several outputs
- Encoder: unary code to binary code (2^n to n)
- Decoder: binary code to unary code (n to 2^n)

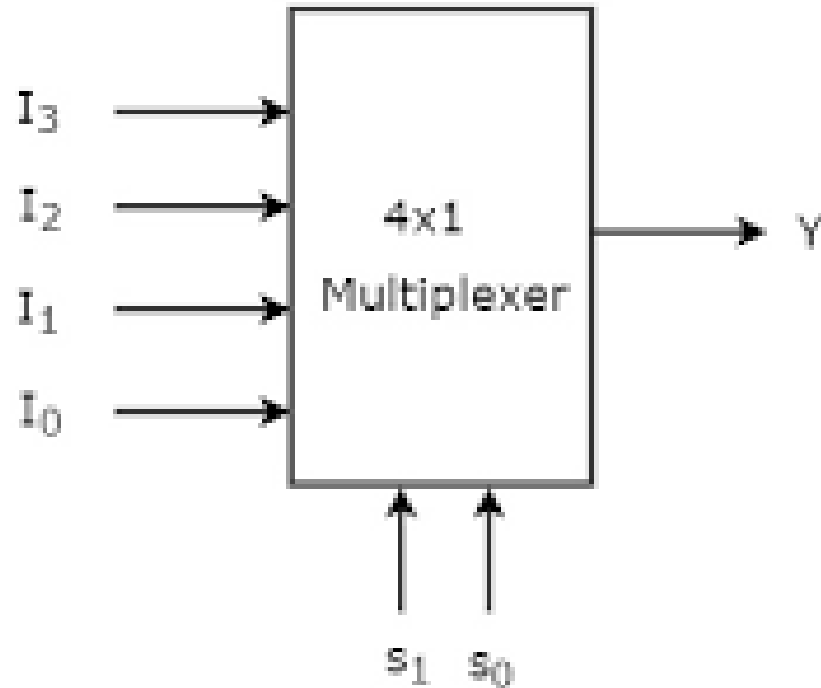
Multiplexer



Multiplexer and demultiplexer



4x1 Multiplexer

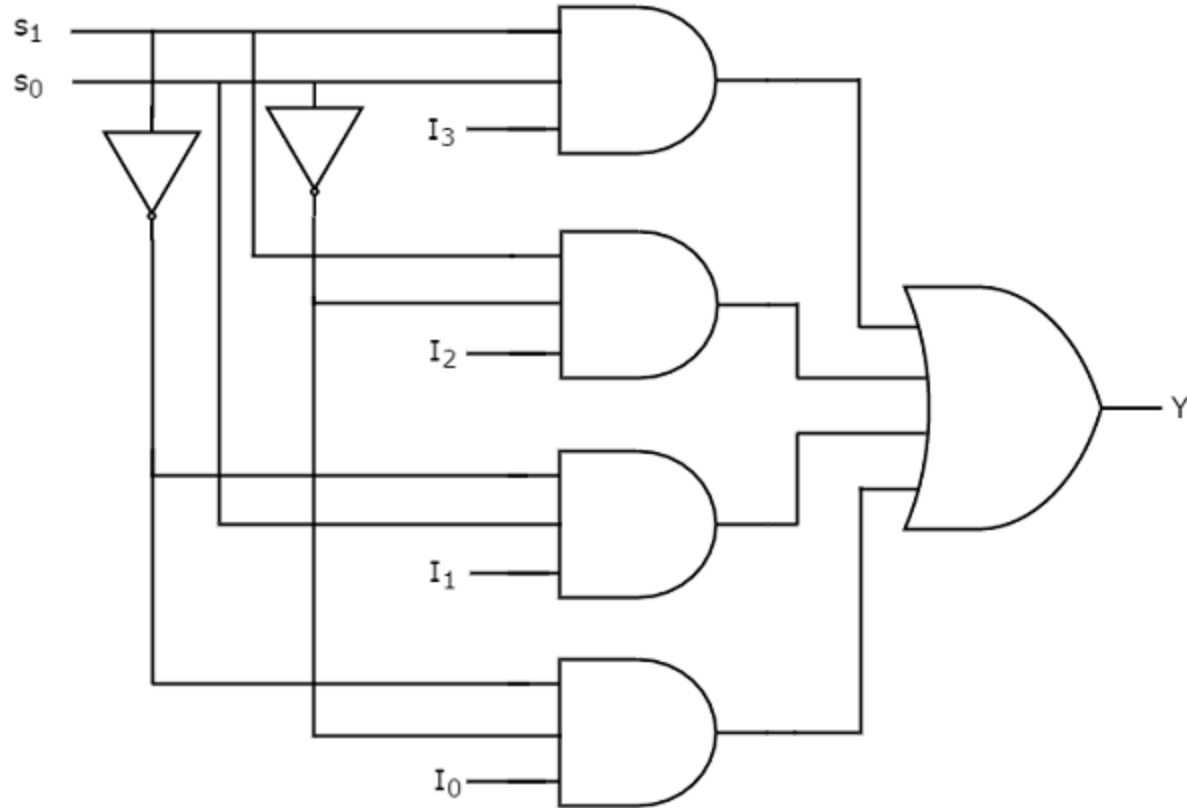


4x1 multiplexer design

s1	s0	y
0	0	i0
0	1	i1
1	0	i2
1	1	i3

$$y = i_0 \bar{s}_1 \bar{s}_0 \vee i_1 \bar{s}_1 s_0 \vee i_2 s_1 \bar{s}_0 \vee i_3 s_1 s_0$$

4x1 Multiplexer layout



4x1 Multiplexor Verilog module

```
module MUX4x1(  
    input [3:0] i,  
    input [1:0] s,  
    output y  
);  
    assign  
    y = i[0]&~s[1]&~s[0] |  
        i[1]&~s[1]&s[0] |  
        i[2]&s[1]&~s[0] |  
        i[3]&s[1]&s[0];  
  
endmodule
```

4x1 Multiplexor design in Verilog

DigitalJS Online

Not secure | digitaljs.tilk.eu

12055

Setup I/O

unnamed.sv

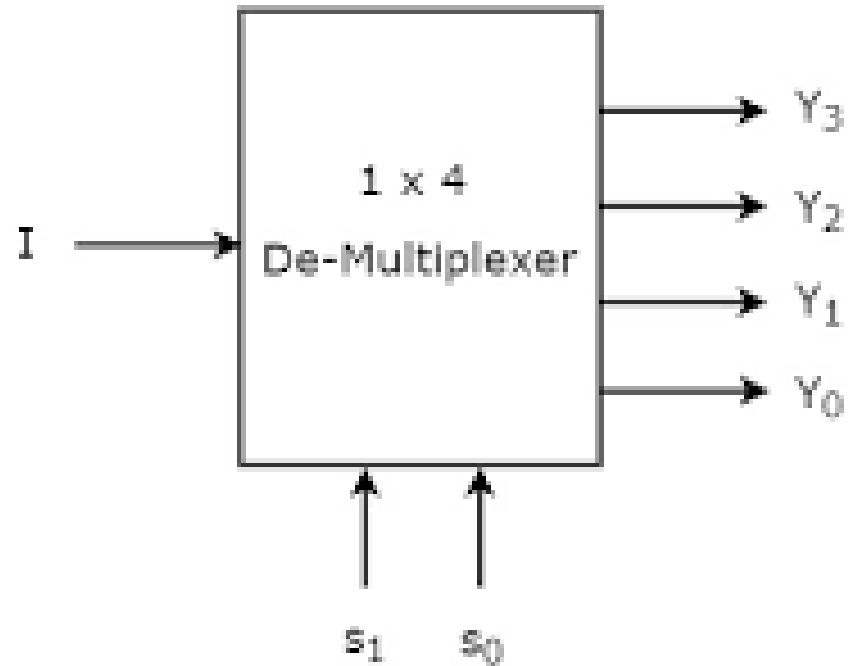
```
1 module MUX4x1(  
2     input [3:0] i,  
3     input [1:0] s,  
4     output y  
5 );  
6  
7 assign  
8 y = i[0]&~s[1]&~s[0] |  
9     i[1]&~s[1]&s[0] |  
10    i[2]&s[1]&~s[0] |  
11    i[3]&s[1]&s[0];  
12  
13 endmodule  
14
```

Synthesize and simulate!

00000000

Construction on Aven... 9:00 AM 5/10/2023

1x4 Demultiplexer



1x4 Demultiplexer design

Input			Output			
I	S_1	S_0	Y_0	Y_1	Y_2	Y_3
I	0	0	I	0	0	0
I	0	1	0	I	0	0
I	1	0	0	0	I	0
I	1	1	0	0	0	I

1x4 Demultiplexer truth table

Input			Output			
I	S ₁	S ₀	Y ₀	Y ₁	Y ₂	Y ₃
0	*	*	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

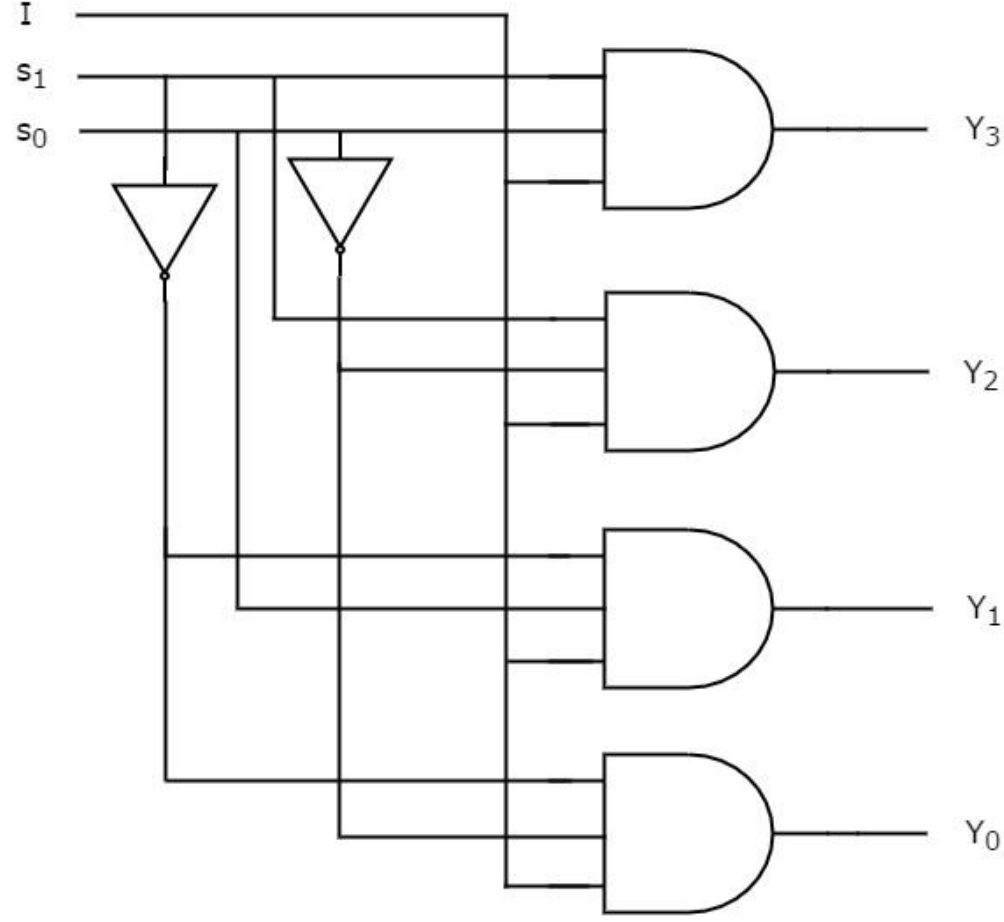
$$Y_0 = I\bar{S}_1\bar{S}_0$$

$$Y_1 = I\bar{S}_1S_0$$

$$Y_2 = IS_1\bar{S}_0$$

$$Y_3 = IS_1S_0$$

1x4 Demultiplexer layout



1x4 Demultiplexer Verilog module

```
module DEMUX1x4(  
    input i,  
    input [1:0] s,  
    output [3:0] y  
);  
  
    assign  
        y[0] = i&~s[1]&~s[0],  
        y[1] = i&~s[1]&s[0],  
        y[2] = i&s[1]&~s[0],  
        y[3] = i&s[1]&s[0];  
  
endmodule
```


1x4 Demultiplexer design in Verilog

DigitalJS Online

Not secure | digitaljs.tilk.eu

26253

Setup I/O unnamed.v

```
1 module DEMUX1x4(  
2     input i,  
3     input [1:0] s,  
4     output [3:0] y  
5 );  
6  
7 assign  
8     y[0] = i&~s[1]&~s[0],  
9     y[1] = i&~s[1]&s[0],  
10    y[2] = i&s[1]&~s[0],  
11    y[3] = i&s[1]&s[0];  
12  
13 endmodule
```

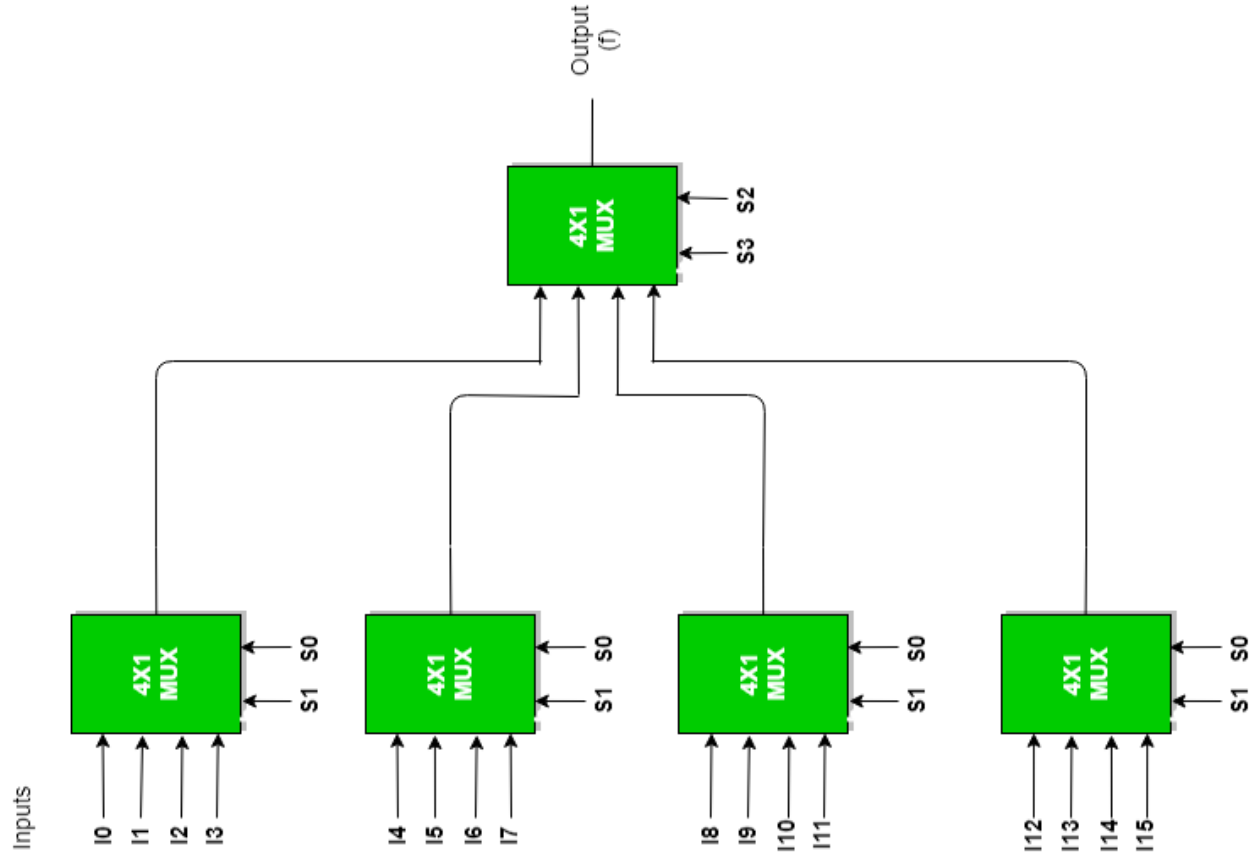
Synthesize and simulate!

The circuit diagram illustrates the implementation of the Verilog code. It features four 2-input AND gates. The first input of all AND gates is the data input 'i'. The second inputs are generated by combining the select inputs 's[1]' and 's[0]' using inverters and AND gates. Specifically, the second inputs are $\sim s[1] \& \sim s[0]$, $\sim s[1] \& s[0]$, $s[1] \& \sim s[0]$, and $s[1] \& s[0]$. The outputs of these AND gates are connected to a 4-bit bus 'y', which is shown with the value 0100.

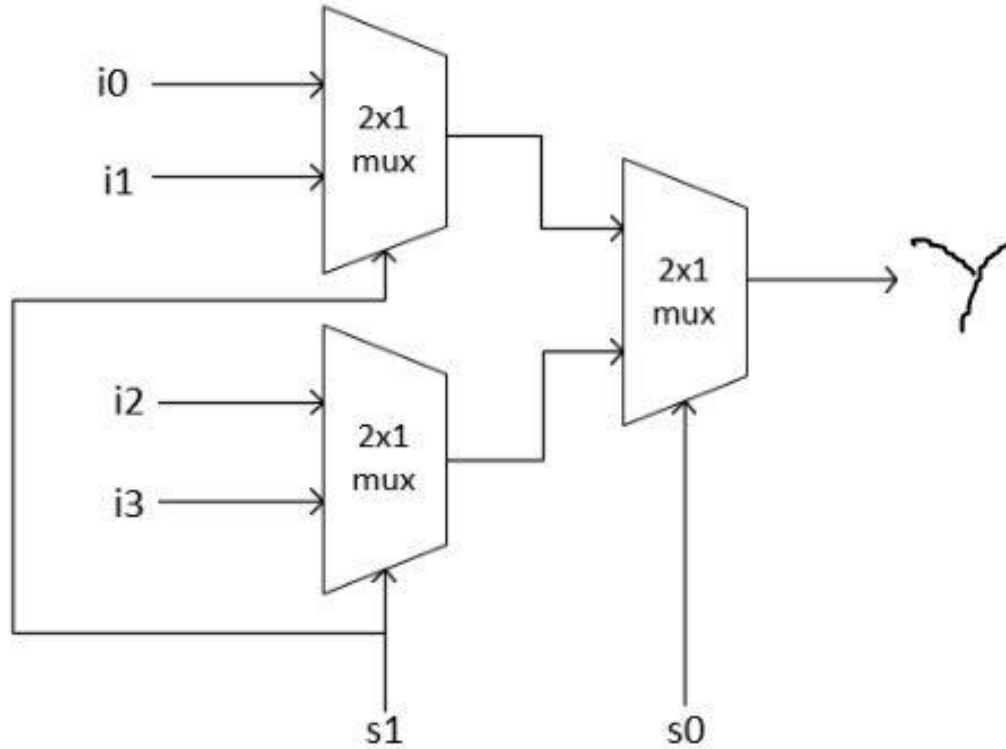
Digital circuits design using multiplexers

- Basic gates in MUX 2x1:
 - NOT: $01(x)$
 - AND: $y1(x)$
 - OR: $1y(x)$
- MUX 4x1 as composition of 2 MUX 2x1 etc
- Adders and other arithmetics

MUX 16x1 as 5 MUX 4x1



MUX4x1 design of 3 MUX2x1



Verilog design of MUX2x1

DigitalJS Online x 16-to-1 multiplexer (16X1 MUX) x +

Not secure | digitaljs.tilk.eu

2377

Setup I/O unnamed.sv

```
1 module MUX_2X1(input I0, I1, S, output Y);
2   wire sbar, w1, w2;
3   not G1 (sbar, S);
4   and G2 (w1, I0, sbar);
5   and G3 (w2, I1, S);
6   or G4 (Y, w1, w2);
7 endmodule
8
9
```

Synthesize and simulate!

The circuit diagram illustrates the implementation of a 2-to-1 multiplexer (MUX2x1). It features three inputs: I0 (green), I1 (red), and S (red). The output is Y (green). The circuit is composed of two AND gates (G2 and G3), one NOT gate (G1), and one OR gate (G4). I0 is connected to the top input of G2. I1 and S are connected to the inputs of G3. S is also connected to the input of G1, which produces sbar. sbar is connected to the bottom input of G2. The outputs of G2 (w1) and G3 (w2) are connected to the inputs of G4, which produces Y.

Verilog MUX4x1 design of 3 MUX2x1

DigitalJS Online x 16-to-1 multiplexer (16X1 MUX) x +

Not secure | digitaljs.tilk.eu

2963

Setup I/O unnamed.sv

```
1 module MUX_2X1(input I0, I1, S, output Y);
2   wire sbar, w1, w2;
3   not G1 (sbar, S);
4   and G2 (w1, I0, sbar);
5   and G3 (w2, I1, S);
6   or G4 (Y, w1, w2);
7 endmodule
8
9 module MUX_4X1(input i0, i1, i2, i3, s0, s1, output Y);
10  wire Y1, Y2;
11  MUX_2X1 M1 (i0, i1, s0, Y1);
12  MUX_2X1 M2 (i2, i3, s0, Y2);
13  MUX_2X1 M3 (Y1, Y2, s1, Y);
14 endmodule
```

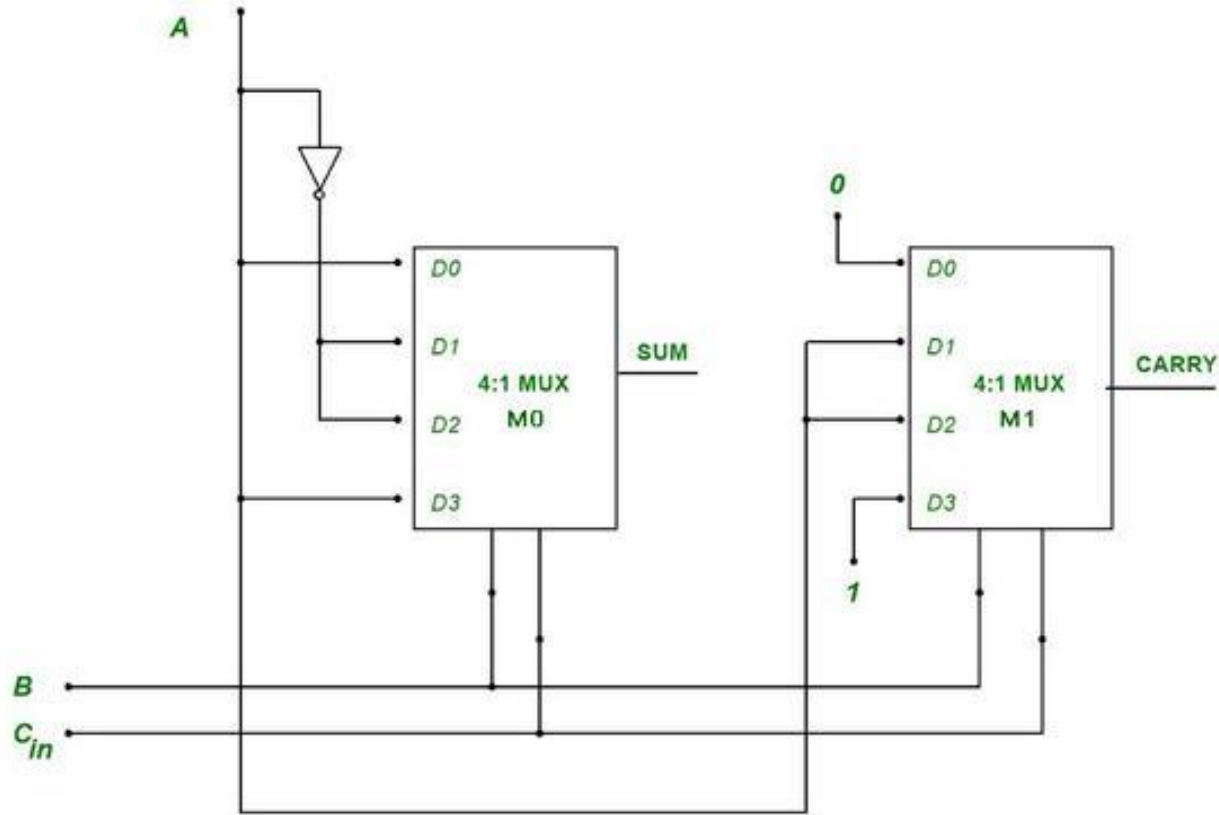
Synthesize and simulate!

The diagram illustrates the implementation of a 4-to-1 multiplexer (MUX_4X1) using three 2-to-1 multiplexers (MUX_2X1). The inputs are i0, i1, i2, i3, s0, and s1. The outputs are Y1, Y2, and Y. The circuit is configured as follows:

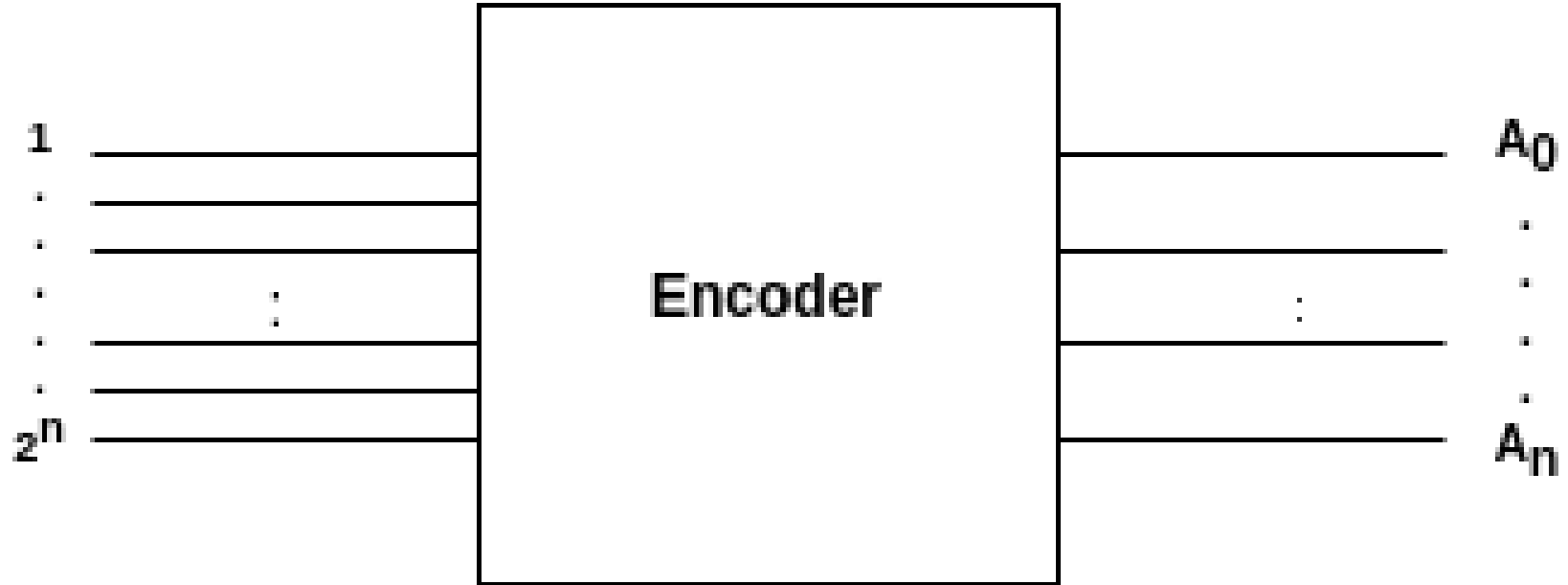
- Two 2-to-1 MUXes (M1 and M2) are connected to inputs i0, i1 and i2, i3 respectively, with select line s0.
- The outputs of M1 and M2 are Y1 and Y2.
- A third 2-to-1 MUX (M3) is connected to inputs Y1 and Y2, with select line s1.
- The output of M3 is Y.

Windows Taskbar: 65°F Mostly cloudy 9:17 AM 5/10/2023

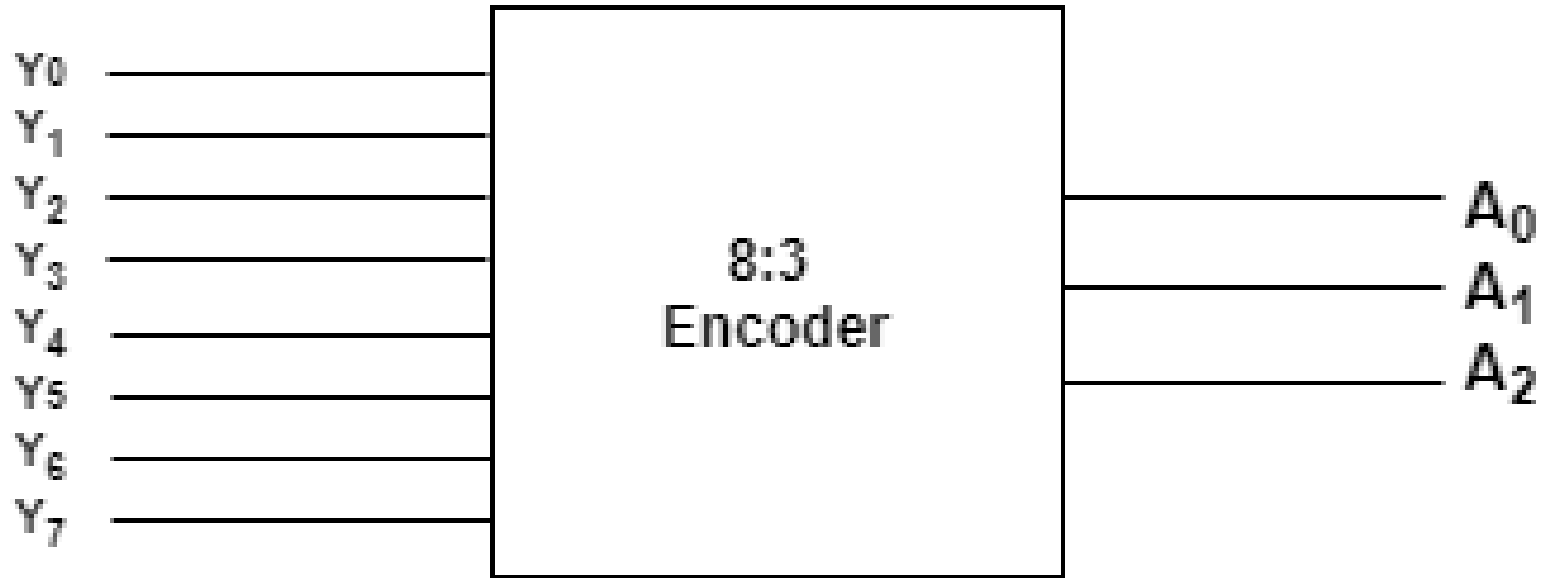
1 bit full adder on MUX4x1



Encoder



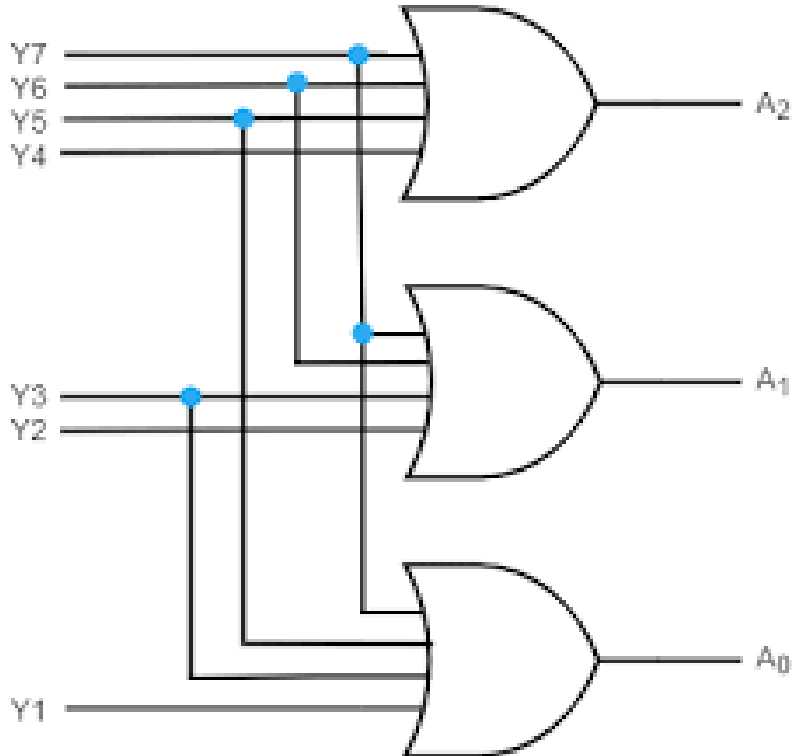
8x3 Encoder



8x3 Encoder truth table

[illegible]

8x3 Encoder layout



$$A2 = Y4 \vee Y5 \vee Y6 \vee Y7$$

$$A1 = Y2 \vee Y3 \vee Y6 \vee Y7$$

$$A0 = Y1 \vee Y3 \vee Y5 \vee Y7$$

Ambiguity of encoder

- only one input can be active at any given time;
Priority Encoder – assigned priority of input lines
- all inputs are 0 – output is 000 the same as for Y0; extra output – the valid bit: 0 when all inputs 0 and 1 otherwise

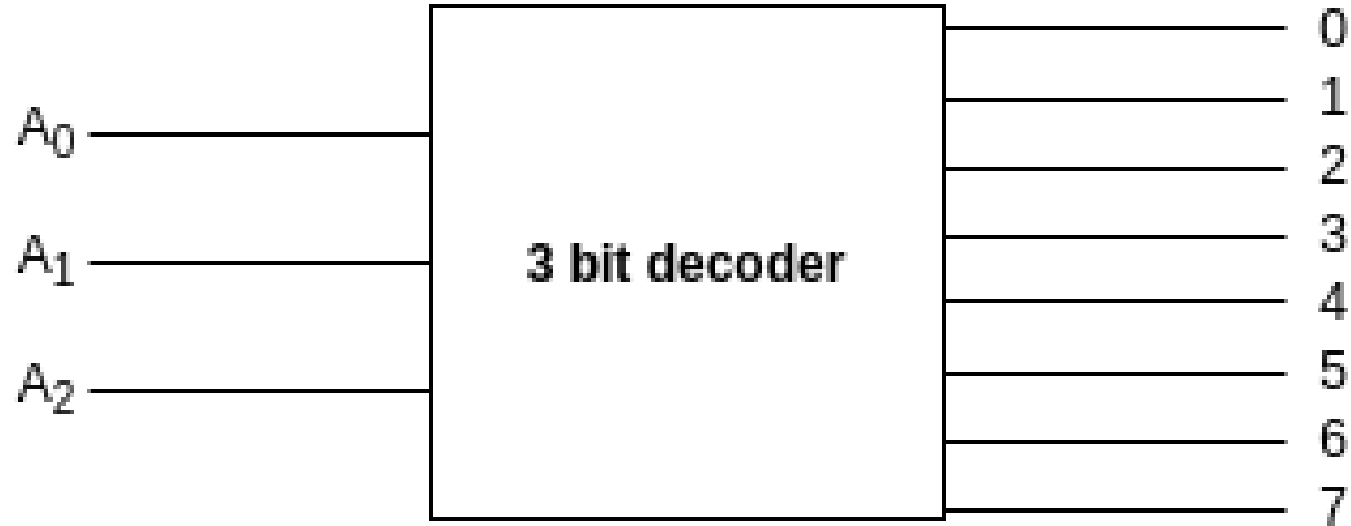
4x2 Priority Encoder with Valid Bit

X3	X2	X1	X0	A1	A0	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Decoder



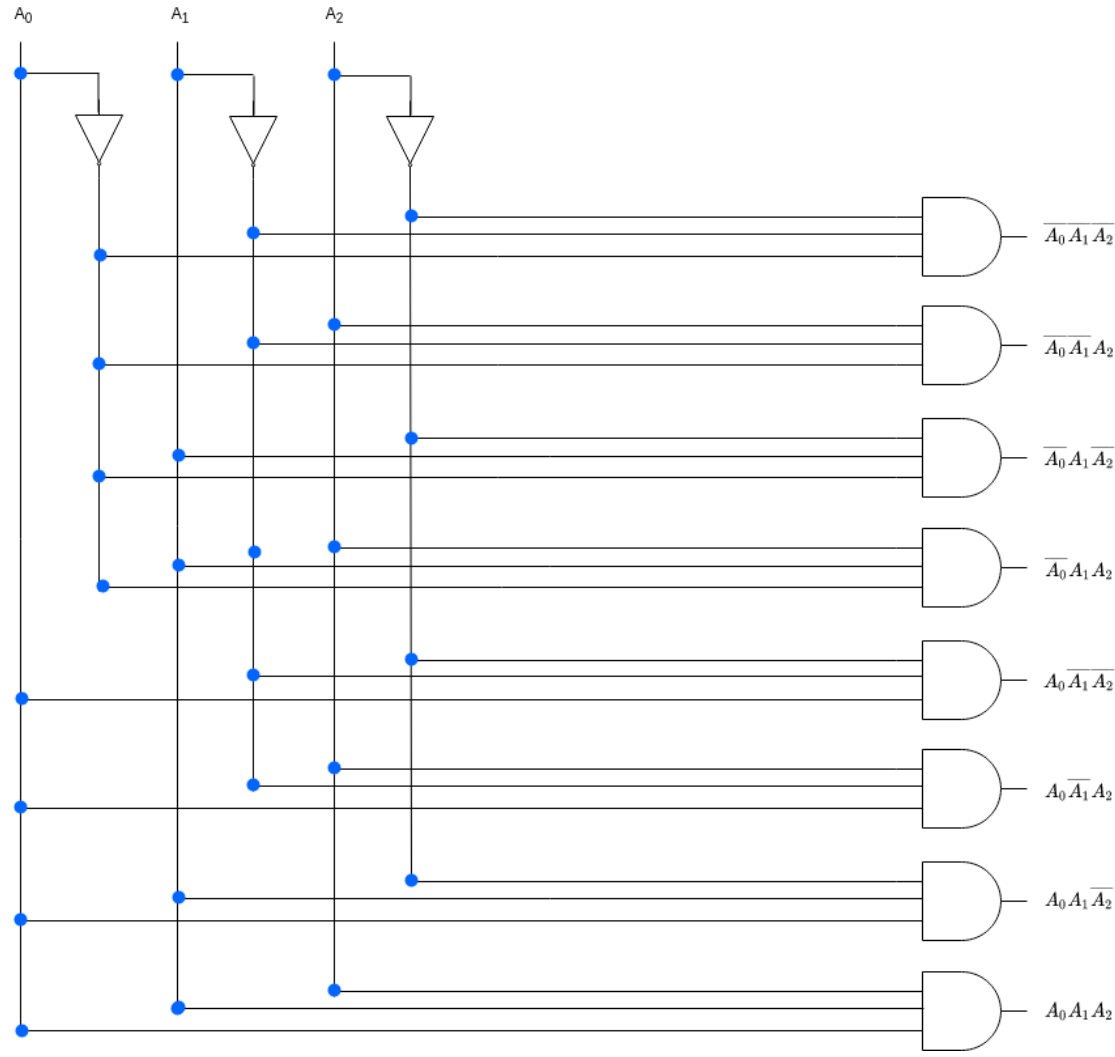
3x8 Decoder



3x8 Decoder truth table

Decimal Digit	Binary			Output							
	A0	A1	A2	0	1	2	3	4	5	6	7
0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0	0	0	0
3	0	1	1	0	0	0	1	0	0	0	0
4	1	0	0	0	0	0	0	1	0	0	0
5	1	0	1	0	0	0	0	0	1	0	0
6	1	1	0	0	0	0	0	0	0	1	0
7	1	1	1	0	0	0	0	0	0	0	1

3x8 Decoder layout



Full adder using 3x8 decoder

