VE270 RC Week 5
Combinational Circuit

Shi Li

2019.10.9

Combinational Circuit

- The output depends only upon the present combination of its inputs
- Input change $\Leftarrow\Rightarrow$ Output change

Combinational Circuit compared with Sequential Circuit

- Truth table for a combinational circuit

p	q	p∨q
T	T	T
T	F	T
F	T	T
F	F	F

- Characteristic table for a sequential circuit

S(t)	R(t)	Q(t)	$Q(t+\Delta)$ → Q^+	
0	0	0	0	hold
0	0	1	1	
0	1	0	0	reset
0	1	1	0	
1	0	0	1	set
1	0	1	1	
1	1	0	X	not allowed
1	1	1	X	

Design Process

- Capture the function
  - truth table/equation from requirements
- Convert to equation
  - k-map logic optimization
- Implement the circuit
  - from the optimized logic expression

Design Process
Exercise 1 (modified from 2018 Fall RC Slides)

- Now imagine that you are the smart tech support for a spy, you have to design a lock for his secret suitcase:
  - Each input is a 4-bit binary number $x_1x_2x_3x_4$;
  - 1010 opens the suitcase;
  - 0100, 0101, 1100, 1101, 1111 blows the suitcase up (doesn’t matter whether the suitcase opens or not);
  - Output x for whether to open the suitcase and y for whether to make it explode

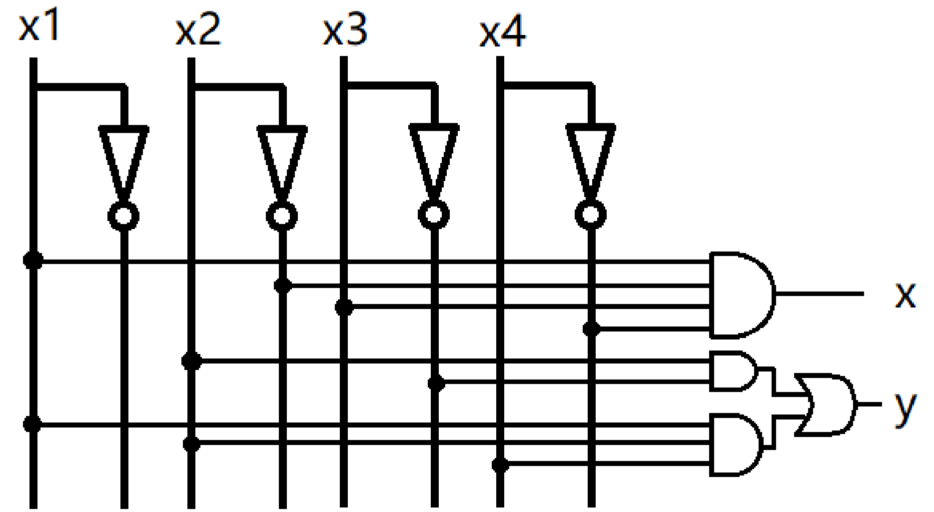
Design Process
Exercise 1 (modified from 2018 Fall RC Slides)

- Step 1. Capture the function
  - $x = x_1 x_2' x_3 x_4'$
  - $y = x_1' x_2 x_3' x_4' + x_1' x_2 x_3' x_4 + x_1 x_2 x_3' x_4' + x_1 x_2 x_3' x_4 + x_1 x_2 x_3 x_4$
- Step 2. Convert the equation
  - We use k-map to simplify the expression for y.
  - $y = x_2 x_3' + x_1 x_2 x_4$

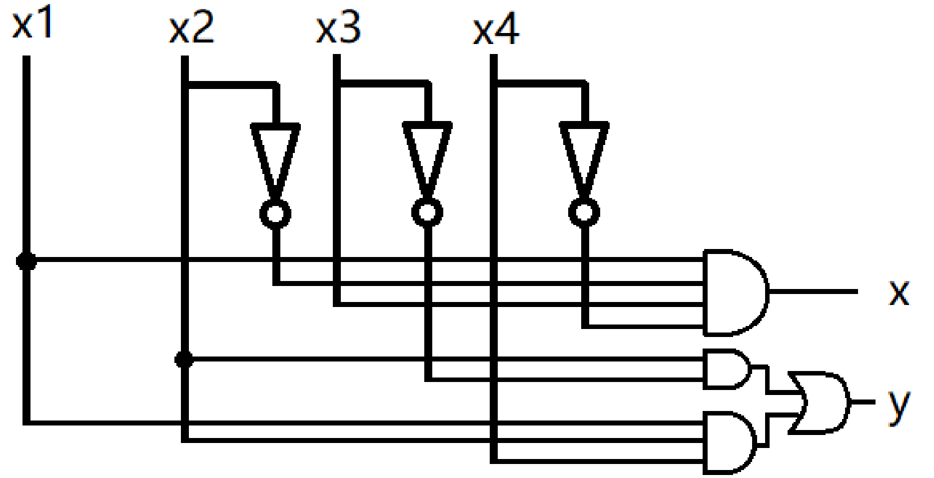
x3x4 \ x1x2	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	0	1	0
10	0	0	0	0

Design Process
Exercise 1 (modified from 2018 Fall RC Slides)

- Step 3. Implement the circuit

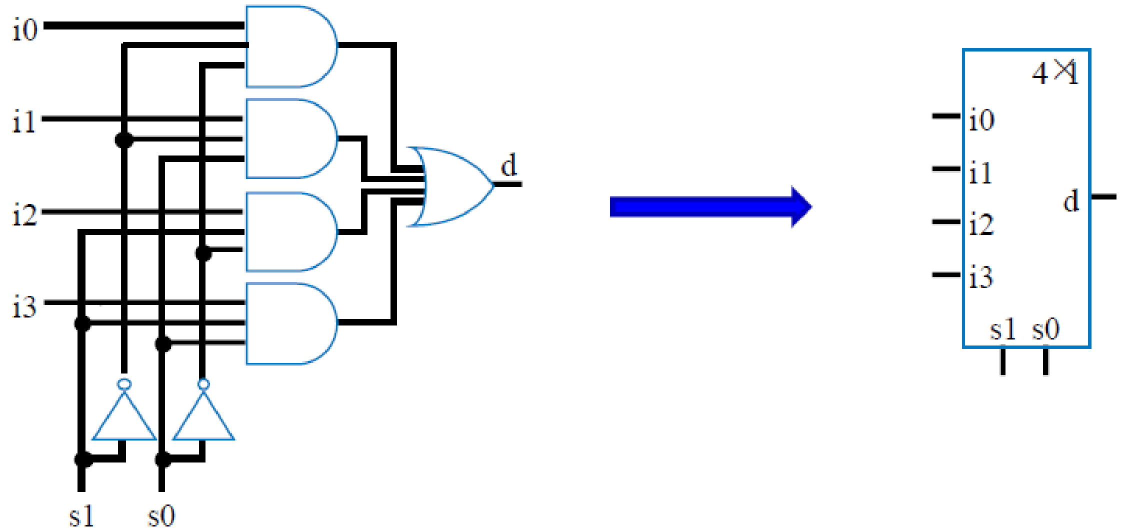


- We can further simplify it.



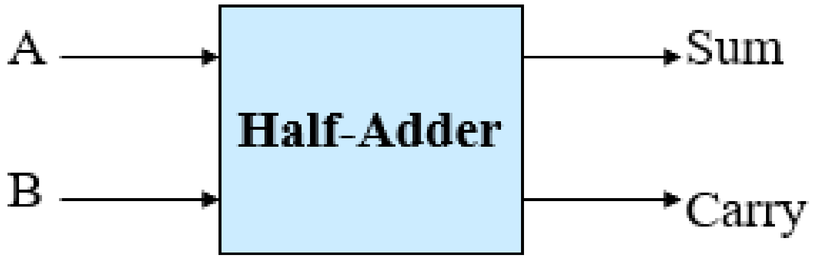
Combinational Building Blocks
1. MUX

- Example: 4 to 1 Mux
- Using 2 “switches” to choose one of the four input signals

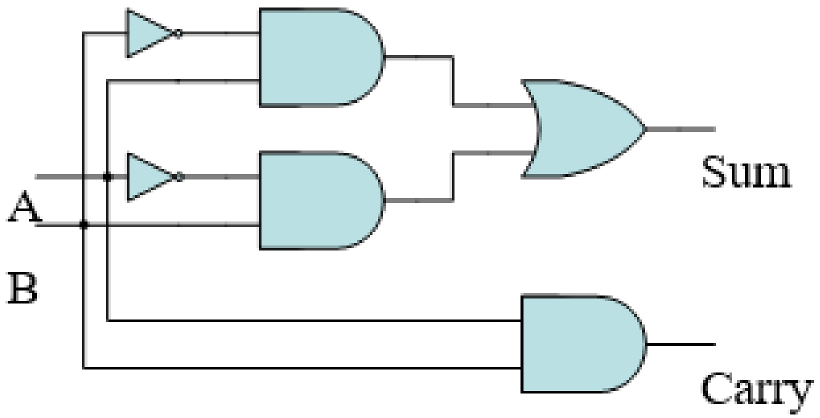


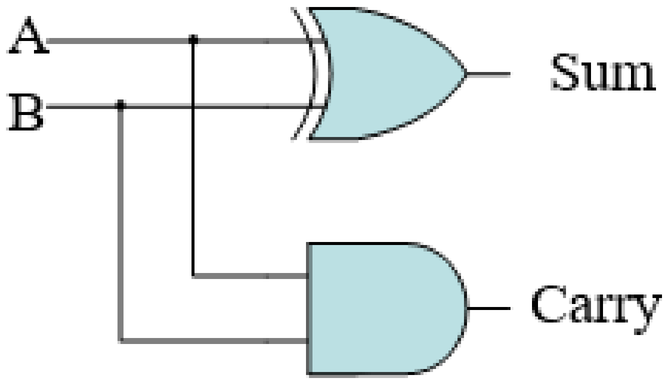
- Question: If we have 35 input from i0 to i34, how much “switches” do we need?
- Answer: 6 (i35~i63 can be seen as don’t cares)

Combinational Building Blocks
2. Adder: Half Adder



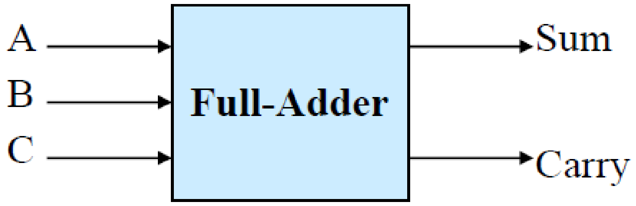
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1





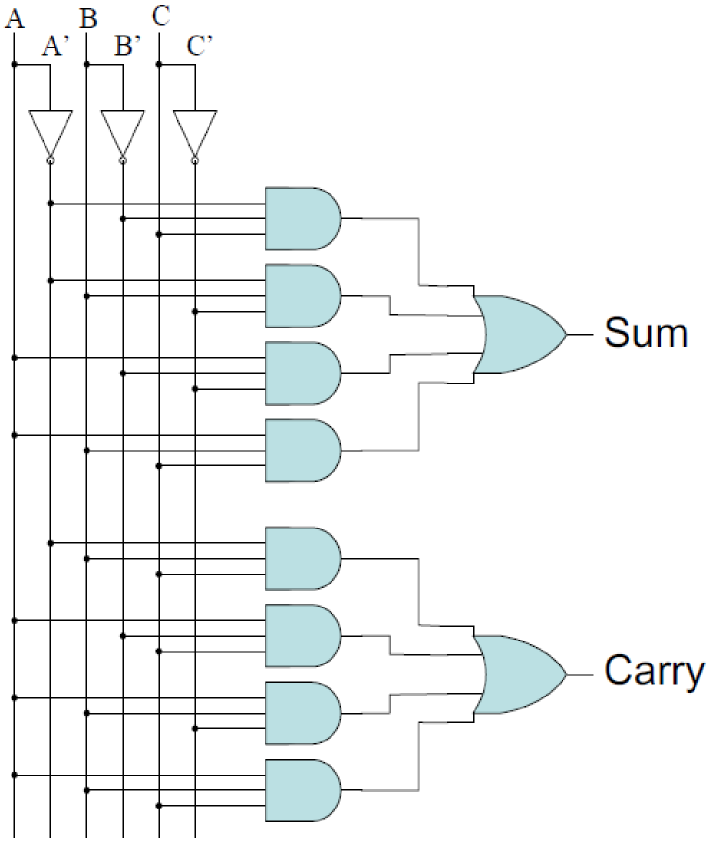
Combinational Building Blocks
2. Adder: Full Adder

- The first way to implement a full adder.



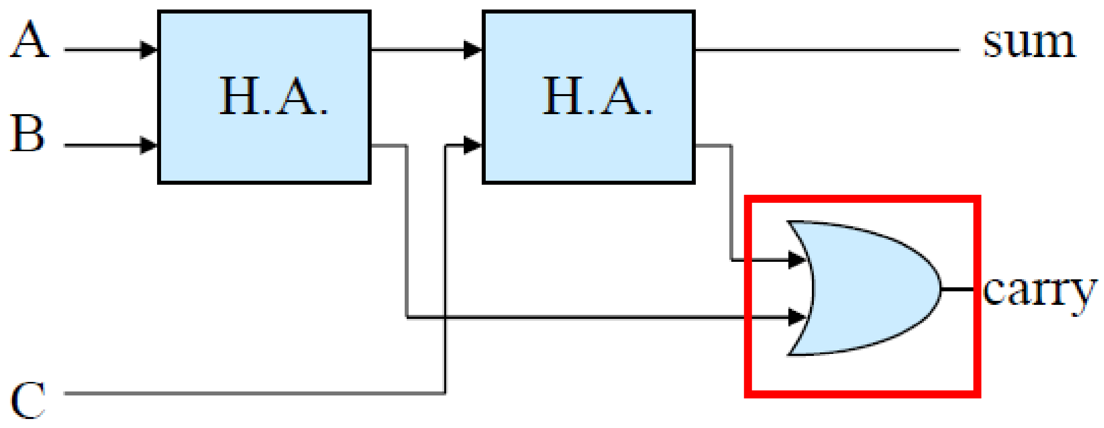
A	B	C	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\text{Sum} = A'B'C + A'BC' + AB'C' + ABC$$
$$= \Sigma\, m(1, 2, 4, 7)$$
$$\text{Carry} = A'BC + AB'C + ABC' + ABC$$
$$= \Sigma\, m(3, 5, 6, 7)$$



Combinational Building Blocks
2. Adder: Full Adder

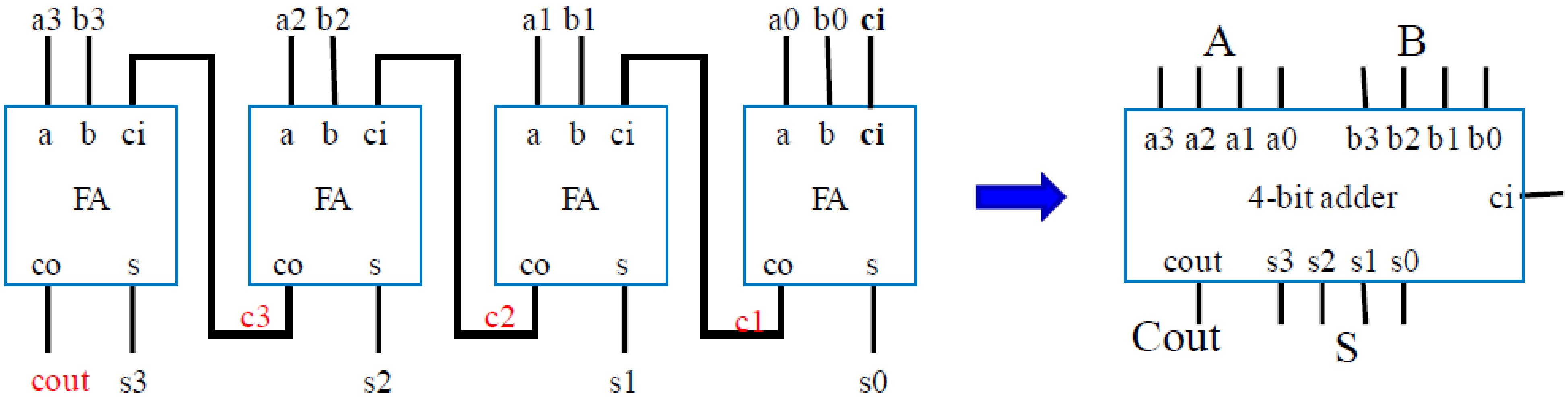
- Is there any way to make it simpler?



Question: Why use OR gate here?
Answer: Because the two H.A. cannot generate 1 at the same time. An OR gate is enough to deal with all the cases.

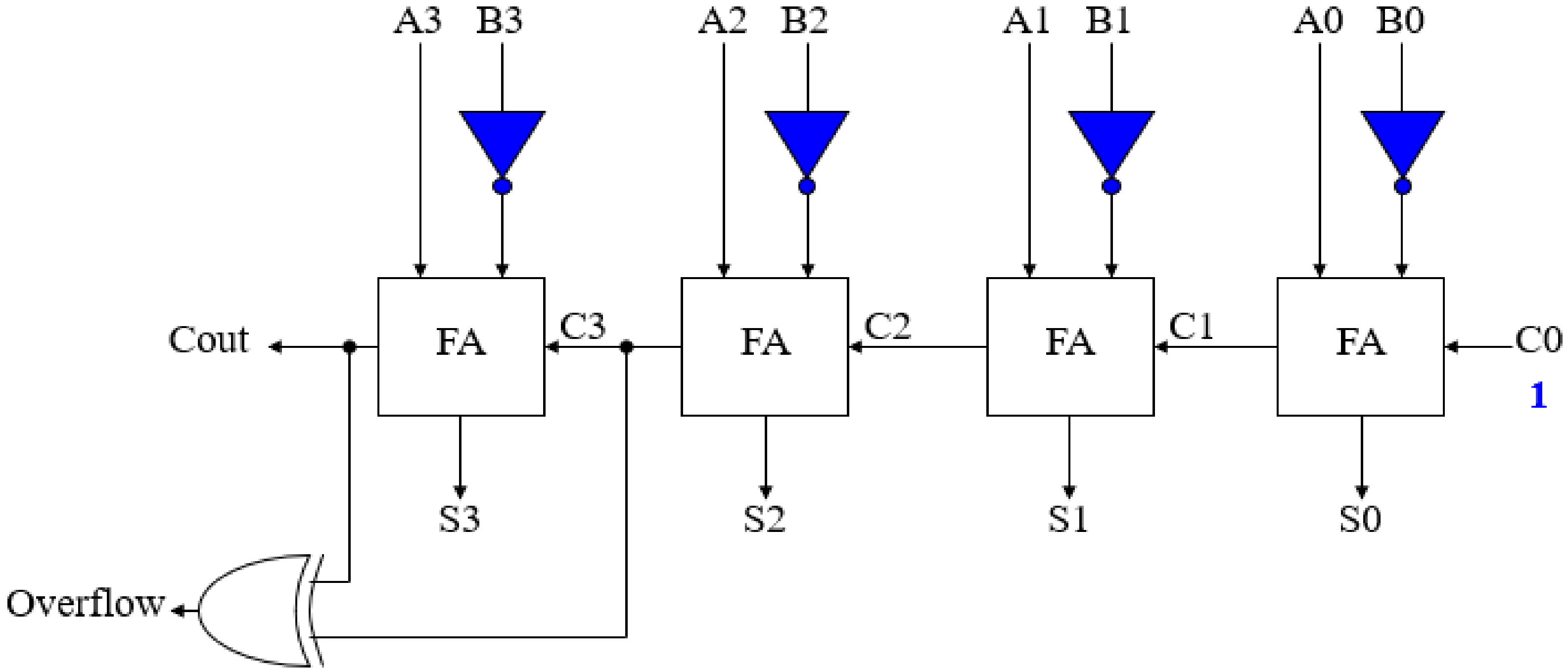
Combinational Building Blocks
2. Adder: Carry-Ripple Adder

- For a carry-ripple adder with 4-bit input, it generates 5-bit output.



Combinational Building Blocks
2. Adder: Subtractor (for 2's complement numbers)

- When we do subtraction, we need to
- 1. add inverters to B input
- 2. set C0 to 1

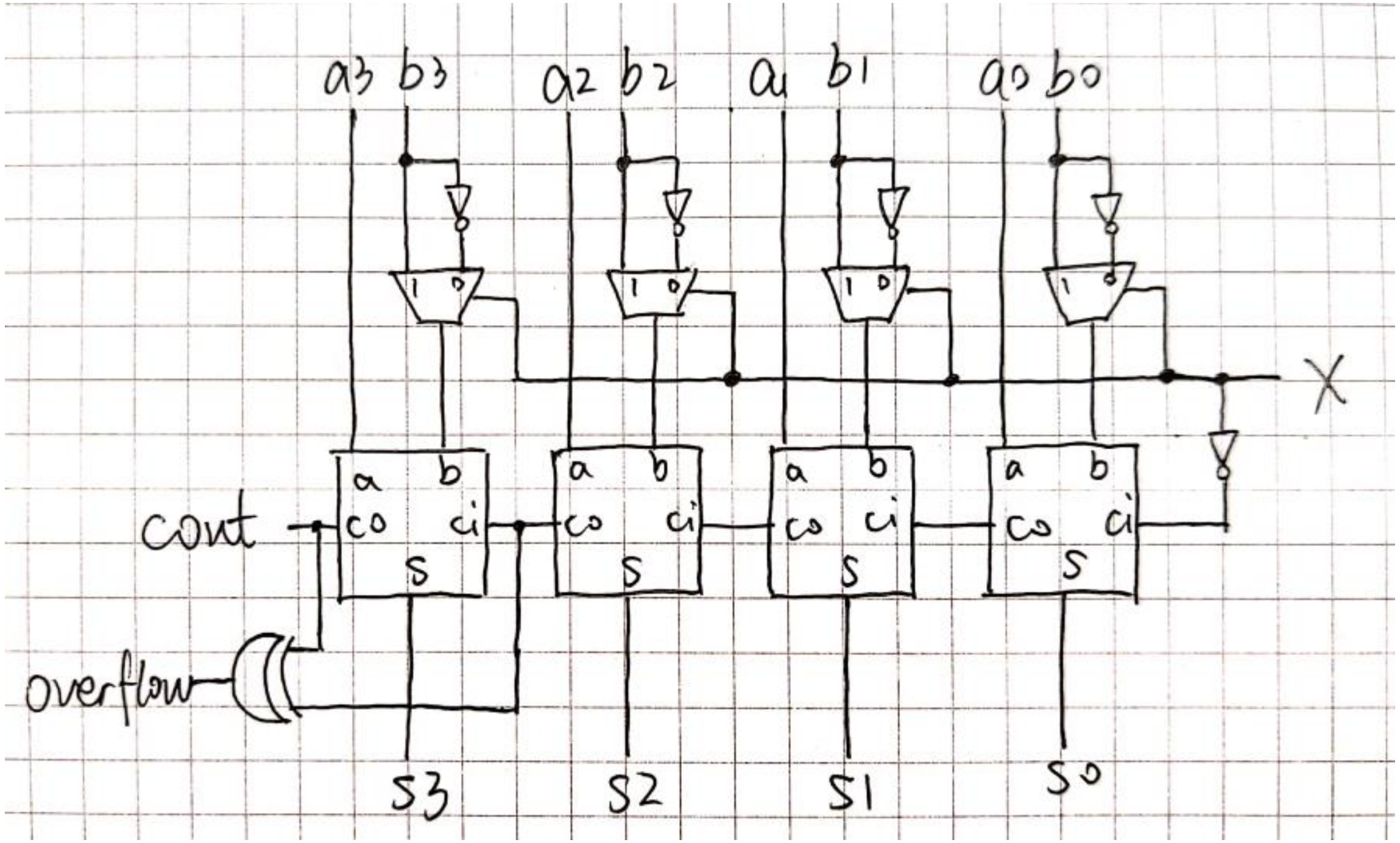


Combinational Building Blocks
Exercise 2

- Use 2-1 muxes to design a simple 4-bit Arithmetic-Logic Unit.
- The input is ~~Cin~~, A0~A3, B0~B3, X. (My mistake, cin is not needed)
- The output is Cout, Overflow, S0~S3.

- Specifications: When X=1, S=A+B. When X=0, S=A-B.
- You do not need to draw the complete circuit. Use blocks (mux, full adder, etc) instead.

Combinational Building Blocks
Exercise 2



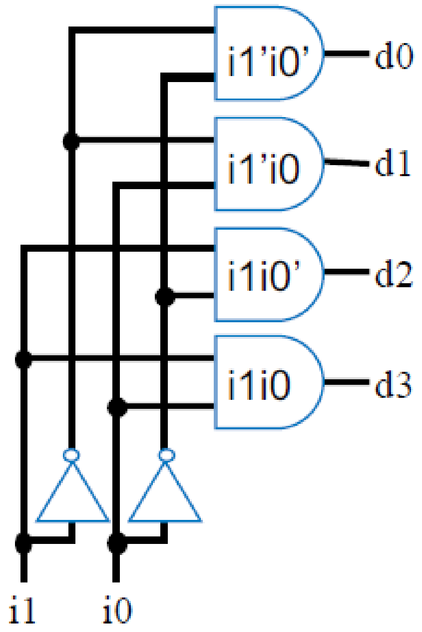
Combinational Building Blocks
3. Encoder & Decoder

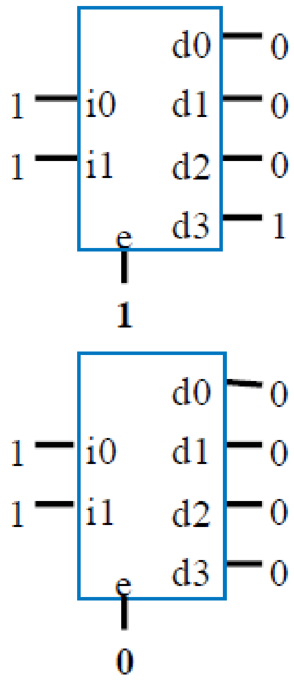
Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Combinational Building Blocks
3. Encoder & Decoder

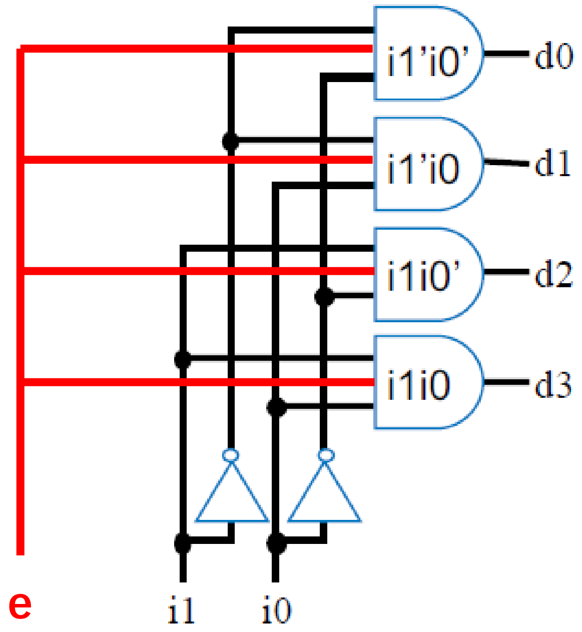
- Decoder: N inputs, 2^N outputs
- Enable e (Question: How to implement it in the circuit?)
- Use decoder to implement any combinational circuit

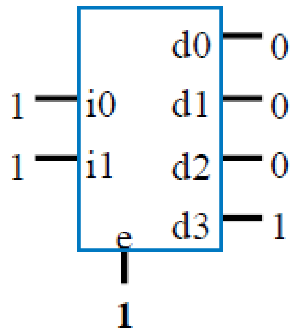


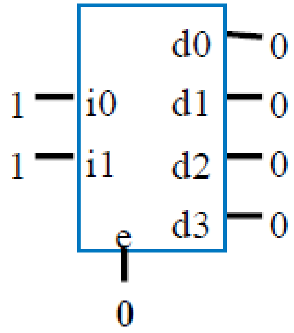


Combinational Building Blocks
3. Encoder & Decoder

- Decoder: N inputs, 2^N outputs
- Enable e (Question: How to implement it in the circuit? – Red lines)
- Use decoder to implement any combinational circuit

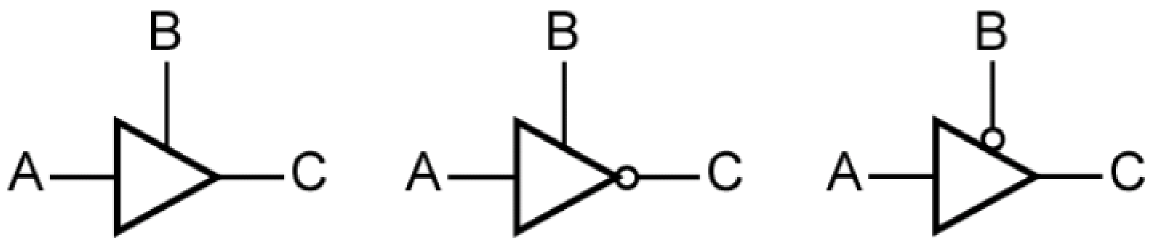






Combinational Building Blocks
4. Buffer & Tri-state Buffer

- Why we use buffers?
  - Amplify the driving capability of a signal
  - Insert delay
  - Protect input from output
- Why we use tri-state buffer?
  - Provide another state “Z”
  - Z: high impedence



B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1

B	A	C
0	0	Z
0	1	Z
1	0	1
1	1	0

B	A	C
0	0	0
0	1	1
1	0	Z
1	1	Z