Digital Logic

CS6133 - Computer Architecture I

Vikram Padman

Polytechnic Institute of New York University

vikram@poly.edu

Agenda

1. **Recap**
2. **Reading List**
3. **Where Are We Going?**
4. **Digital Logic Circuits**
5. **Combinatorial Building Blocks**
6. **Timing Analysis**
7. **Synchronous Building Blocks**
8. **Combinatorial vs Synchronous Logic**
9. **Activity**

Recap

- Hierarchical and Layered design
- Binary Numbers, Addition and Subtraction
- Logic Gates
- Digital Discipline

Reading List

- "Digital Design and Computer Architecture", Chapter 2, chapter 3 section 3.1-3.3 and 3.5, chapter 5 section 5.1 and 5.2

Where Are We Going?

- Why are we looking at digital electronics?

Where Are We Going?

- Why are we looking at digital electronics?
  1. Charles Babbage built a mechanical computer.

Where Are We Going?

- Why are we looking at digital electronics?
  1. Charles Babbage built a mechanical computer.
  2. John Von Newman and A.M.Turing theorized an electronic Digital computer, which we use today.

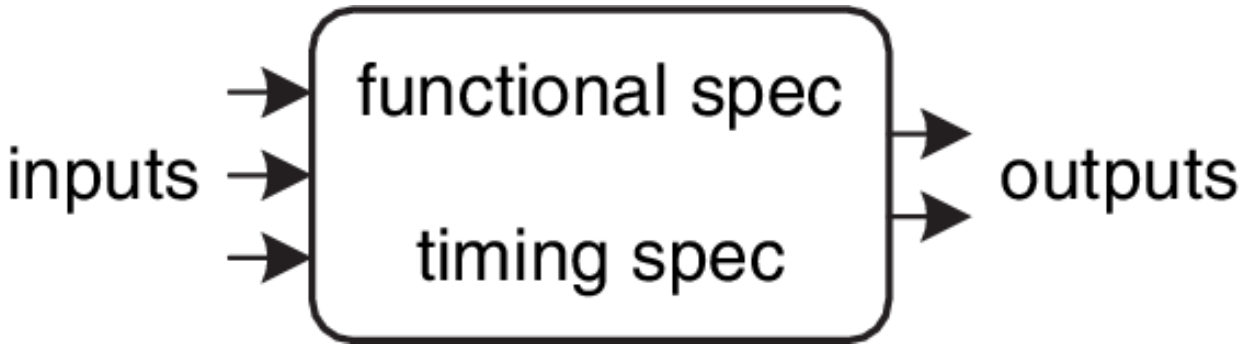
Where Are We Going?

- Why are we looking at digital electronics?
  1. Charles Babbage built a mechanical computer.
  2. John Von Newman and A.M.Turing theorized an electronic Digital computer, which we use today.
- We are here to understand how a digital computers work and we will do that by:
  - Understanding the fundamental of digital logic circuits
  - Building a small CPU with a handful of instructions

Digital Logic Circuits

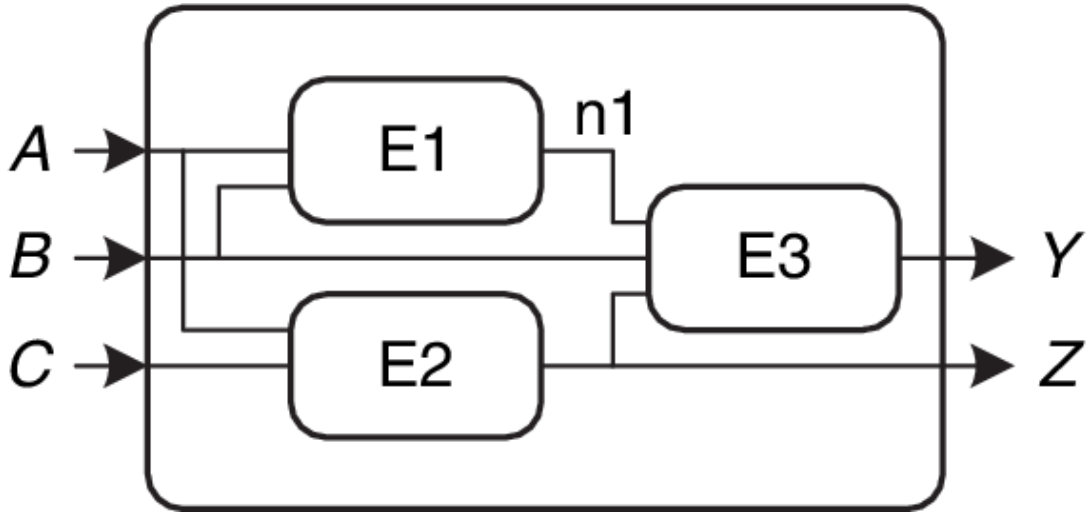
A digital logic circuit is composed of

1. Functional Specifications
   1. Inputs
   2. Outputs
2. Timing Specifications



Digital Logic Circuits

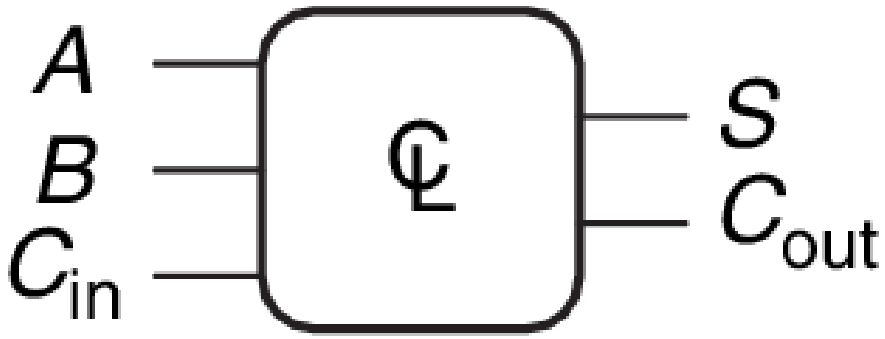
1. Functional and Timing Specifications
2. Nodes
   - Input Ports: A, B, C
   - Output Ports: Y, Z
   - Internal Wires: n1
3. Circuit Elements or Components
   - E1, E2, E3
   - Internal elements are digital circuits by themselves



- **Combinatorial Circuits**
  - Memoryless
  - Only current input values determine outputs
- **Synchronous or Sequential Circuits**
  - Has memory
  - Current and previous input values determine future outputs

Combinatorial Components/Elements

- Boolean equations are used to specify outputs in terms of inputs
- Example:

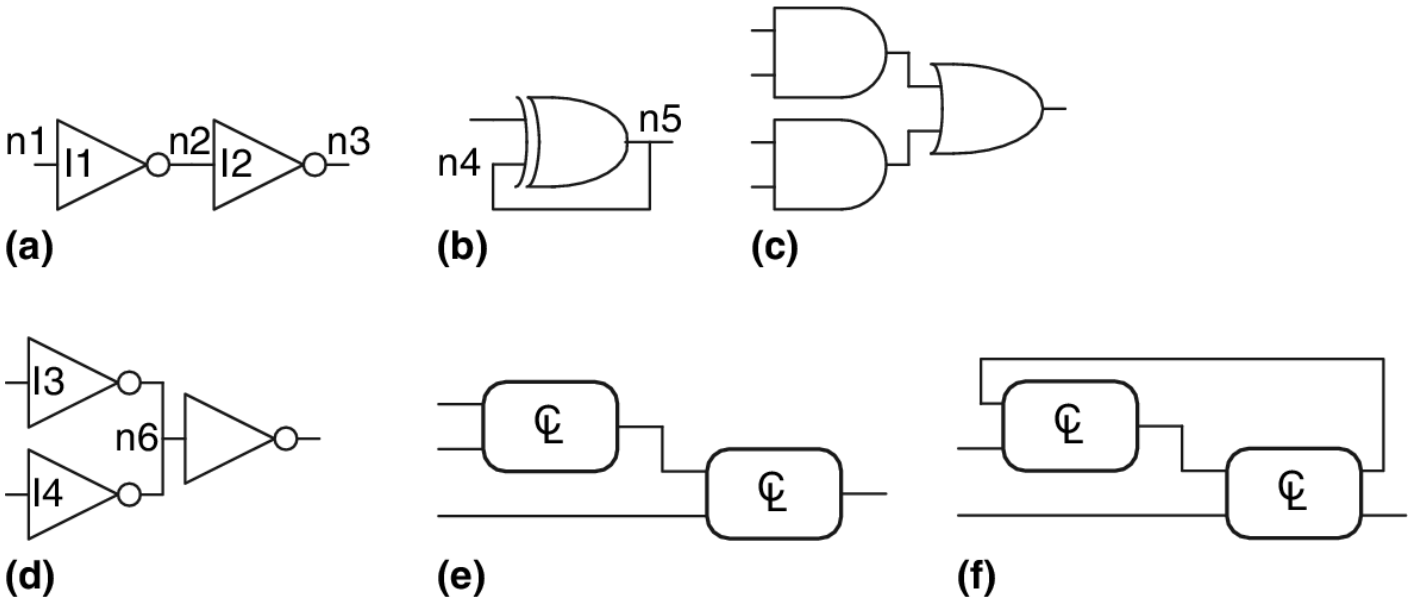


$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AB + AC_{in} + BC_{in}$$

Rules of Combinatorial Components/Elements

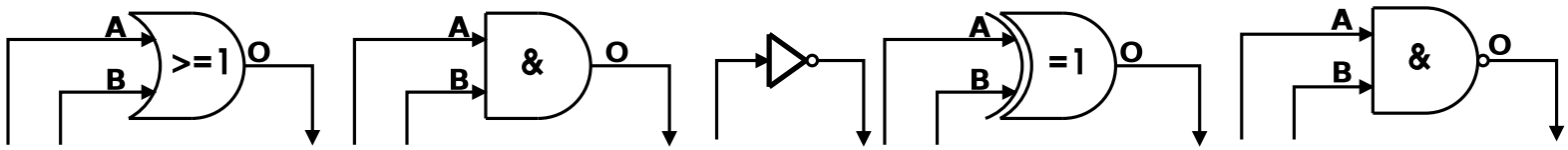
- Every component within a combinatorial element is also combinatorial
- Every internal node (or wire) is an input and connects to exactly one output
- The component does not contain cyclic paths



Find the culprit

Combinatorial Building Blocks

Logic Gates

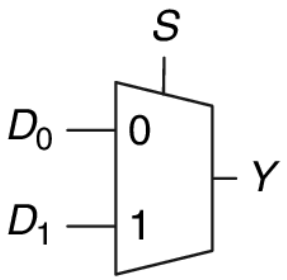


- Fundamental building blocks that are used in all digital electronic devices.
- Logic gates are used to build general purpose components (such as Adder / Subtractors, MUX / DEMUX ...etc)
  - General purpose components are then used to build larger, function specific, components such as arithmetic and logic unit, control unit and various type of memories.
  - Many function specific components are assembled together to make a microprocessor, I/O HUB, SRAM ... etc
- **Completeness Theorem:** NAND or NOR gate could be used to build any boolean function.

Combinatorial Building Blocks

Multiplexer

- Select between one of **N** inputs to connect to output
- A control signal of size $\log_2(N)$ bit(s) is used for selection

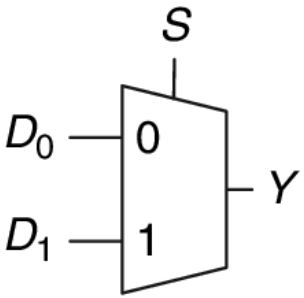


S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

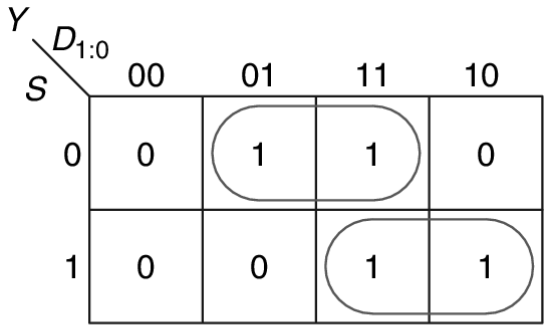
Combinatorial Building Blocks

Multiplexer

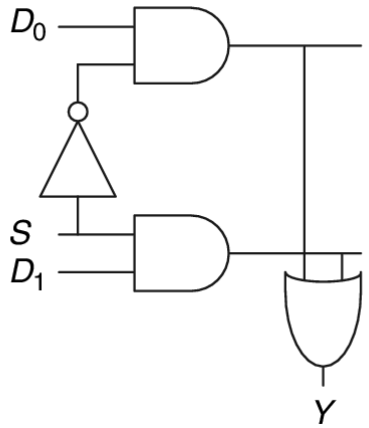
Using Logic Gates



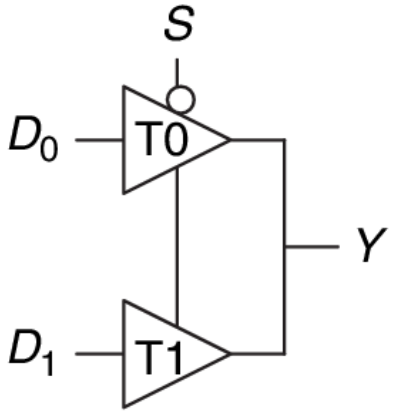
S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



$$Y = D_0\bar{S} + D_1S$$



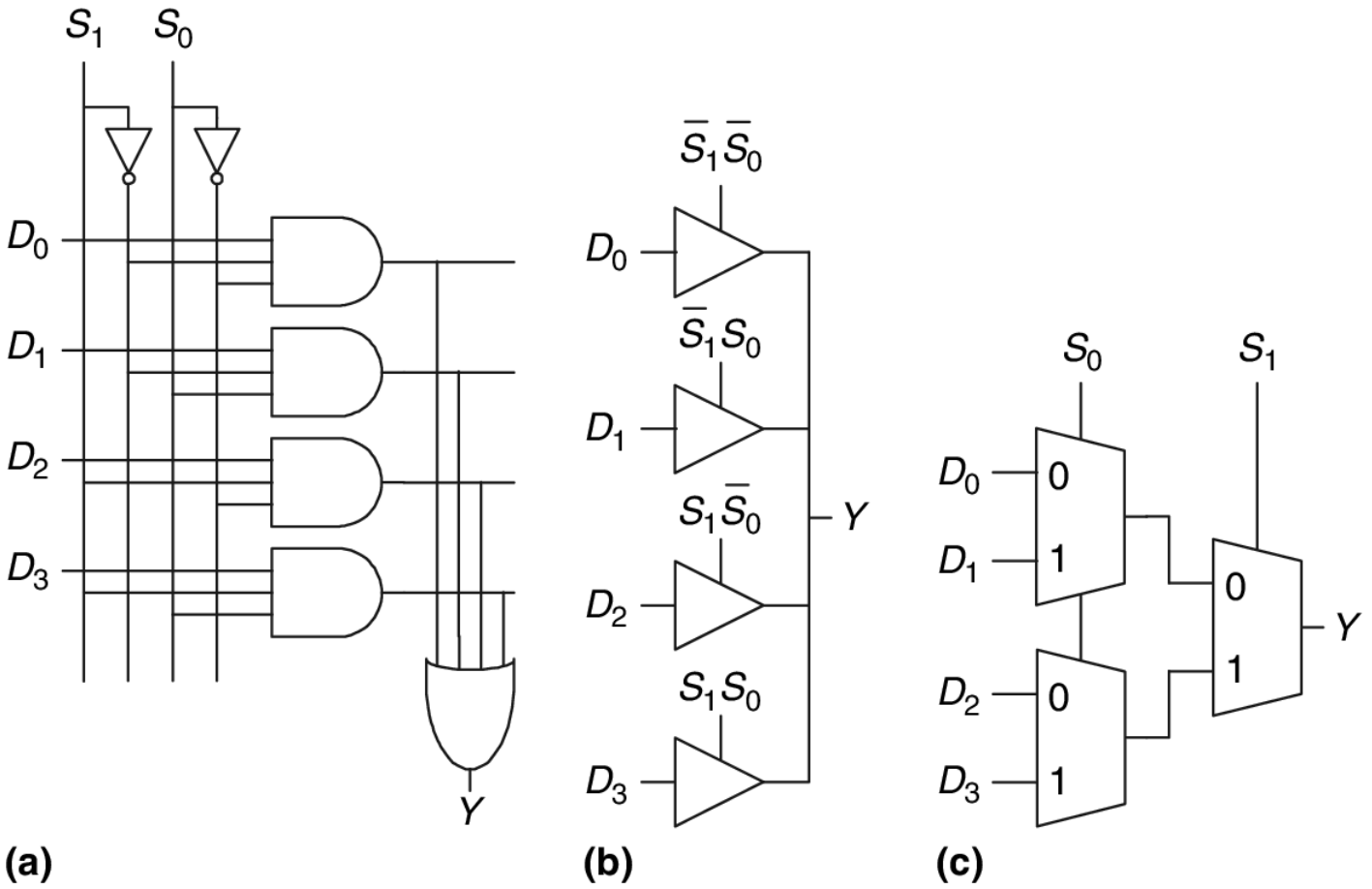
Using Tristates



$$Y = D_0\bar{S} + D_1S$$

Combinatorial Building Blocks

Multiplexer - 4 to 1 MUX



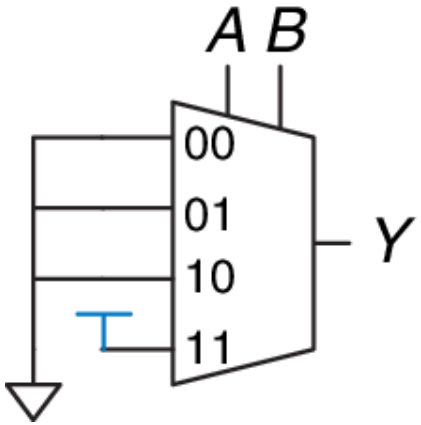
Combinatorial Building Blocks

Multiplexer - Look Up Table

- Using the MUX as a Look Up Table (LUT) [1]

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

$Y = AB$

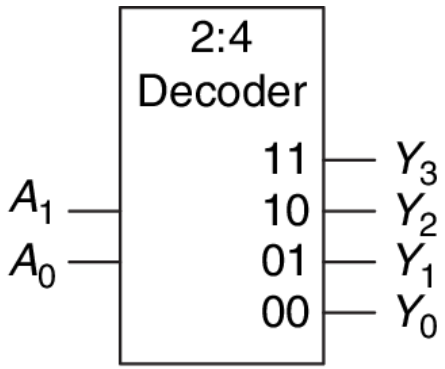


[1] Fundamental block in a FPGA

Combinatorial Building Blocks

De-Multiplexer or Decoders

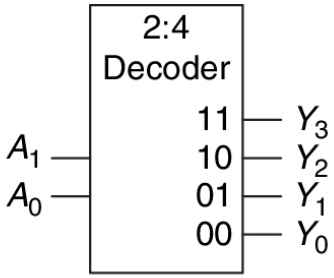
- **N** inputs to 2^N outputs
- One-hot outputs: One one output is "Active" at any given time



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

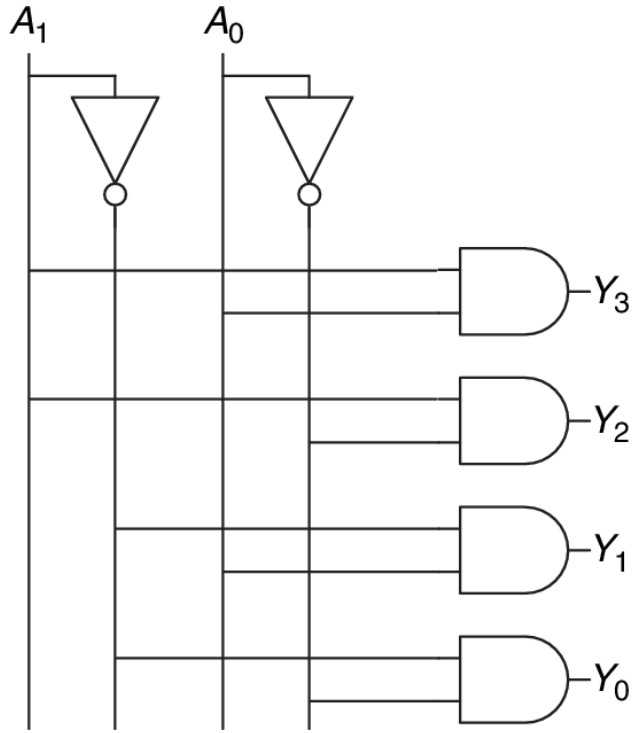
Combinatorial Building Blocks

De-Multiplexer or Decoders



A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

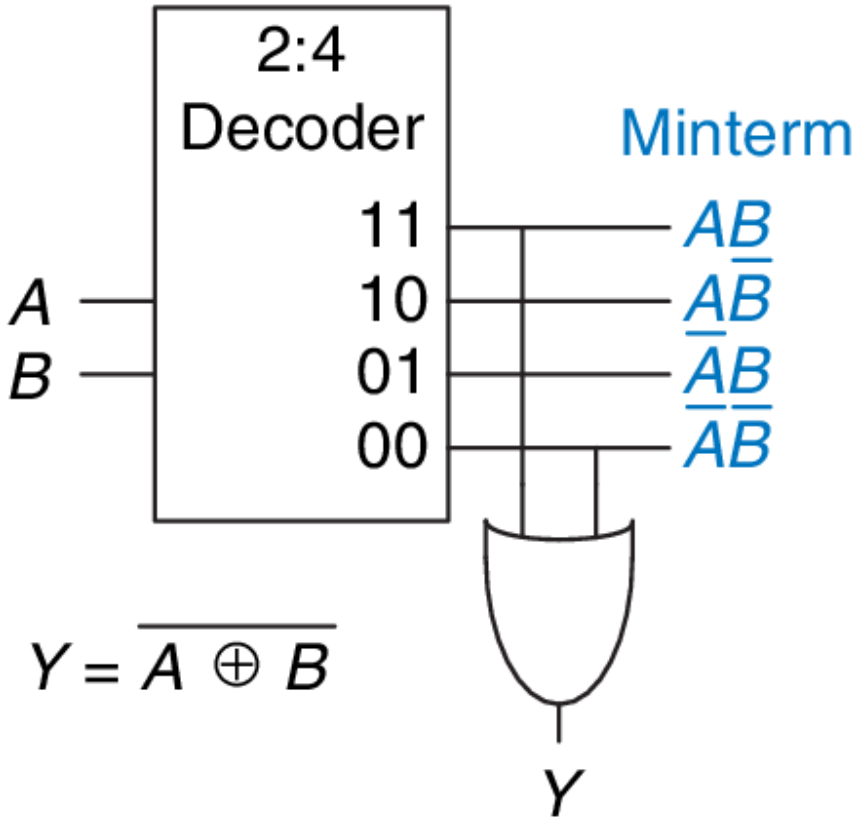
Using Logic Gates



Combinatorial Building Blocks

De-Multiplexer or Decoders

- Logic Function using De-MUX/Decoder



- Which gate is being implemented here?

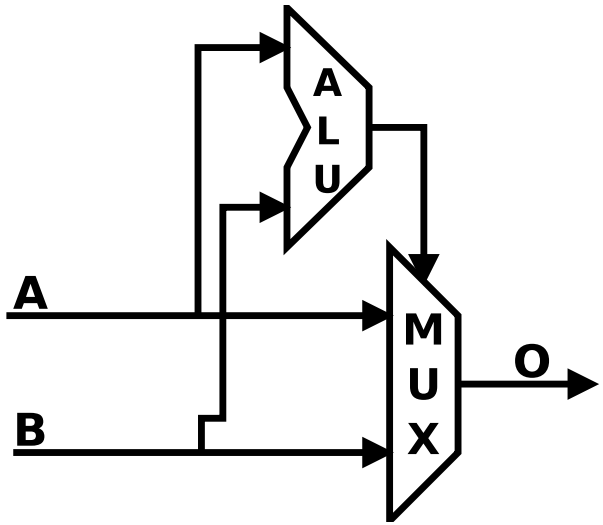
MUX/DEMUX

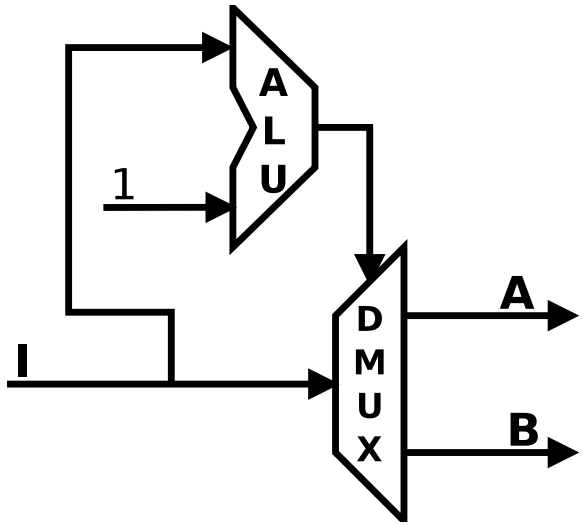
Lets consider the following ternary statements:

O = (A >B) ? A : B

A = (I >1) ? I : 0

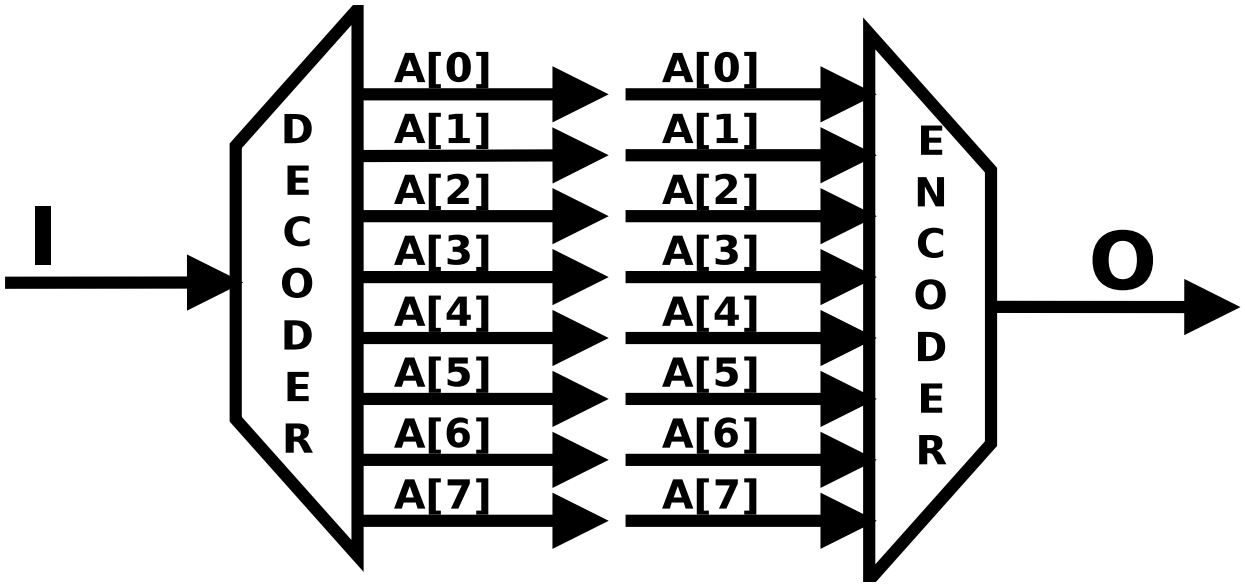
B = (I <1) ? I : 0



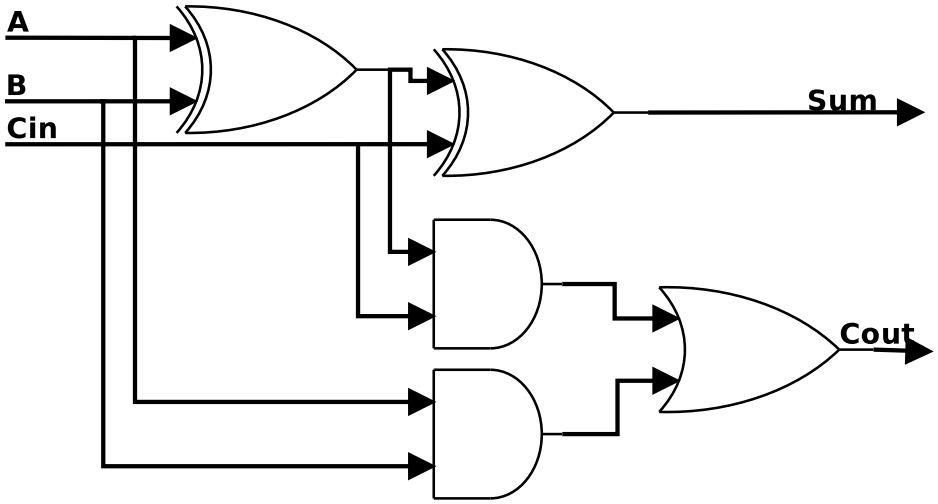


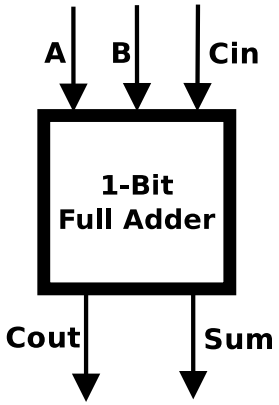
Encoders / Decoders

```
--+-------------------------------------------------------------------------------+
1 | char A[8]; char O; char I; int i;                                             |
2 | I = getc() - 48;              \\ get user input valid input are numbers 0 to 7 |
3 | memset(A,0,sizeof(char)*8); \\ Initialize array A                             |
4 | A[I] = 1;                     \\ Set I'th position to 1                       |
5 | for(i=0;i<8;i++){             \\Decode array A                                |
6 |   if(A[i]==1){                                                                |
7 |    O = i;                                                                     |
8 |    break;                                                                     |
9 |    }                                                                          |
10|}                                                                              |
--+-------------------------------------------------------------------------------+
```



Adder

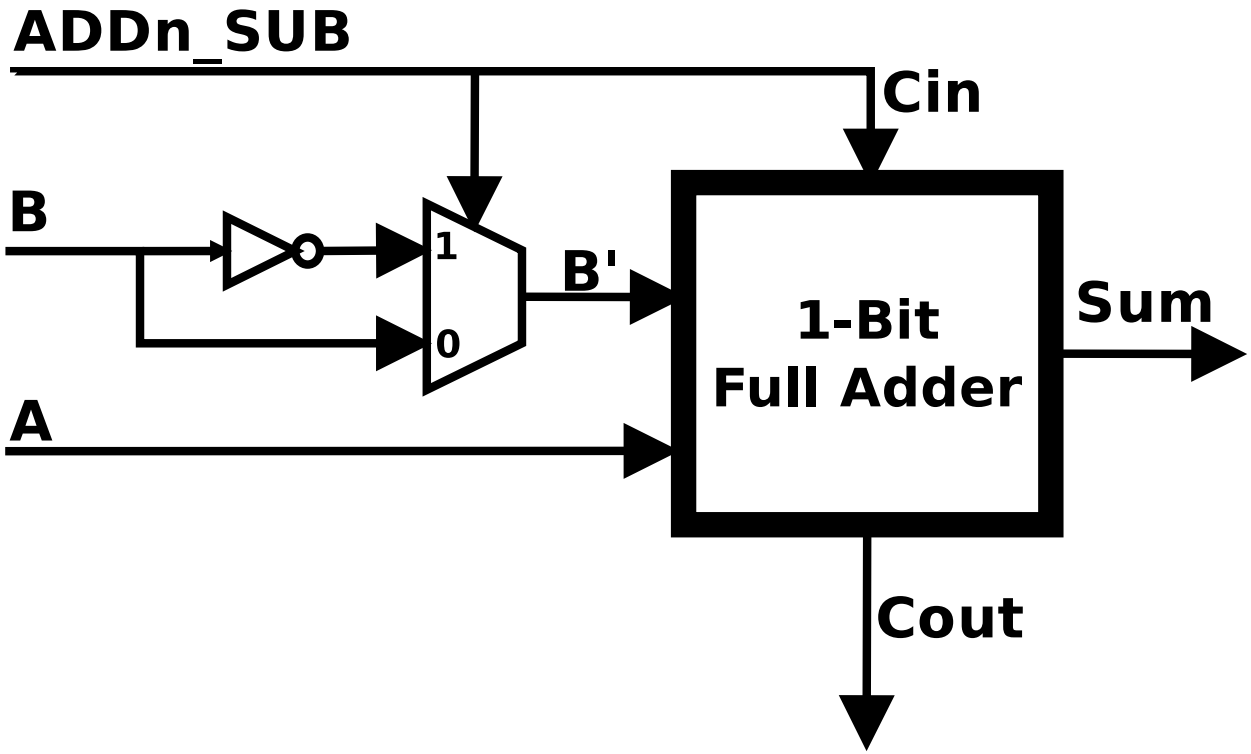




Inputs			Outputs	
Cin	A	B	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

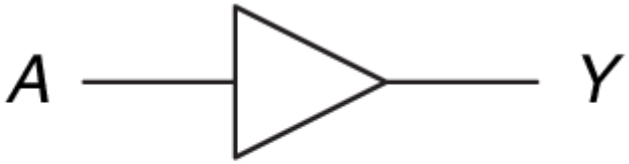
Subtractor

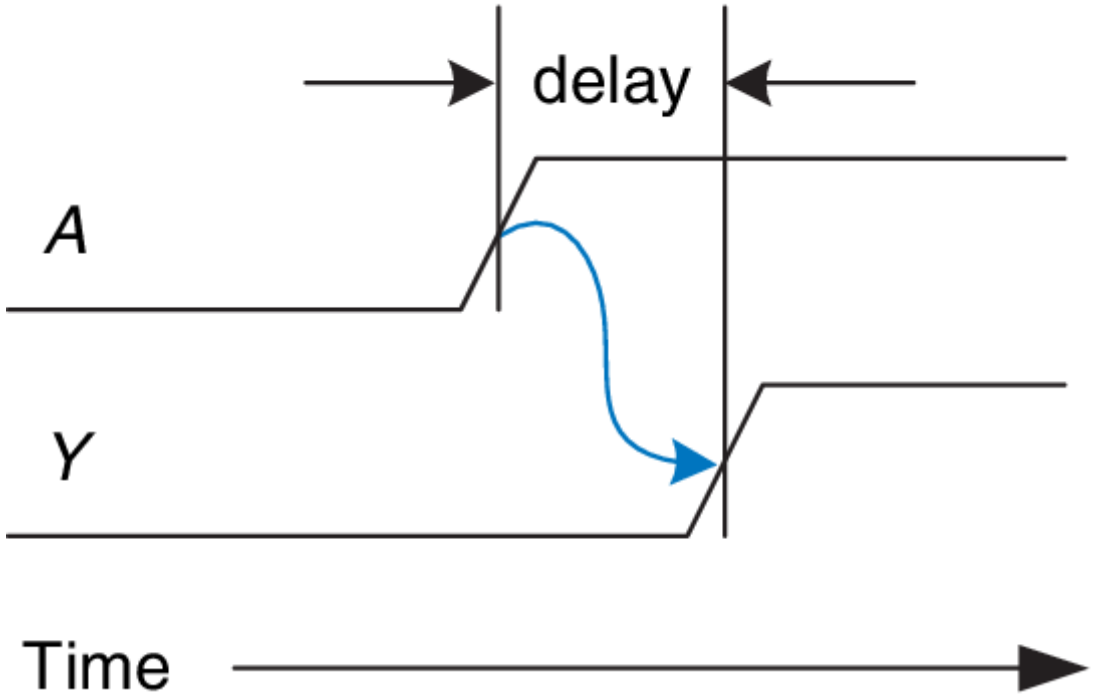
- A full adder could be used for subtraction by converting the negative number into a 2's complement number.
- For example lets say we want to perform $sum = A - B$ we could convert this into $sum = A + (notB + 1)$



Timing Analysis

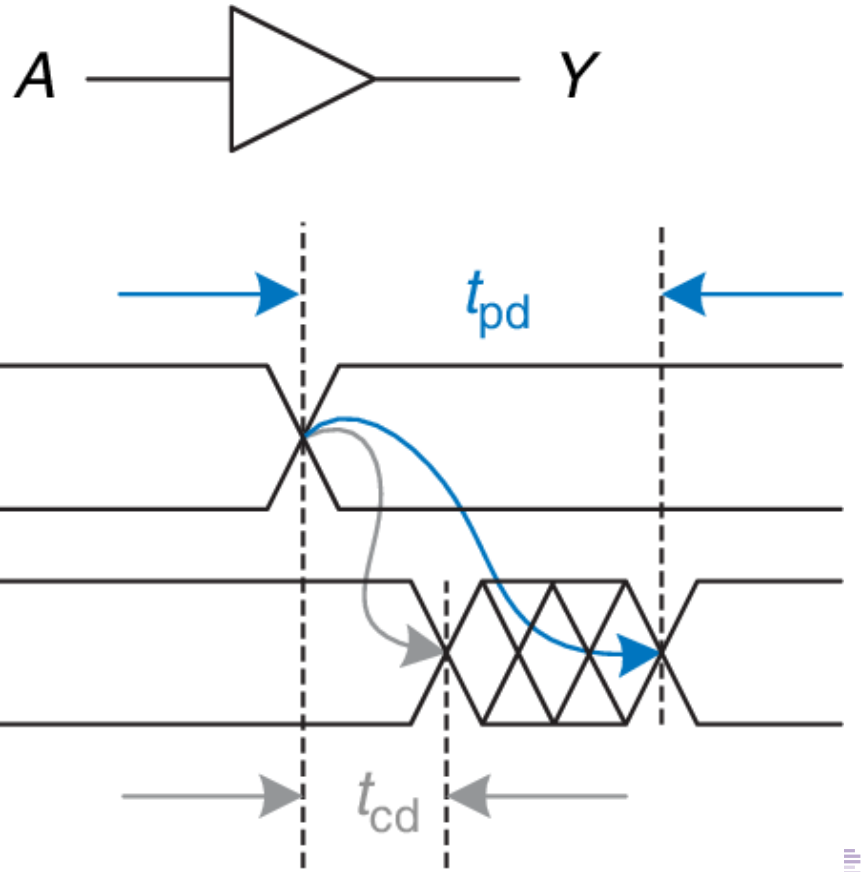
- Propagation delay between input and output





Timing Analysis

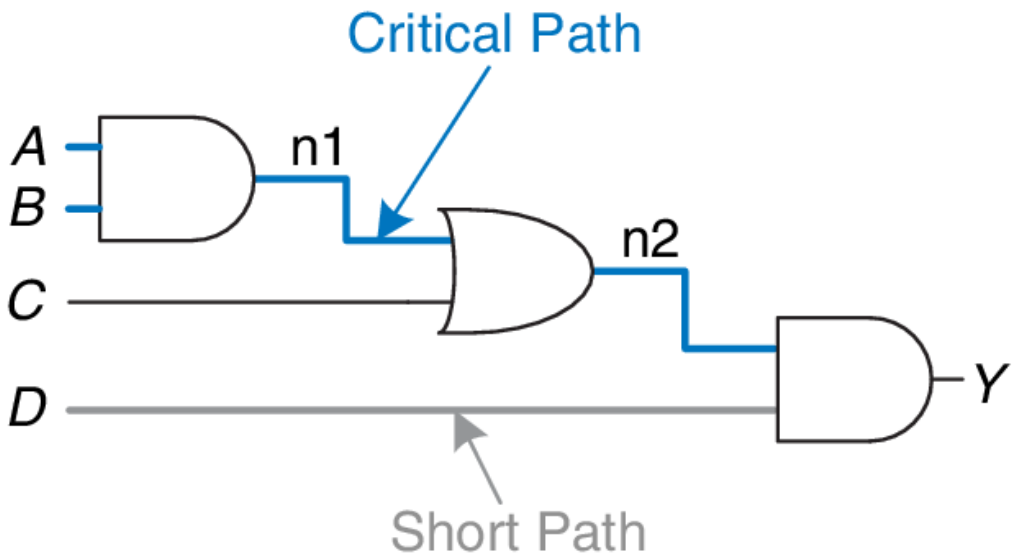
- **Propagation Delay**: t_{pd} is the max delay from input to output
- **Contamination Delay**: t_{cd} us the min dalay from input to output

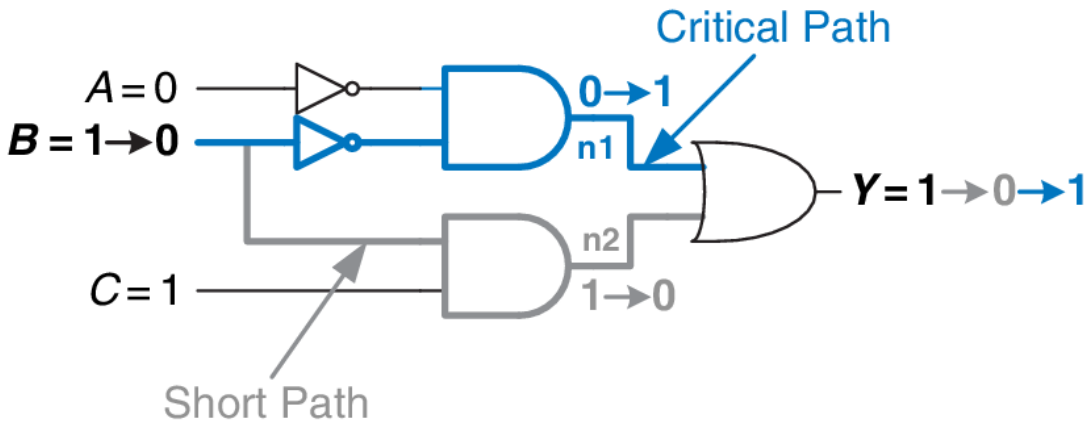


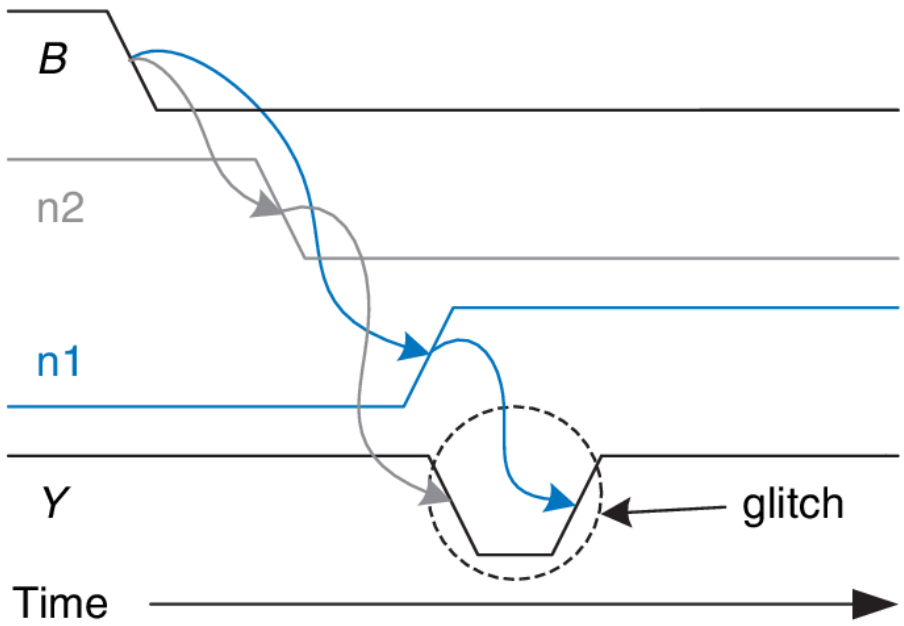
Timing Analysis
Propagation & Contamination Delay

- Delay is caused by
  - Capacitance and resistance in a circuit
  - Speed of light limitation
- Reason why t_{pd} & t_{cd} may be different:
  - Inputs may have different rise and fall time
  - Circuits slow down when hot and speed up when cold

- **Critical Path**: Long path from input to output
- **Short Path**: Shortest path from input to output




Critical Path
A = 0
B = 1→0
0→1
n1
Y = 1→0→1
n2
1→0
C = 1
Short Path


B
n2
n1
Y
glitch
Time

Synchronous or Sequential Circuits

- Output of sequential logic depends on the current and *prior* input values - it has **memory**
- Some Definitions:
  - **State**: All information about a circuit necessary to explain its future behaviour
  - **Latches and Flip-Flops**: State elements that store one binary bit
  - **Synchronous Circuit**: Combinational logic followed by a bank of flip-flops

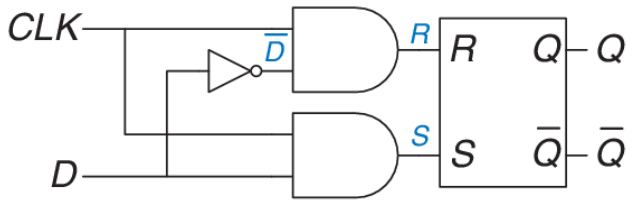
Synchronous or Sequential Logic

Synchronous Circuit & State Elements

- **Synchronous Circuit**:
  - Gives sequence to events
  - Has short-term memory
  - Uses feedback from output to store information
- **State Element:**
  - The current state of a circuit influences its future behavior
  - State elements store the current state
  - Types of State Elements:
    - Bistable Circuit
    - SR Latch
    - D Latch
    - D Flip-Flop & Registers

Synchronous or Sequential Logic

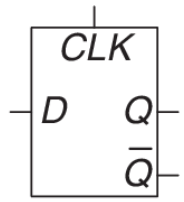
D Flip-Flop & Registers



(a)

CLK	D	$\overline{D}$	S	R	Q	$\overline{Q}$
0	X	$\overline{X}$	0	0	Q_{prev}	$\overline{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

(b)

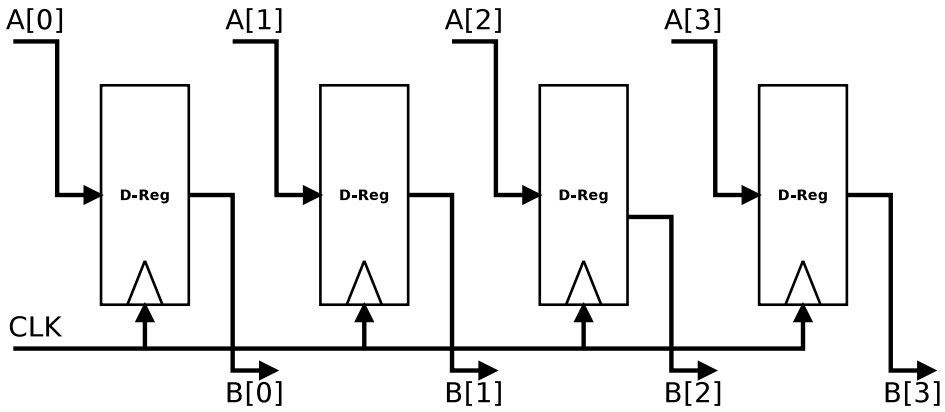


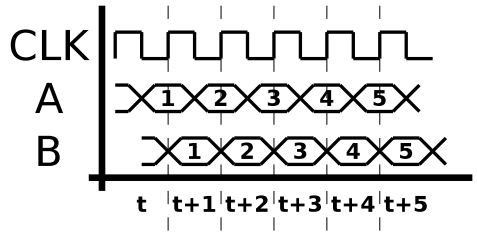
(c)

Synchronous or Sequential Logic

D Flip-Flop & Registers

- A flip-flop is a basic memory element built using logic gates. It is used to store state information
- A notation for time: Lets say t is the current time and $t + 1$ is the very next moment in future, t is usually specified in seconds, ms, us, ns, ps ...etc
- A register is a flip-flop triggered by a clock (CLK)

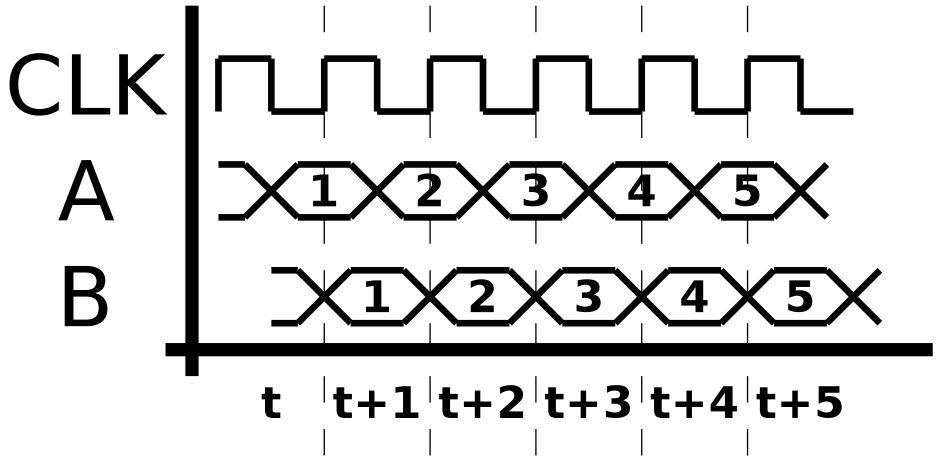




Synchronous or Sequential Logic

Clock Generators and Dividers

- A Clock Generator produces the "CLK", the timing signal
- In Digital logic, a square wave with 50% duty cycle and constant period, or frequency, is used for clock input
- Dedicated circuit is used to generate precise clock signal
- A clock divider or multiplier generates several phase aligned clock signals from a single clock input and each output frequency could be some multiple of input frequency

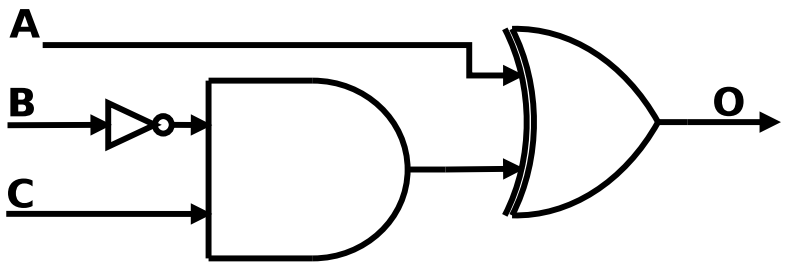


Combinatorial vs Synchronous or Sequential Logic

- **Combinatorial Logic**
  - Memoryless
  - Only current input values determine outputs
- **Synchronous or Sequential Logic**
  - Has memory
  - Current and previous input values determine outputs

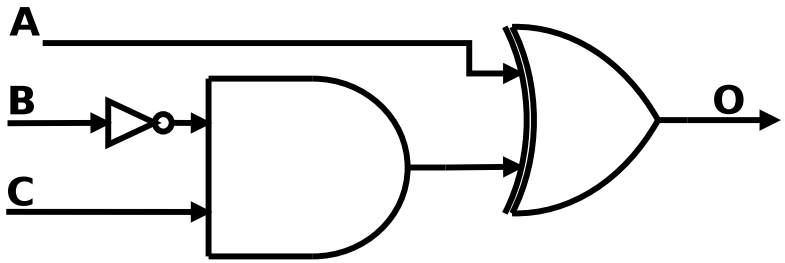
Combinatorial Logic

- A boolean equation is implemented using combinatorial logic



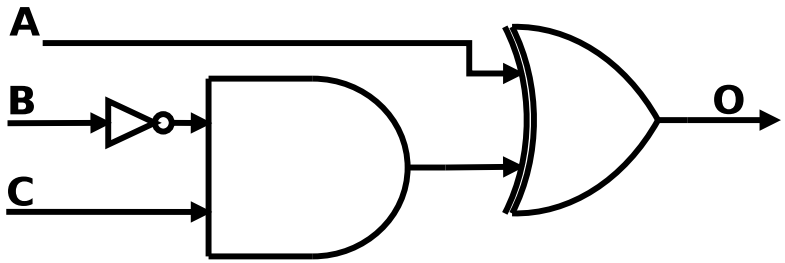
Combinatorial Logic

- A boolean equation is implemented using combinatorial logic
- Combinatorial logic contains logic gates, adders, multipliers, encoders ... etc.



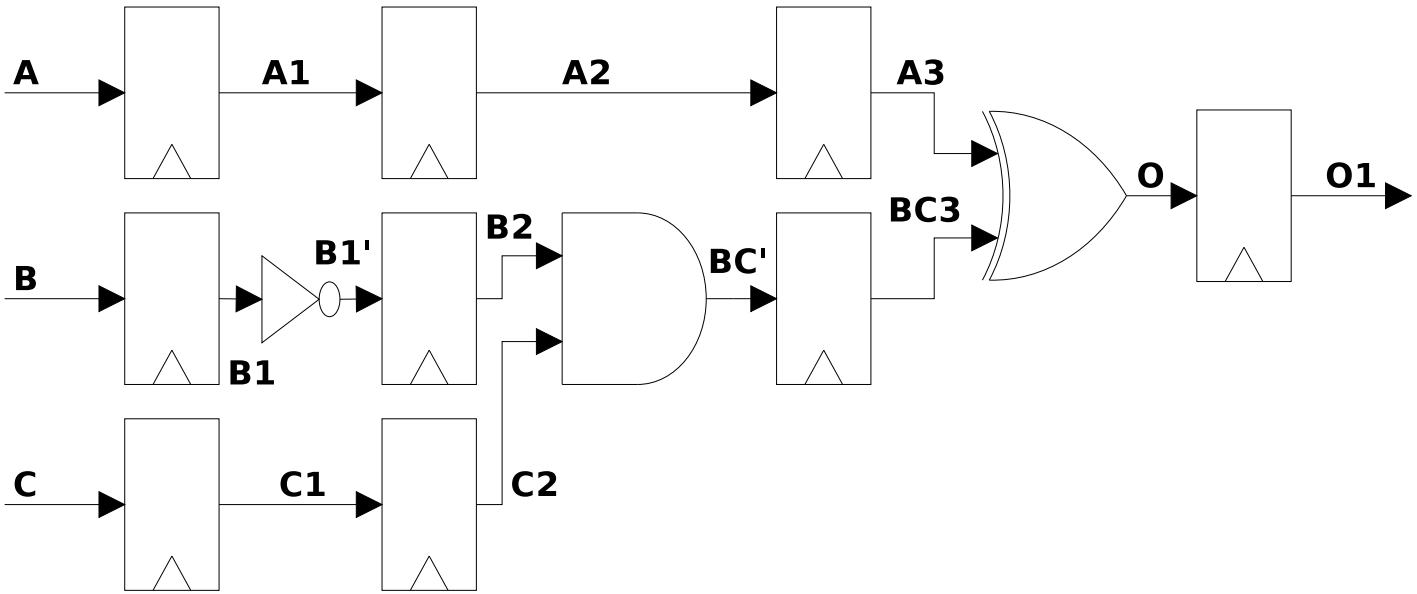
Combinatorial Logic

- A boolean equation is implemented using combinatorial logic
- Combinatorial logic contains logic gates, adders, multipliers, encoders ... etc.
- The output depend on the current state of the input



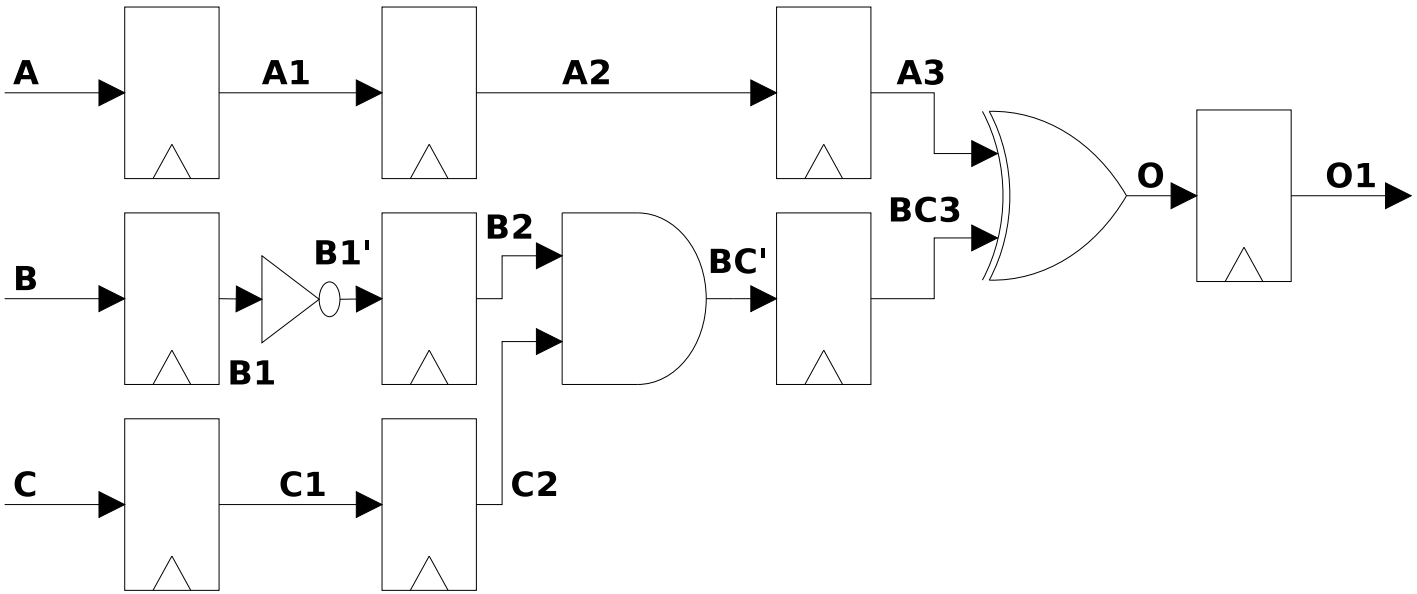
Synchronous Logic

- Uses both register and combinatorial logic

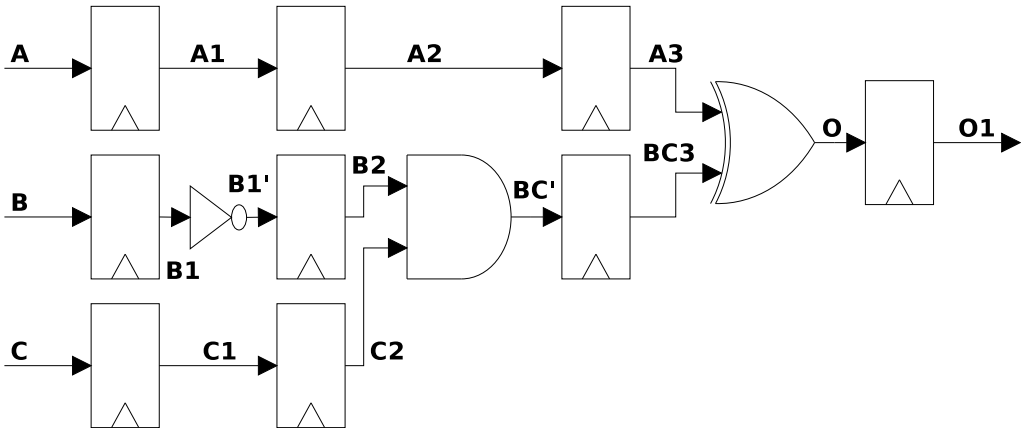


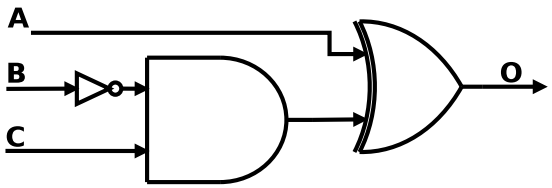
Synchronous Logic

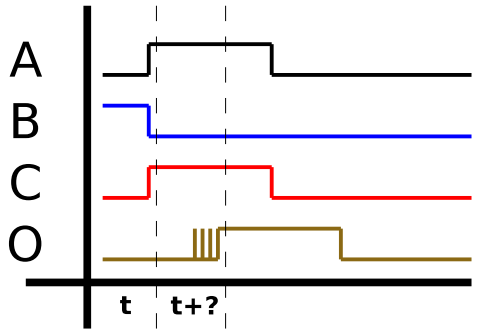
- Uses both register and combinatorial logic
- Registers are used to synchronize timing

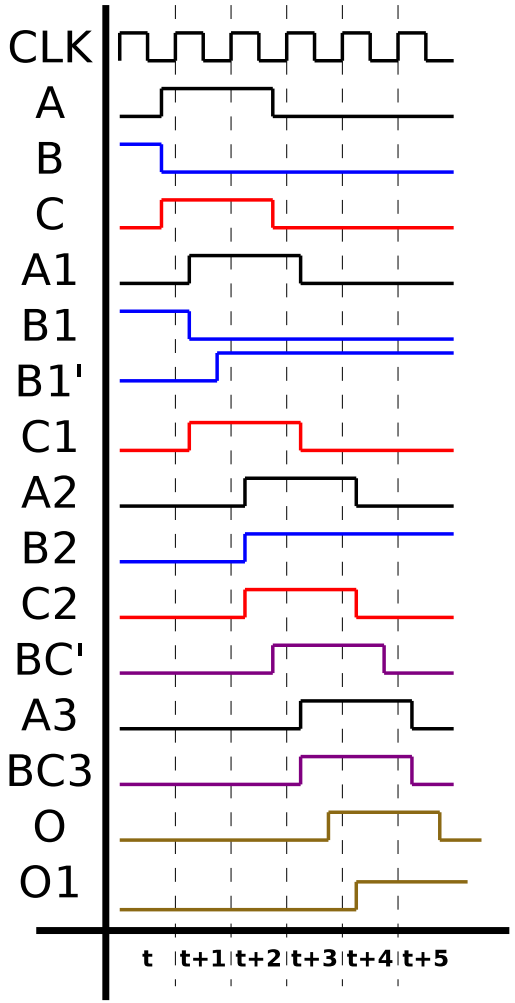


Combinatorial vs Synchronous Logic









Week 3 Activity 1

For activities 1 & 2 you should use slow_adder project file.

1. **Design Analysis**:
   1. List and explain the types of logic gates used in adder_16 and its sub-modules
   2. You have been tasked to build a 64-bit adder/subtractor. Could you use adder_16 as a sub-module to build a 64-bit adder/subtractor? If yes, how? if not, why?
   3. What is the maximum frequency (fmax) at which slow_adder will run reliably? Does it run at the same speed in all temperatures (0C and 85C)? If not, why?
   4. What is "out_reg" and its purpose? Give detailed description of this component.

Week 3 Activity 2

2. **Functional Analysis**:
   1. How does adder_16 perform subtraction?
   2. List, explain and write function specification for each sub-module within adder_16.
   3. What is the use of "Cout", an output of adder_16, and does it confirm to functional specification? If not, why? and how could you fix it?

Week 3 Activity 3 & 4

Use fast_adder project for the activities 3 & 4.

3. **Timing Analysis**:
   1. By design, how many logic levels are there in adder16?
   2. How fast (MHz) is the clock running in fast_adder?
   3. Does fast_adder meet fmax requirements? If not, why?
4. **Redesign**:
   1. Redesign fast_adder, in paper, to meet timing requirement.
   2. Implement your modification using Quartus II software and test it in DE0-Nano. You should make sure that your new design meets timing requirement before downloading it to DE0-Nano.