211: Computer Architecture
Spring 2021

Instructor: Prof. David Menendez

Topics:

- Digital Logic

- Reading material avail ai

Logic Design

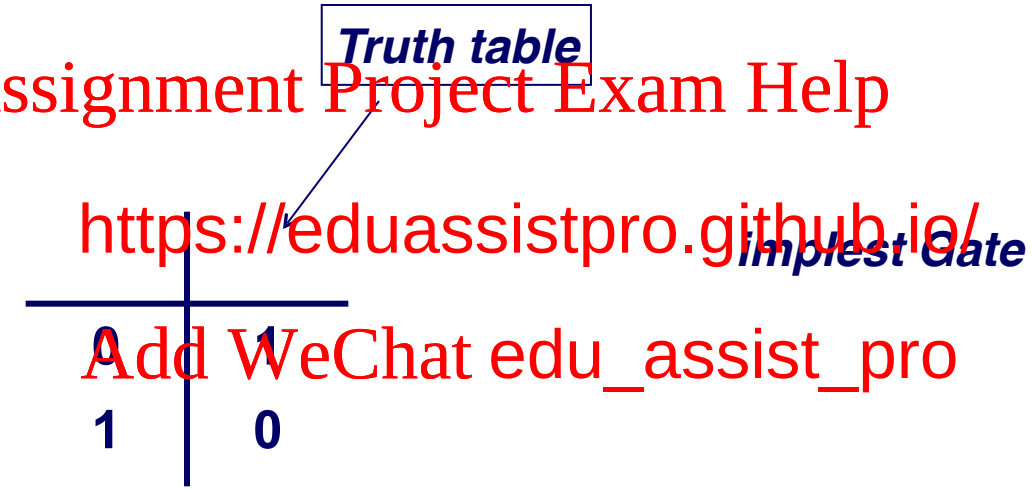
Logic Gates

Transition from representing information to implementing them

Logic gates are simple digital circuits

- Take one or m
- Produce a bin
- Truth table: relationship betwe and the output

Not Gate



AND Gate

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

Two inputs, One output

Result is 1 only if both the inputs are 1.

OR Gate

A	B	C
0	0	0
	1	1
	0	1
		1

Logical Completeness

Can implement ANY truth table with AND, OR, NOT.

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

1. AND combinations that yield a "1" in the truth table.

2. OR the results of the AND gates.

NAND and NOR Gate

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

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

Beneath the Digital Abstraction

A digital system uses discrete values

- Represent it with continuous variables (eg, voltage), handle noise

Use transistors to implement logical functions: AND, OR, NOT

Digital symbols:

- recall that we assi o each digital (logic) sym

- assignment of voltage ranges depends on electrical properties of transistors being used
  - typical values for "1": +5V, +3.3V, +2.9V
  - from now on we'll use +2.9V

Transistor: Building Block of Computers

Microprocessors contain millions (billions) of transistors

- Intel Pentium 4 (2000): 48 million
- IBM PowerPC 750FX (2002): 38 million
- IBM/Apple PowerPC G5 (2003): 58 million

Logically, each tra

Combined to implement logic fun

- AND, OR, NOT

Combined to build higher-level structures

- Adder, multiplexer, decoder, register, …

Combined to build processor

DeMorgan's Law

Converting AND to OR (with some help from NOT)

Consider the following gate:

A	B	$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$	$\overline{\overline{A} \cdot \overline{B}}$
0	0	1	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	0	1

To convert AND to OR (or vice versa), nputs and output.

DeMorgan's Laws:

1. $\overline{PQ} = \overline{P} + \overline{Q}$
2. $\overline{P + Q} = \overline{P}\ \overline{Q}$

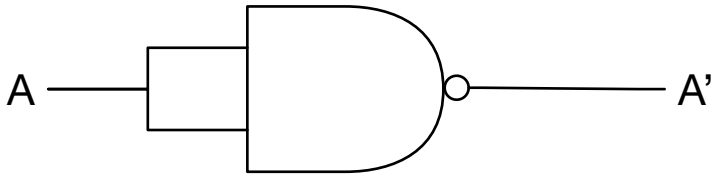
Same as A+B!

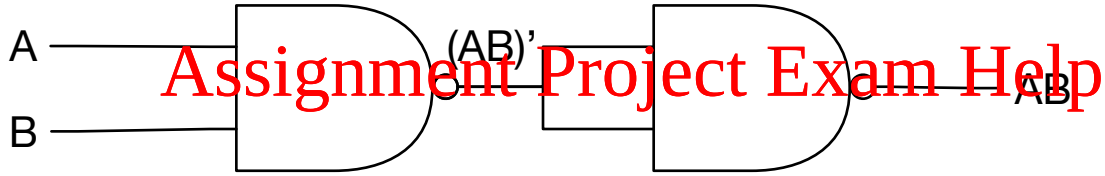
NAND and NOR Functional Completeness

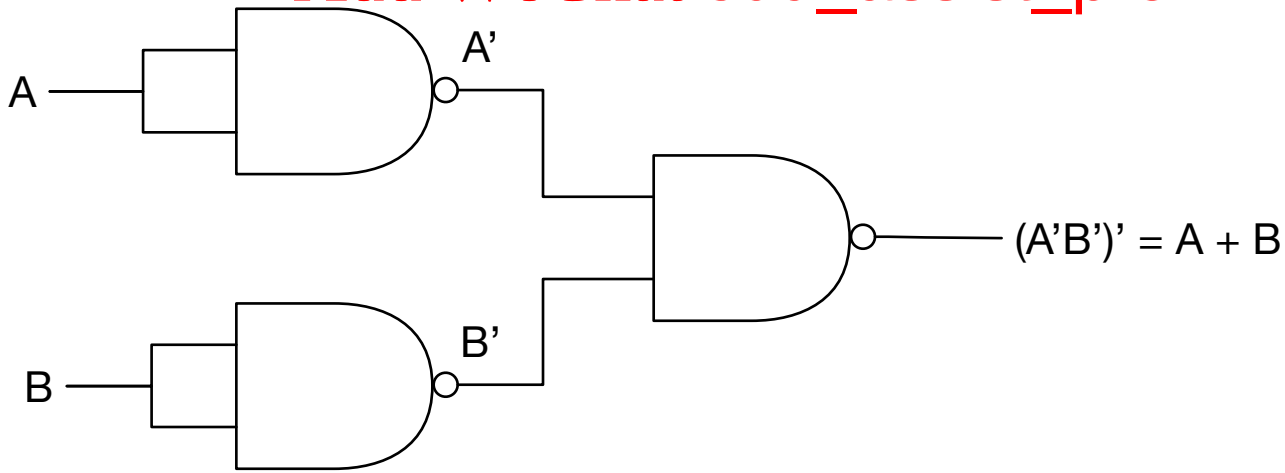
Any gate can be implemented using either NOR or NAND gates.

Why is this important?

- When building a chip, easier to build one with all of the same gates.







More than 2 Inputs?

AND/OR can take any number of inputs.

- AND = 1 if all inputs are 1.
- OR = 1 if any input is 1.
- Similar for NAND/NOR.

Can implement wit s or with single CMOS circuit.

Circuit Design

Have a good idea. What kind of circuit might be useful?

Derive a truth table for this circuit

Derive a Boolean expression for the truth table

Build a circuit given a Boolean expression

- Building the circuit is easy once you have the Boolean expression to actual gates. This part is easy.
- Deriving the Boolean expression is tricky. Deriving a good one is tricky.

Converting Truth Table to Boolean Expression

Given a circuit, isolate the rows in which the output of the circuit should be true

Converting Truth Table to Boolean Expression

Given a circuit, isolate that rows in which the output of the circuit should be true

A product term that contains exactly one instance of every variable is called a minterm

Converting Truth Table to Boolean Expression

Given the expressions for each row, build a larger Boolean expression for the entire table.

- This is a sum-of-products (SOP) form.

Canonical Forms

We have studied two canonical forms

1. Sum of Products (SoP)
2. Product of Sums (PoS)

How to convert to SoP from PoS (multiple through)

How to convert to PoS fr hrough, complement via DeMorgan's)

Formal Definition of Minterms

Converting Truth Table to Boolean Expression

Finally build the circuit.

- Problem: SOP forms are often not minimal.
- Solution: Make it minimal. We'll go over two ways.

First Approach: Algebraic

Simply use the rules of Boolean logic

The Result

Decoder

n inputs, 2^n outputs

- exactly one output is 1 for each possible input pattern

2-bit decoder

Decoder Circuits

Decoder Example

Internal 2:4 Decoder Design

2:4 Decoder from 1:2 Decoders

Hierarchical 3:8 Decoder

Encoder: Inverse of Decoder

Inverse of decoder: converts m bit input to n bit output

(n <= m)

Multiplexer (MUX)

n-bit selector and 2^n inputs, one output

- output equals one of the inputs, depending on selector

4-to-1 MUX

Multiplexers (Muxes)

Combinational circuit that selects binary information from many inputs to one output

Functions with Decoders or Muxes

Can we do it a Smaller Mux?

Can actually use a smaller mux with a trick: $F = A\overline{C} + BC$

Look at the rows below, A & B have the same value, C iterates between 0 & 1

For the pair of rows, F either equals 0, or 1, C or not(C)

Another Example

Where are we?

We have already seen

-- Basic gates: AND, NOT, OR

-- Building blocks: Decoder and Multiplexer

-- Implement circuit

-- We know: (a) mi cts

-- We know basic identities

Implement A+B

With Multiplexers

(1) Using 2:1 mux

(2) Using 4:1 mux

With Decoders

(1) Using a 2:4 decoder

Half Adder

Add two bits and produce a sum and a carry.

How do we go about building the circuit?

Full Adder

Add two bits and carry-in,
produce one-bit sum and carry-out.

A	B	C_{in}	S	C_{out}
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

Four-bit Adder

Karnaugh Maps or K-Maps

K-maps are a graphical technique to view minterms and how they relate.

The “map” is a diagram made up of squares, with each square representing a single minterm.

Minterms resulting in “1”, all others are marked “0”

2 Variable K-Map

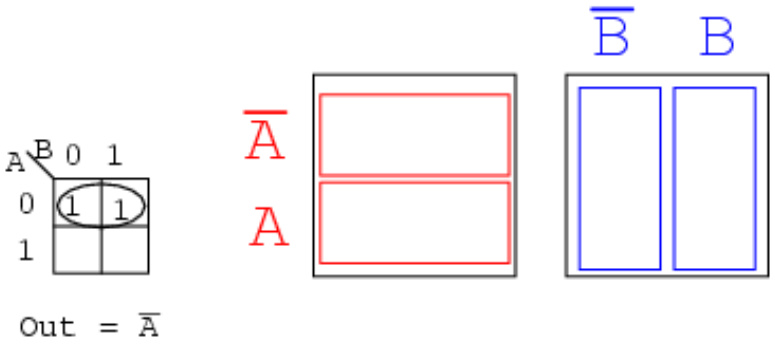
2 Variable K-Map

2 Variable K-Map

0 1

0 1

Finding Commonality



Finding the “best” solution

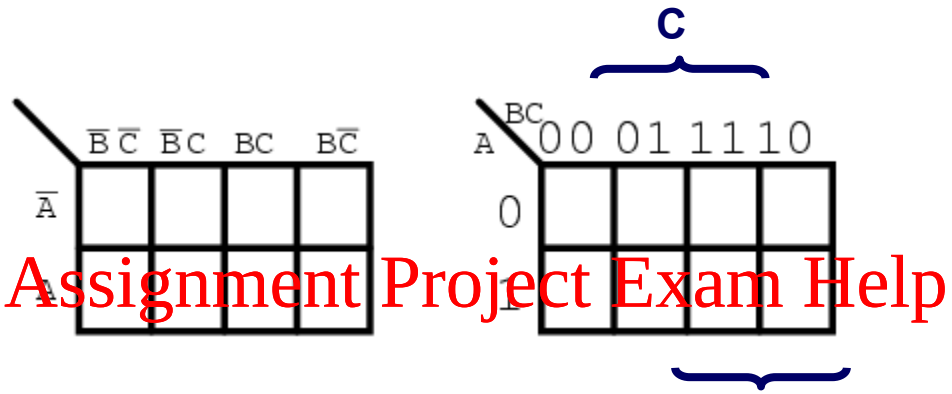
Grouping become simplified products.

Both are “correct”. “A+B” is preferred.

Simplify Example

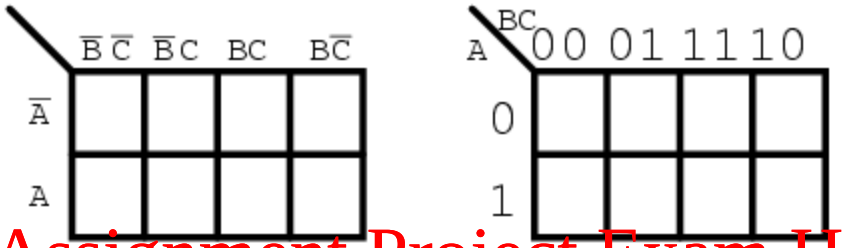
Simplify Example

3 Variable K-Maps

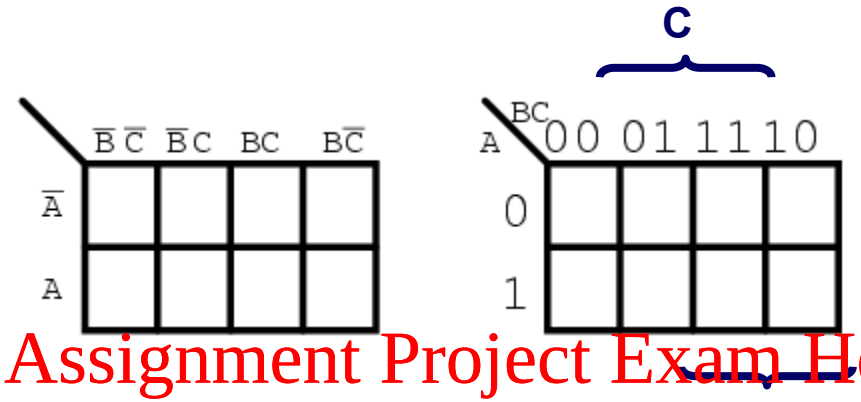


- Note in higher maps, several var py a given axis
- The sequence of 1s and 0s follo de Sequence.
- Grey code is a number system where two successive values differ only by 1-bit

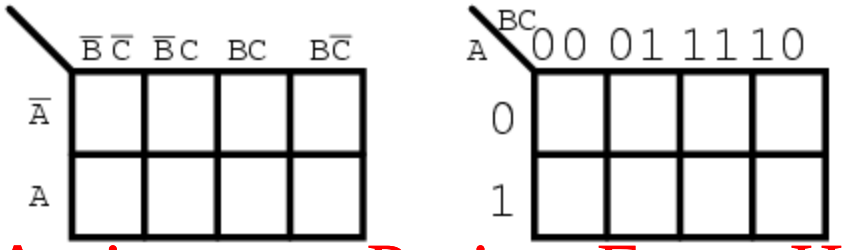
3 Variable K-Maps



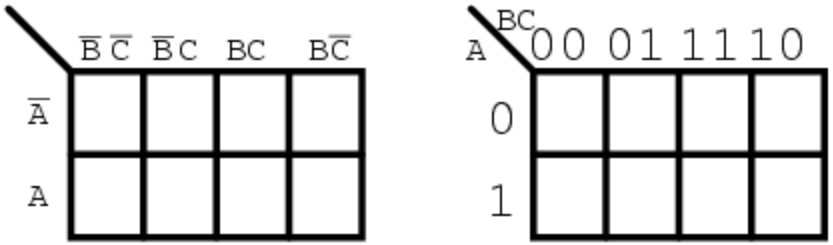
3 Variable K-Maps



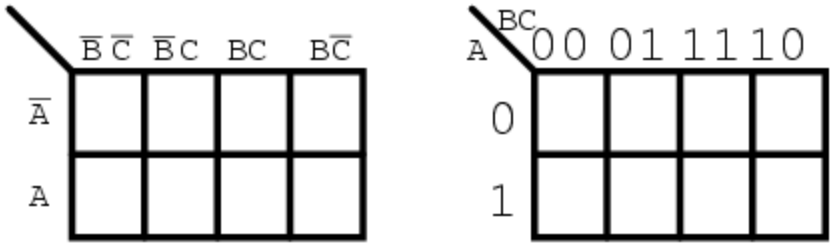
3 Variable K-Maps



3 Variable K-Maps



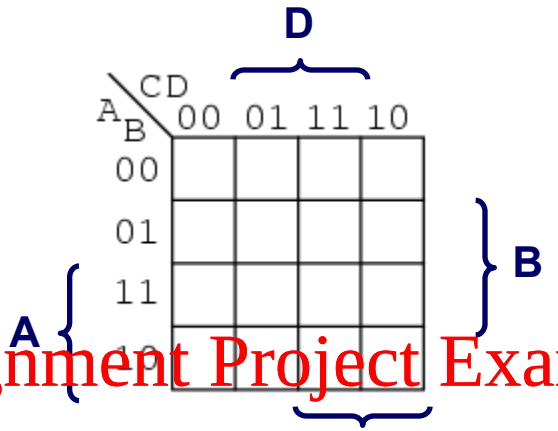
3 Variable K-Maps



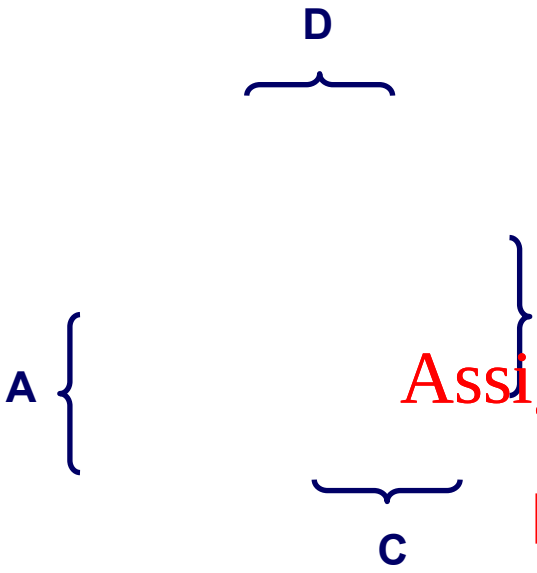
Back to our earlier example…..

The K-map and the algebraic produce the same result.

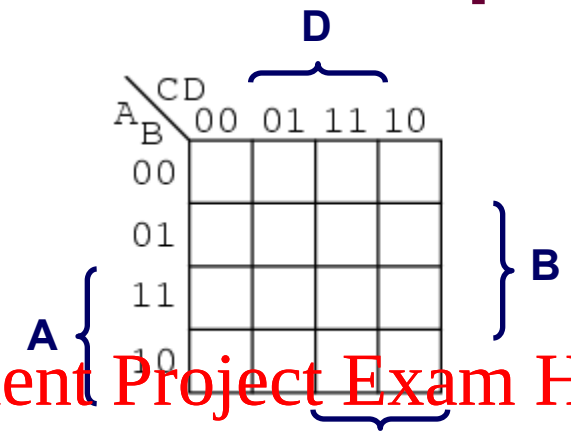
Up… up… and let's keep going



Few more examples



Few more examples



Don’t Care Conditions

Don’t Cares can Greatly Simplify Circuits

Design Example

Design Example

Design Example

We will do f, but you should be able to design a-e as well

$+X\overline{Y}$ $+ X \overline{Y}$

Combinational Circuits

Stateless circuits

Outputs are function of inputs only

Time and State

SEQUENTIAL CIRCUITS

How are Sequential Circuits different from Combinational Circuits?

Outputs of sequential logic depend on both current and prior values – it has memory

Definitions:

State: all the infor explain its future behavior

Latches and flip-flops: state elements that store one bit of state

Synchronous sequential elements: combinational logic followed by a bank of flip-flops

Enabler Circuits

Bistable Circuits

Fundamental building blocks of other elements

No inputs

Two outputs (Q and Q')

Bistable Circuit Analysis

Consider all the cases

Bistable circuit stores 1 bit of state (Q, or Q')

But there are no inputs to control state

Set/Reset Latch

S/R Latch Analysis

S/R Latch Analysis

S/R Latch Symbol

Set operation – makes output 1 (S = 1, R = 0, Q = 1)

Reset operation – makes output 0 (S = 0, R = 1, Q = 0)

What about invalid state? (S = 1, R = 1)

D Latch

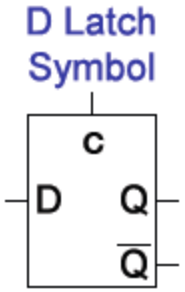
Two inputs (C and D)

C: controls when the output changes

D (data input): controls what the output changes to

When C = 1, D pas parent latch)

When C = 0, Q hol ue latch)



D Latch Internal Circuit

How to Coordinate with Multiple Components?

But how do we coordinate computations and the changing of state values across lots of different parts of a circuit?

We use CLOCKING (eg. 2.6GHz clock on Intel processors)

On each clock puls tations are performed, and results stored in l

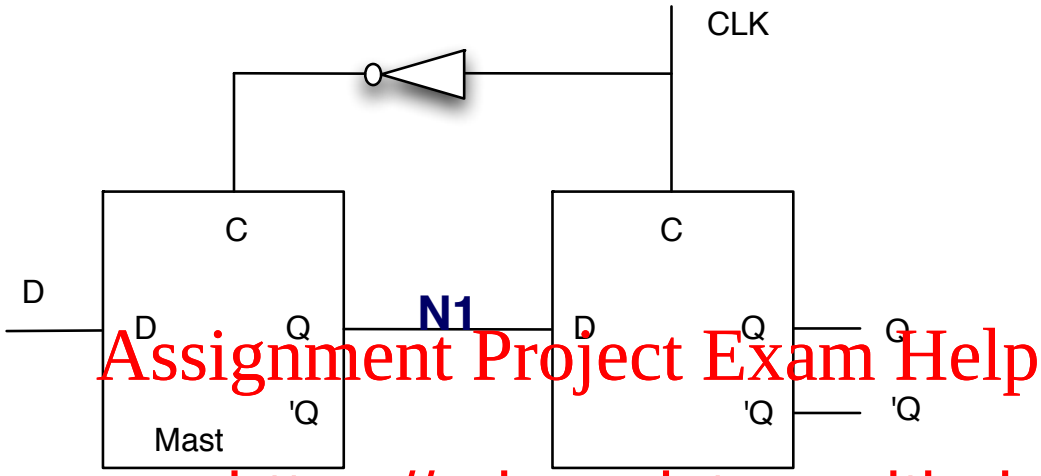
How to introduce clocks into latches?

Flip-flops: Latches on a Clock

A straightforward latch is not safely synchronous (or predictably synchronous)

Flip-flops designed so that outputs will NOT change within a single clock pulse

D Flip-Flop



When CLK is 0

- master is enabled (N1 obtains put to the master)
- slave is disabled (Old output is still output)

When CLK is 1

- then master is disabled (N1 is the old value)
- Slave is enabled, it copies N1 into output

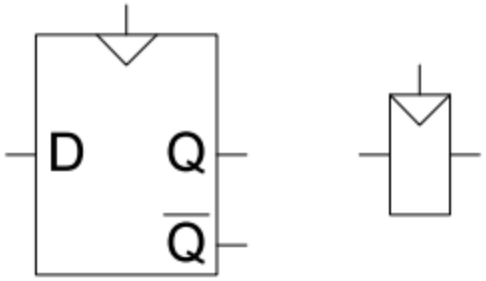
D Flip-Flop Summary

Two inputs: Clk, D

Function

- The flip-flop samples D on rising clock edge
- When clock goes from 0 to 1, D passes through Q
- Otherwise, Q holds
- Q only changes on rising clock edge
- Flip-flop is called "edge-triggered" because it is activated only on the clock edge

D Flip-Flop Symbols

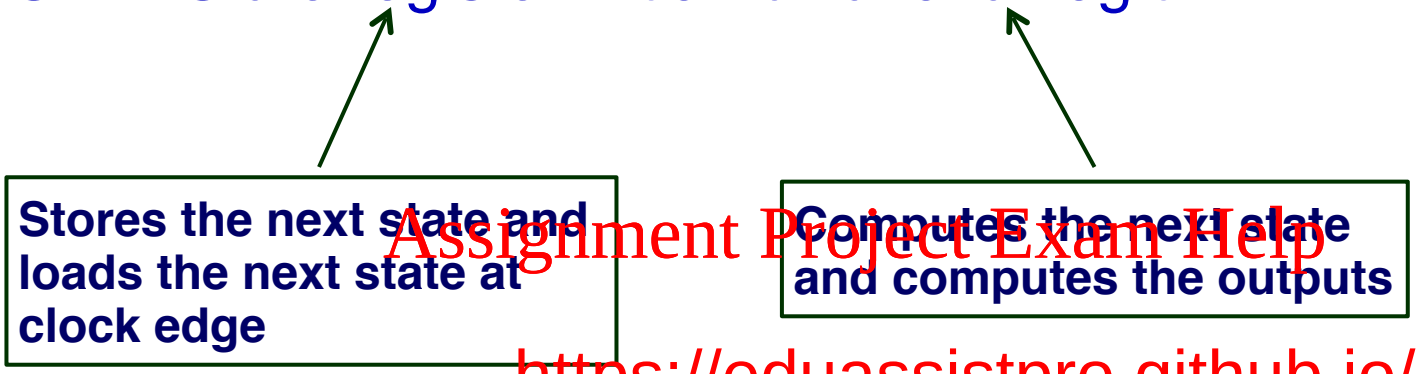


Flip-Flop versus Latch

Registers

Finite State Machines

FSM = State register + combinational logic



Traffic Light Controller Example

FSM State Transition Diagram

FSM State Transition Table

State transitions from diagram can be rewritten in a state transition table

Encoded State Transition Table

After selecting a state encoding, the symbolic states in the transition table can be realized with current state/next state bits

Computing Next State Logic

From K-maps, figure out expressions for e:

$$S_1' = S_1 \oplus S_0$$
$$S_0' = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\,\overline{S_0}\,\overline{T_B}$$

FSM Output Table

FSM output logic is computed in similar manner as next state logic

In this system, output is a function of current state (Moore machine)

Alternative – Mealy machine (output function of both current state and inputs, though we won't cover this in class)

$$L_{A1} = S_1; L_{A0} = \overline{S_1}\, S_0$$
$$L_{B1} = \overline{S_1}; L_{B0} = S_1\, S_0$$

State Register: Assume D-FF

FSM: Figure out Next State Logic

$$S_1' = S_1 \oplus S_0$$
$$S_0' = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\,\overline{S_0}\,\overline{T_B}$$

FSM: Figure out Output Logic

$$L_{A1} = S_1; L_{A0} = \overline{S_1}\, S_0$$
$$L_{B1} = \overline{S_1}; L_{B0} = S_1\, S_0$$

FSM Example 2

Design an FSM that detects a stream of three or more consecutive 1s on an input stream

Input: 0 1 1 0 1 0 1 1 0 1 1 1 0 1 …

Output: 0 0 0 1 0 0 …

Finite State Machine for the 3 1's problem

FSM Truth Table

Truth Table for Next State (AN and BN are next states)

A	B	X	AN	BN
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1

Encoding

	A	B
S0	0	0
S1	0	1
S2	1	0
S3	1	1

We need two bits to encode 4 states (lets call these bits A & B)

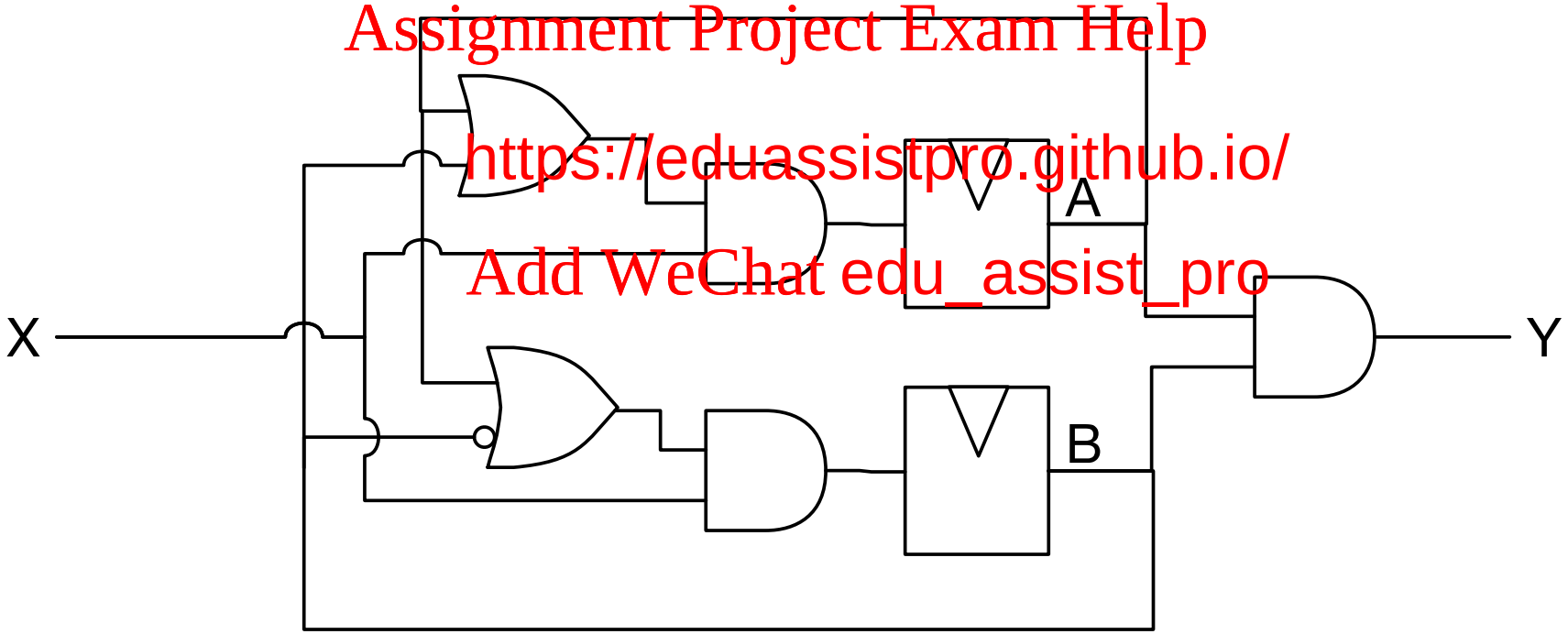
FSM with D-Flip Flops

$$A' = A\,X + B\,X = (A + B)\,X \qquad + \overline{B}\,X = (A + \overline{B})\,X$$

Y = AN. BN

FSM Circuit

$$A' = A\,X + B\,X = (A + B)\,X$$
$$B' = A\,X + \overline{B}\,X = (A + \overline{B})\,X$$



Backup

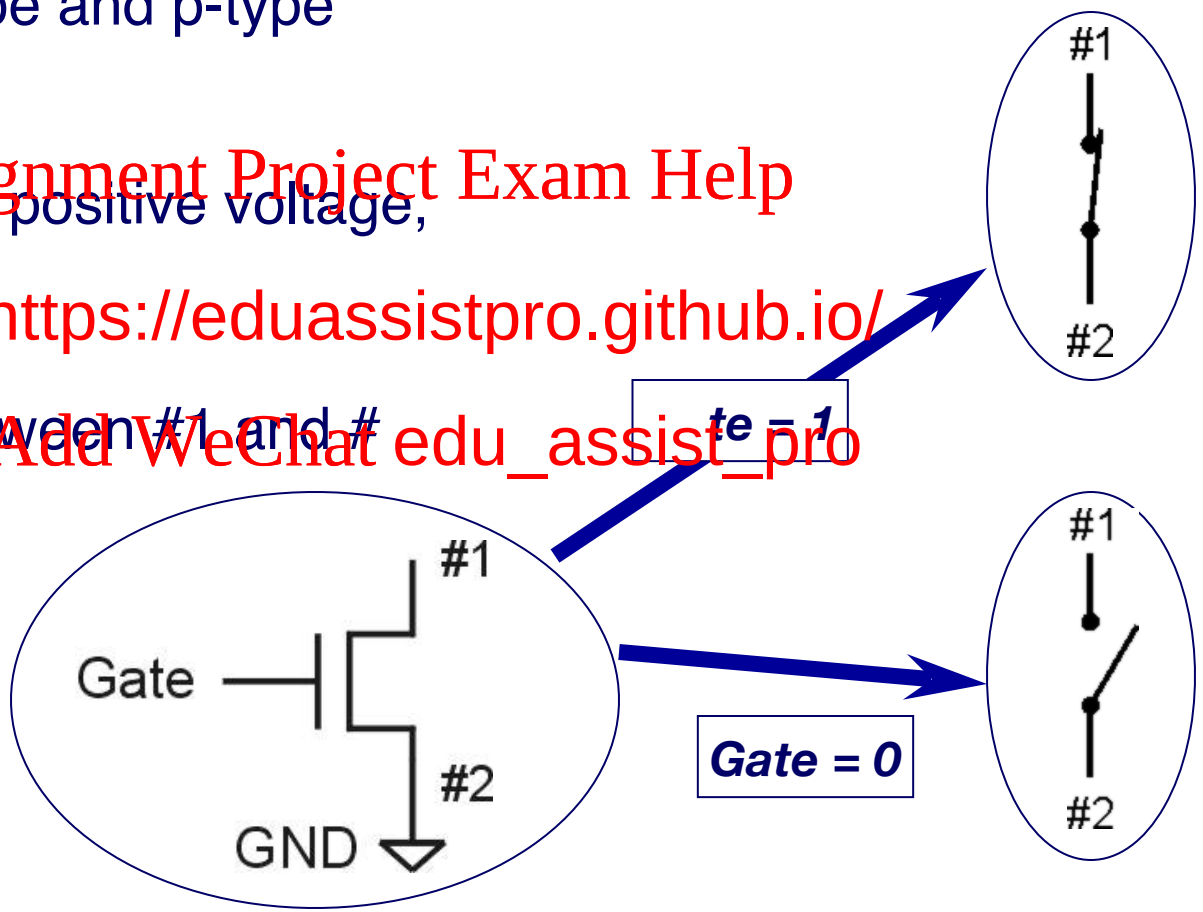
n-type MOS Transistor

MOS = Metal Oxide Semiconductor

- two types: n-type and p-type

n-type

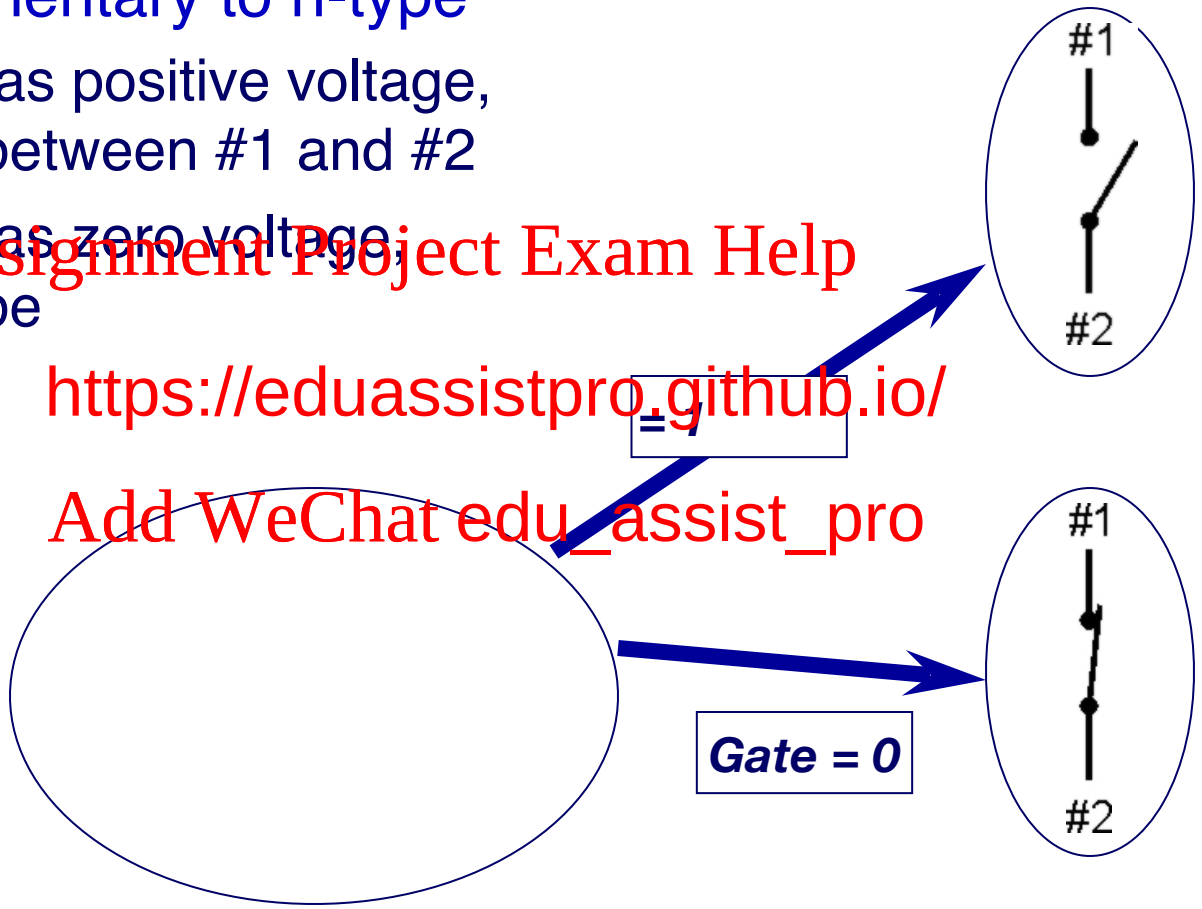
- when Gate has positive voltage, short circuit be
- when Gate ha open circuit between #1 and #



p-type MOS Transistor

p-type is complementary to n-type

- when Gate has positive voltage, open circuit between #1 and #2
- when Gate has zero voltage, short circuit be



CMOS Circuit

Complementary MOS

Uses both n-type and p-type MOS transistors

- p-type
  - Attached to + voltage
  - Pulls output to zero
- n-type
  - Attached to GND
  - Pulls output voltage DOWN when input is one

MOS transistors are combined to form Logic Gates

For all inputs, make sure that output is either connected to GND or to +, but not both!

Inverter (NOT Gate)

In	Out
0 V	2.9 V
2.9 V	0 V

In	Out
0	1
1	0