CS8803: Advanced Digital Design for Embedded Hardware

Lecture 4:
Latches, Flip-Flops, and Sequential Circuits

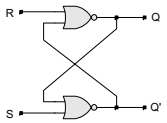
Instructor: Sung Kyu Lim (limsk@ece.gatech.edu)

Website: http://users.ece.gatech.edu/limsk/course/cs8803

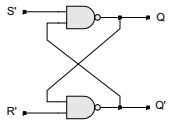
STORAGE ELEMENTS: R-S LATCH

- Storage elements are used to store information in a binary format (e.g. state, data, address, opcode, machine status).
- Storage elements may be clocked or unclocked.
- Two types: level-sensitive, edge-sensitive

Example: R-S latch (Unclocked) The state of an R-S latch is dependent on the value of its R and S inputs.



S	R	Q_{next}	Q'_{next}	
0	0	Q	Q'	Hold
0	1	0	1	Reset
1	0	1	0	Set
1	1	0	0	Not Allowed

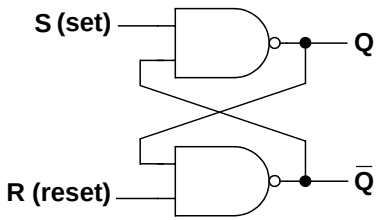


S'	R'	Q_{next}	Q'_{next}	
0	0	1	1	Not Allowed
0	1	1	0	Set
1	0	0	1	Reset
1	1	Q	Q'	Hold

Note: Avoid applying *11* to a R-S Nor latch, and *00* to an R'S' Nand latch. The circuit is unstable and oscillation will result.

LATCHES
$\bar{S}\bar{R}$ LATCH (NAND GATES)

- NAND gates can also be used to create a latch, this time an $\bar{S}\bar{R}$ latch.



S	R	Q	$\bar{Q}$	
1	0	0	1	
1	1	0	1	(after S = 1, R = 0)
0	1	1	0	
1	1	1	0	(after S = 0, R = 1)
0	0	1	1	

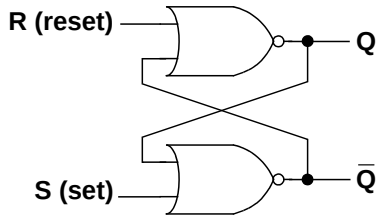
Recall:

A	B	NAND
0	0	1
0	1	1
1	0	1
1	1	0

- Notice that this latch is level-sensitive.

LATCHES
SR LATCH (NOR GATES)

- The SR latch also uses feedback to "store" a bit.



S	R	Q	$\bar{Q}$	
1	0	1	0	
0	0	1	0	(after S = 1, R = 0)
0	1	0	1	
0	0	0	1	(after S = 0, R = 1)
1	1	0	0	

Recall:

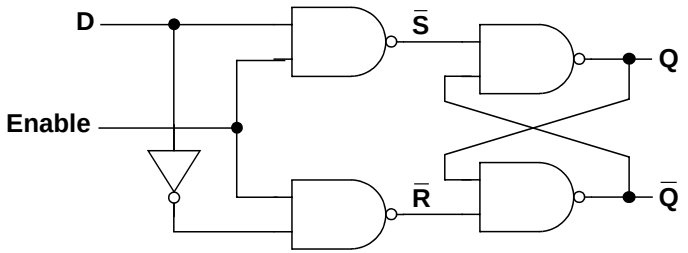
A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

- Notice that this latch is level-sensitive.

LATCHES

D LATCH (WITH $\overline{S}\overline{R}$ LATCH)

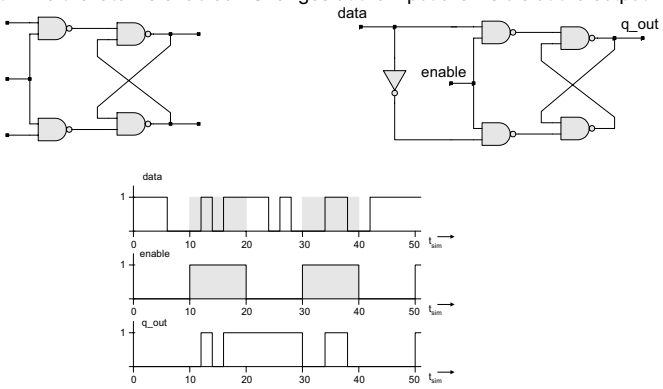
- A D latch can be implemented using what is effectively the SR latch with a control line as follows.



- Note that as long as **C** = **1**, that the latch will change according to the value of **D**.

STORAGE ELEMENTS: TRANSPARENT LATCHES

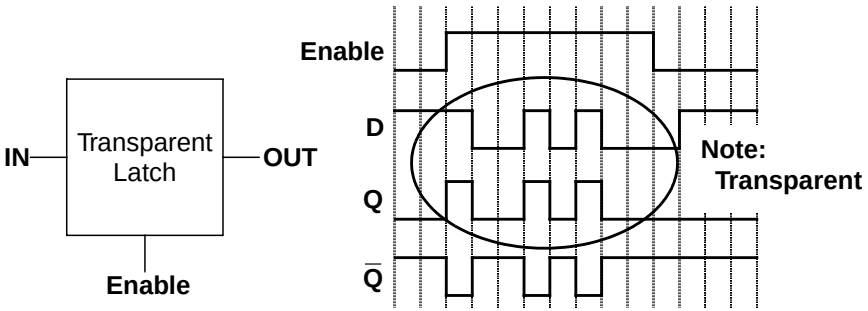
Latches are level-sensitive storage elements; data storage is dependent on the level (value) of the input clock (or enable) signal. The output of a transparent latch changes in response to the data input while the latch is enabled. Changes at the input are visible at the output



LATCHES

TRANSPARENCY (1)

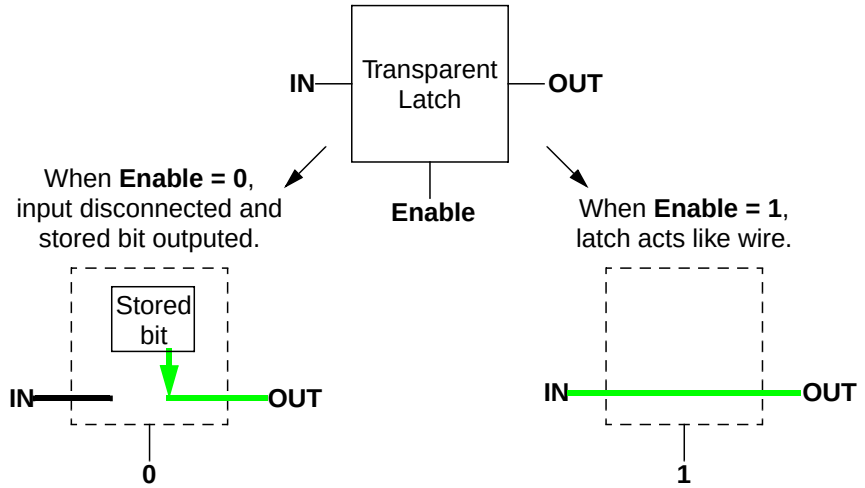
- Latches like the D latch are termed **"transparent"** or **level-sensitive**.
  - This is because, when enabled, the **output follows the input**.



LATCHES

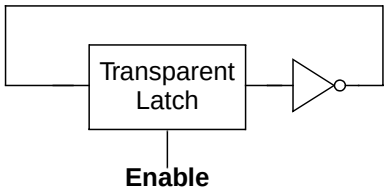
TRANSPARENCY (2)

- The following behaviour is observed for Enable = 0 and Enable = 1.



LATCH EXAMPLE
PROBLEMS W/ TRANSPARENCY

- A problem with latches is that they are level-sensitive.
  - A momentary change of input changes the value passed out of the latch.
- This is a problem if the input of a latch depends on the output of the same latch.
  - Example: Design a system that flips a stored bit whenever **Enable** goes high. An inexperienced engineer might design the following.



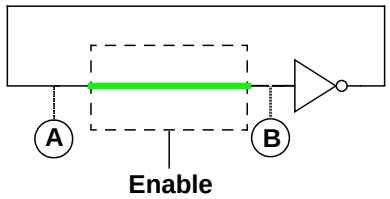
How will this design behave?

Will the bit flip once when the **Enable** signal goes high?

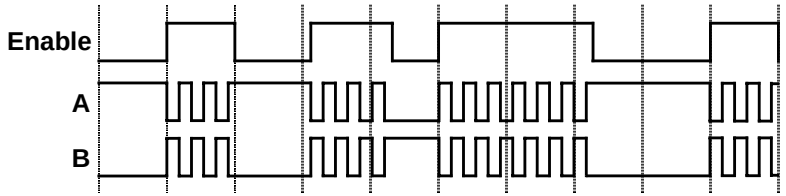
Answer: The output will follow the input, which happens to keep changing.

LATCH EXAMPLE
PROBLEMS W/ TRANSPARENCY

- Let's analyze the timing behaviour of this "poor" design.

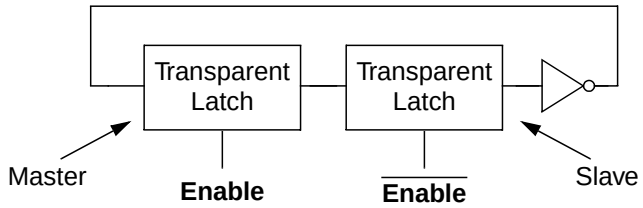


- Notice that instead of the desired bit flip when **Enable=1**, that the input oscillates. This is because the output depends directly on the input since **A** and **B** appear to be connected by a wire.



LATCH EXAMPLE
ELIMINATING TRANSPARENCY

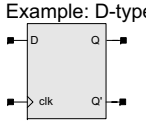
- The problem with transparent, level-sensitive latches can be fixed by splitting the input and output so that they are independent.
  - New solution: Consider the following improved design that flips a stored bit whenever **Enable** goes high. This design now uses a master and a slave transparent latches to separate the input from the output.



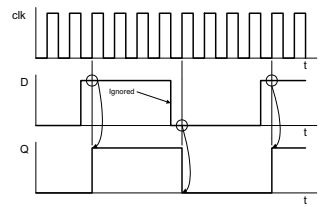
STORAGE ELEMENTS: FLIP-FLOPS

Flip-flops are edge-sensitive storage elements; data storage is synchronized to an edge of a clock. The value of data stored depends on the data that is present at the data input(s) when the clock makes a transition at its active (rising or falling) edge.

Example: D-type flip-flop



D	Q	Q_{next}
0	0	0
0	1	0
1	0	1
1	1	1

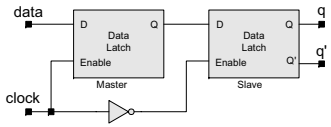


- Characteristic equation: $q_{next} = D$.
- This example is active on the rising (positive) edge of the clock.
- Intermediate data transitions are ignored.
- Timing constraints (setup, hold, minimum pulse width) must be met.

MASTER-SLAVE FLIP-FLOP

A master-slave configuration of two data latches samples the input during the active cycle of the clock applied to the master stage. The input is propagated to the output during the slave cycle of the clock.

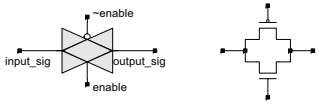
Master-slave implementation of a negative edge-triggered D-type flip-flop:



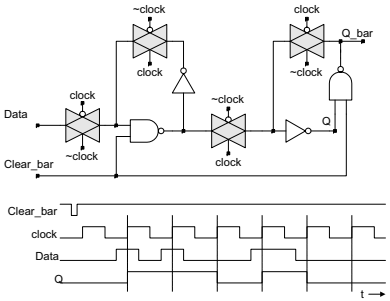
Timing constraint: the output of the master stage must settle before the enabling edge of the slave stage. The master stage is enabled on the inactive edge of the clock, and the slave stage is enabled on the active edge. Timing constraints apply to the active edge.

CMOS TECHNOLOGY - MASTER-SLAVE FLIP-FLOP

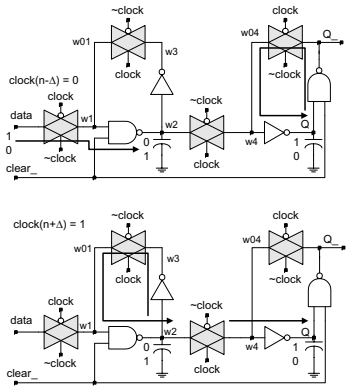
CMOS Transmission Gate:



D-type flip-flops in CMOS technology are formed by combining transmission gates with glue logic to form a master-slave circuit.



CMOS TECHNOLOGY MASTER-SLAVE FLIP-FLOP (Cont.)



- Master stage: output capacitor (node *w2*) is charged and sustained by the feedback loop. The delays of the master stage determine the setup conditions of the flip-flop.
- Slave stage: The output of the slave stage is sustained while the master stage is charging. At the active edge of the flip-flop, the output of the master stage charges the output of the slave stage, which is sustained by the feedback loop during the active cycle.
- Note: the read operation is non-destructive.

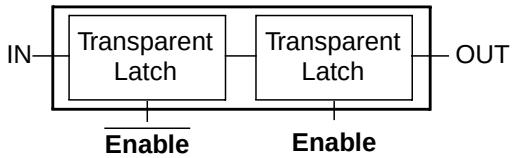
INTRO. TO COMP. ENG.
CHAPTER VII-21
SEQUENTIAL SYSTEMS

FLIP-FLOPS

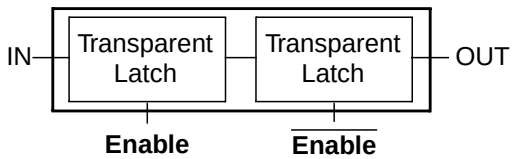
EDGE TRIGGERED

- **LATCH EXAMPLE**
- **FLIP-FLOPS**
  - **-SINGLE BIT STORAGE**

- A common and useful type of flip-flop are edge triggered flip-flops.
  - Positive edge triggered flip-flops



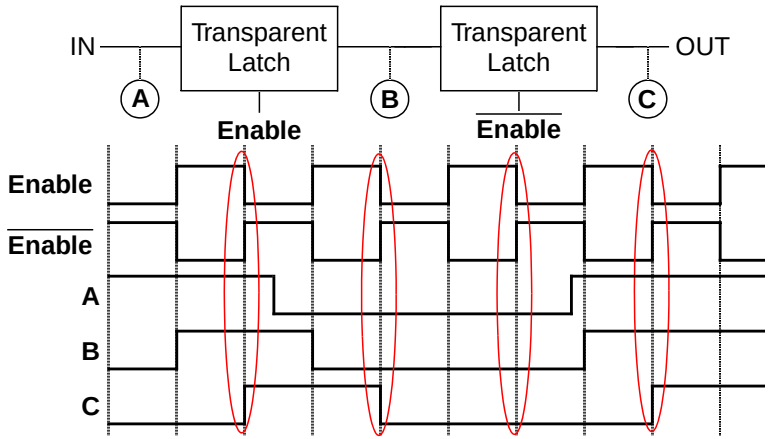
  - Negative edge triggered flip-flops



R.M. Dansereau; v.1.0

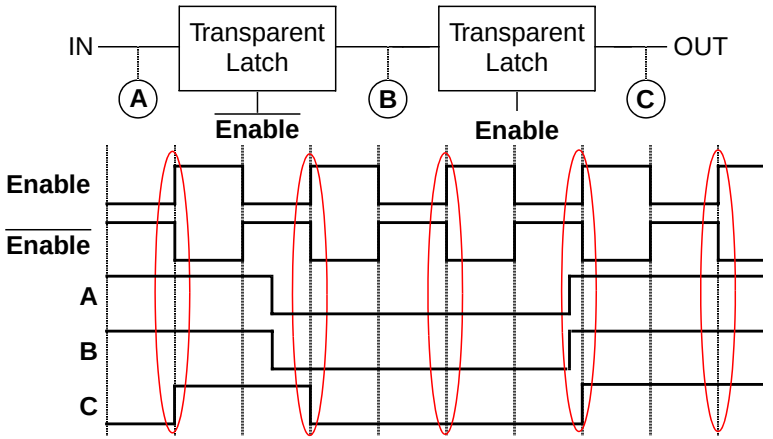
FLIP-FLOPS
NEGATIVE EDGE TRIGGERED

- The output C, which is also the bit stored, appears to change on the negative edge of the Enable transitions.



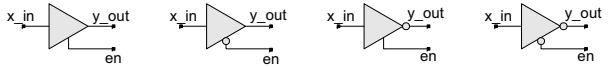
FLIP-FLOPS
POSITIVE EDGE TRIGGERED

- The output C, which is also the bit stored, appears to change on the positive edge of the Enable transitions.



BUILDING BLOCKS: THREE-STATE DEVICES

Three-state devices provide high-impedance interface devices.



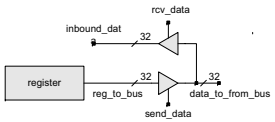
x_in	en	y_out
0	0	Hi-Z
0	1	0
1	0	Hi-Z
1	1	1

x_in	en	y_out
0	0	0
0	1	Hi-Z
1	0	1
1	1	Hi-Z

x_in	en	y_out
0	0	Hi-Z
0	1	1
1	0	Hi-Z
1	1	0

x_in	en	y_out
0	0	1
0	1	Hi-Z
1	0	0
1	1	Hi-Z

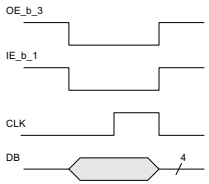
Typical applications: i/o pad and bus isolation.

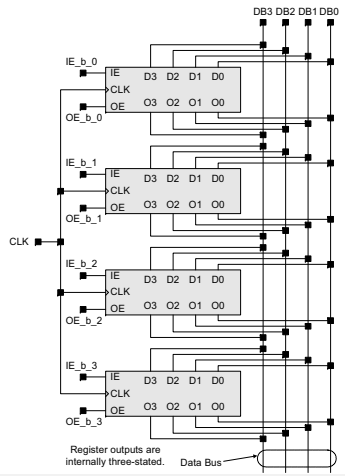


BUILDING BLOCKS: BUSSES

- Busses provide parallel datapaths and control interfaces and between functional units.
- Synchronous and asynchronous busses
- Handshaking protocols are required for coherent communication
- Key Issues: Bus Contention and Arbitration

Example: Register-to-Register transfer on a 4-bit datapath.





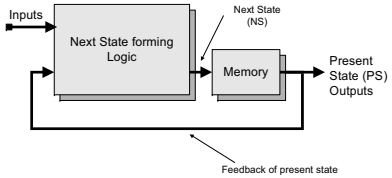
SEQUENTIAL MACHINES (p 80)

- Sequential machines, also called finite state machines, are characterized by an input/output relationship in which the value of the outputs at a given time depend on the history of the applied inputs as well as their present value.

  Example: A machine that is to count the number of *1s* in a serially transmitted frame of bits.

- The history of the inputs applied to a sequential machine is represented by the **state** of the machine, and requires hardware elements that store information, i.e. requires memory to store the state of the machine as an encoded binary word.

- All sequential machines require feedback that allows the next state of the machine to be determined from the present state and inputs.

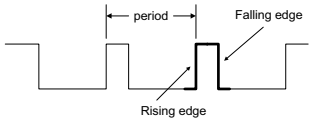


The set of states of a sequential machine is always finite, and the number of states is determined by the number of bits that represent the state.

SEQUENTIAL MACHINES (Cont.)

- Sequential machines may be asynchronous or synchronous (clocked).

- The state transitions of a (edge-triggered) flip-flop-based synchronous machine are synchronized by the active edge (i.e. rising or falling) of a common clock. State changes give rise to changes in the combinational logic that determines the next state and the output of the machine.



- A lower bound on the cycle time (period) of the machine's clock is set by the requirement that the period of the clock must be long enough to allow all transients activated by an a transition of the clock to settle at the outputs of the combinational logic before the next active edge occurs.

SEQUENTIAL MACHINES (Cont.)

- The inputs to the flip-flops must remain stable for a sufficient interval before and after the active edge of the clock. The former constraint establishes an upper bound on the longest path through the circuit, which constrains the latest allowed arrival of data. The latter constraint imposes a lower bound on the shortest path through the combinational logic that is driving the storage device. It constrains the earliest time at which data from the previous cycle could be overwritten.

- Together, these constraints ensure that valid data is stored. Otherwise, timing violations may occur at the inputs to the flip-flops, with the result that invalid data is stored.

- In an edge-triggered clocking scheme, the clock isolates a storage register's inputs from its output, thereby allowing feedback without race conditions.

- The outputs of a state machine controls the synchronous datapath operations and register operations of more general digital machine.

FINITE STATE MACHINES

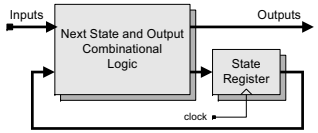
- Synchronous (i.e. clocked) finite state machines (FSMs) have widespread application in digital systems, e.g. as datapath controllers in computational units and processors. Synchronous FSMs are characterized by a finite number of states and by clock-driven state transitions.

- Mealy Machine: The next state and the outputs depend on the present state and the inputs.

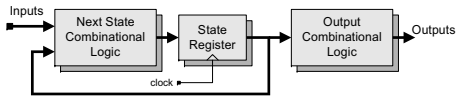
- Moore Machine: The next state depends on the present state and the inputs, but the output depends on only the present state.

FINITE STATE MACHINES (Cont.)

Mealy machine



Moore machine



STATE DIAGRAMS
PATTERN DETECT EXAMPLE

- Suppose we want a sequential system that has the following behaviour

  Input: $x(t) \in \{0, 1\}$

  Output: $z(t) \in \{0, 1\}$

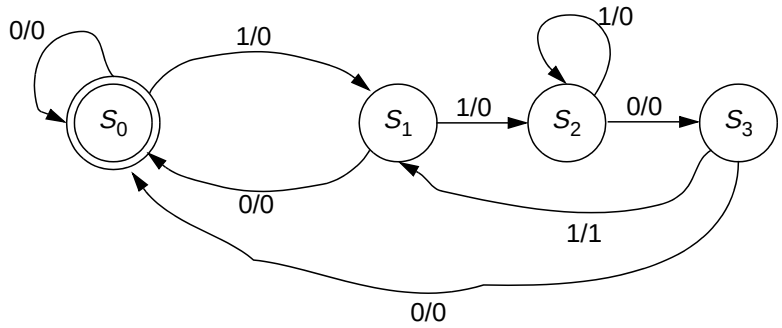
  Function: $z(t) = \begin{cases} \mathbf{1} & \text{if } x(t-3, t) = \mathbf{1101} \\ \mathbf{0} & \text{otherwise} \end{cases}$

  - Effectively, the system should output a **1** when the last set of four inputs have been **1101**.
  - For instance, the following output $z(t)$ is obtained for the input $x(t)$

t	0123456789...
$x(t)$	1001001001001**1**01**1**01**1**01001**1**01001
$z(t)$	???0000000000001**0**000**1**001**0**0000**1**000

STATE DIAGRAMS
PATTERN DETECT EXAMPLE

- The following state diagram gives the behaviour of the desired 1101 pattern detector.
  - Consider S_0 to be the initial state, S_1 when first symbol detected (**1**), S_2 when subpattern **11** detected, and S_3 when subpattern **110** detected.



STATE TABLES
INTRODUCTION

- State tables also express a systems behaviour and consists of
  - Present state
    - The present state of the system, typically given in binary encoded form or with S_k. So, a state of S_5 in our state diagram with 10 states would be represented as 0101 since we require 4 bits.
  - Inputs
    - Whatever external inputs used to cause the state transitions.
  - Next state
    - The next state, generally in binary encoded form.
  - Outputs
    - Whatever outputs, other then the state, for the system. Note that there would be no outputs in a Moore machine.

STATE TABLES

PATTERN DETECT EXAMPLE

- If we consider the pattern detection example previously discussed, the following would be the state table.

Present State	P_1	P_0	Input X	Next State	N_1	N_0	Output Z
S_0 or	0	0	0	S_0 or	0	0	0
S_0 or	0	0	1	S_1 or	0	1	0
S_1 or	0	1	0	S_0 or	0	0	0
S_1 or	0	1	1	S_2 or	1	0	0
S_2 or	1	0	0	S_3 or	1	1	0
S_2 or	1	0	1	S_2 or	1	0	0
S_3 or	1	1	0	S_0 or	0	0	0
S_3 or	1	1	1	S_1 or	0	1	1

STATE TABLES

TRANSLATE TO DIAGRAM

- If given a state table, the state diagram can be developed as follows.
  - Determine the number of states in the table and draw a state circle corresponding to each one.
    - Label the circle with the state name for a Mealy machine.
    - Label the circle with the state name/output for a Moore machine.
  - For each row in the table, identify the present state circle and draw a directed arc to the next state circle.
    - Label the arc with the input/output pair for a Mealy machine.
    - Label the arc with the input for a Moore machine.

SEQ. CIRCUITS

FROM STATE TABLE

- The procedure for developing a logic circuit from a state table is the same as with a regular truth table.
  - Generate Boolean functions for
    - each external outputs using external inputs and present state bits
    - each next state bit using external inputs and present state bits
  - Use Boolean algebra, Karnaugh maps, etc. as normal to simplify.
  - Draw a register for each state bit.
  - Draw logic diagram components connecting external outputs to external inputs and outputs of state bit registers (which have the present state).
  - Draw logic diagram components connecting inputs of state bits (for next state) to the external inputs and outputs of state bit registers (which have the present state).

SEQ. CIRCUITS

PATTERN DETECT EXAMPLE

- Following the procedure outlined, Boolean functions for the pattern detector state table can be formed using Karnaugh maps as follows.

X \ P_1P_0	00	01	11	10
0	0	0	1	0
1	1	1	0	0

N_1

X \ P_1P_0	00	01	11	10
0	0	1	0	0
1	1	0	1	0

N_0

X \ P_1P_0	00	01	11	10
0	0	0	0	0
1	0	0	1	0

Z

$$N_1 = X\overline{P_1} + \overline{X}P_1P_0$$

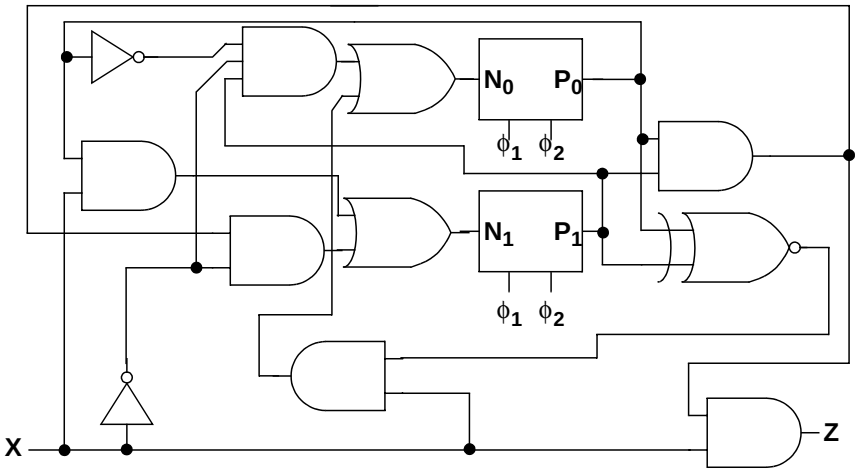
$$N_0 = \overline{X}\,\overline{P_1}P_0 + XP_1P_0 + X\overline{P_1}\,\overline{P_0} = \overline{X}\,\overline{P_1}P_0 + X(\overline{P_1 \oplus P_0})$$

$$Z = XP_1P_0$$

SEQ. CIRCUITS

PATTERN DETECT EXAMPLE

- The following logic circuit implements the pattern detect example.



FSM EXAMPLES

EXAMPLE #2

- A sequential circuit is defined by the following Boolean functions with input X, present states P_0, P_1, and P_2, and next states N_0, N_1, and N_2.
  - $N_2 = X(P_1 \oplus P_0) + \bar{X}\overline{(P_1 \oplus P_0)}$
  - $N_1 = P_2$
  - $N_0 = P_1$
  - $Z = XP_1P_2$
- Derive the state table.
- Derive the state diagram.

FSM EXAMPLES

EXAMPLE #2

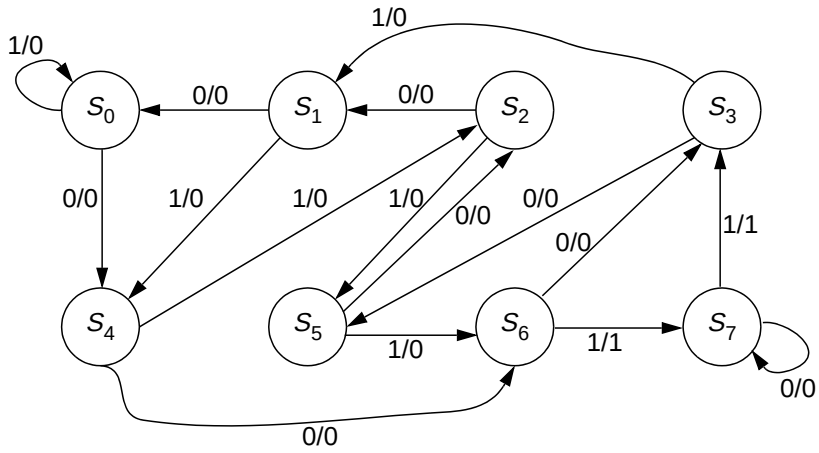
- The state table is formed as follows.

Present State P_2	P_1	P_0	Input X	Next State N_2	N_1	N_0	Output Z
0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	1	0	0	0
0	1	0	0	0	0	1	0
0	1	0	1	1	0	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	0
1	0	0	0	1	1	0	0
1	0	0	1	0	1	0	0
1	0	1	0	0	1	0	0
1	0	1	1	1	1	0	0
1	1	0	0	0	1	1	0
1	1	0	1	1	1	1	1
1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1

FSM EXAMPLES

EXAMPLE #2

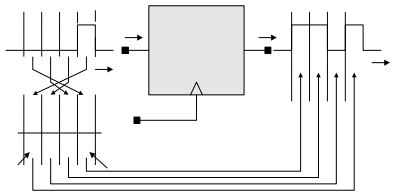
- The state diagram can be drawn as follows.



MEALY FINITE STATE MACHINE - EXAMPLE

A serially-transmitted BCD (8421 code) word is to be converted into an Excess-3 code. An Excess-3 code word is obtained by adding *3* to the decimal value and taking the binary equivalent. Excess-3 code is self-complementing [Wakerly, p. 80], i.e. the *9*'s complement of a code word is obtained by complementing the bits of the word.

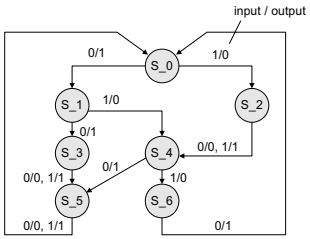
Decimal Digit	8-4-2-1 Code (BCD)	Excess-3 Code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100



MEALY FINITE STATE MACHINE - EXAMPLE (Cont.)

The serial code converter is described by the state transition graph of a Mealy FSM.

State Transition Graph



Next State/OutputTable		
state	next state/output	
	input	
	0	1
S_0	S_1 / 1	S_2 / 0
S_1	S_3 / 1	S_4 / 0
S_2	S_4 / 0	S_4 / 1
S_3	S_5 / 0	S_5 / 1
S_4	S_5 / 1	S_6 / 0
S_5	S_0 / 0	S_0 / 1
S_6	S_0 / 1	- / -

- The vertices of the state transition graph of a Mealy machine are labeled with the states.
- The branches are labeled with (1) the input that causes a transition to the indicated next state, and (2) with the output that is asserted in the present state for that input.
- The state transition is synchronized to a clock.
- The state table summarizes the machine's behavior in tabular format.

DESIGN OF A FINITE STATE MACHINE - EXAMPLE (Cont.)

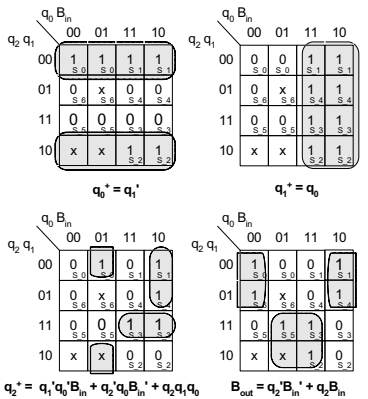
To design a D-type flip-flop realization of a FSM having the behavior described by a state transition graph, (1) select a state code, (2) encode the state table, (3) develop Boolean equations describing the input of a D-type flip-flop, and (4) using K-maps, optimize the Boolean equations.

Next State/Output Table		
state	next state/output	
	input	
	0	1
S_0	S_1 / 1	S_2 / 0
S_1	S_3 / 1	S_4 / 0
S_2	S_4 / 0	S_4 / 1
S_3	S_5 / 0	S_5 / 1
S_4	S_5 / 1	S_6 / 0
S_5	S_0 / 0	S_0 / 1
S_6	S_0 / 1	- / -

State Assigment			
		q_0	
q_2	q_1	0	1
0	0	S_0	S_1
0	1	S_6	S_4
1	0		S_2
1	1	S_5	S_3

Encoded Next state/ Output Table					
	state	next state		output	
	$q_2\ q_1\ q_0$	$q_2^+\ q_1^+\ q_0^+$			
		input		input	
		0	1	0	1
S_0	000	001	101	1	0
S_1	001	111	011	1	0
S_2	101	011	011	0	1
S_3	111	110	110	0	1
S_4	011	110	010	1	0
S_5	110	000	000	0	1
S_6	010	000	-	1	-
	100	-	-	-	-

DESIGN OF A FINITE STATE MACHINE - EXAMPLE (Cont.)



Note: We will optimize the equations individually. In general - this does not necessarily produce the optimal (area, speed) realization of the logic. We'll address this when we consider synthesis.

$$q_2^+ = q_1'q_0'B_{in} + q_2'q_0B_{in}' + q_2q_1q_0$$

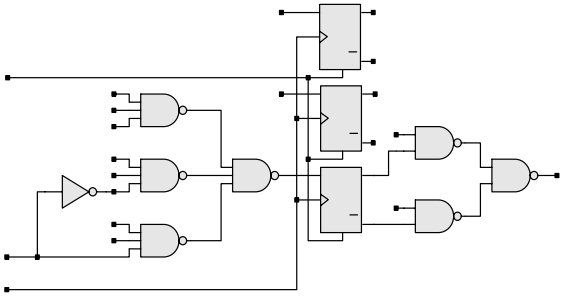
$$\overline{q_2^+} = \overline{q_1'q_0'B_{in} + q_2'q_0B_{in}' + q_2q_1q_0}$$

$$\overline{q_2^+} = \overline{q_1'q_0'B_{in}}\ \overline{q_2'q_0B_{in}'}\ \overline{q_2q_1q_0}$$

$$q_2^+ = \overline{\overline{q_1'q_0'B_{in}}\ \overline{q_2'q_0B_{in}'}\ \overline{q_2q_1q_0}}$$

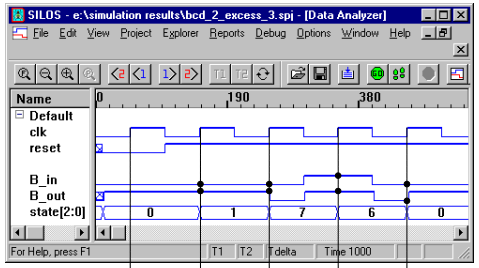
DESIGN OF A FINITE STATE MACHINE - EXAMPLE (Cont.)

Realization of the sequential BCD-to-Excess-3 code converter (Mealy machine):



DESIGN OF A FINITE STATE MACHINE - EXAMPLE (Cont.)

Simulation results for Mealy machine:



B_in	0	0	1	0
B_out	1	1	1	0

Note: s3 = 111_2