

Sequential Logic – Introduction to Computer (計算機概論)



Winston H. Hsu (徐宏民)
National Taiwan University, Taipei

October 3/17, 2022

Office: R512, CSIE Building

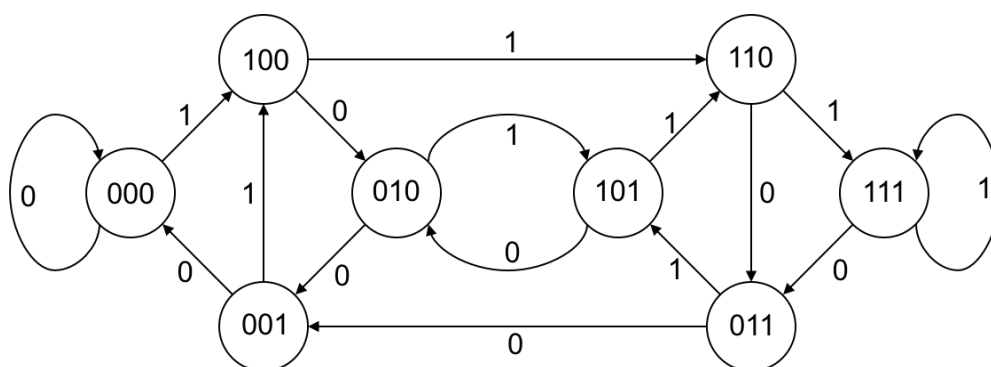
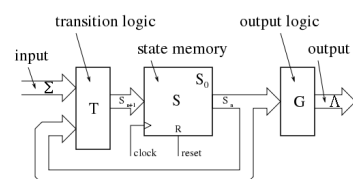
Communication and Multimedia Lab (通訊與多媒體實驗室)

<http://winstonhsu.info>

The majority of the slides are from the textbook

Finite States – State Diagram to Keep Track of States

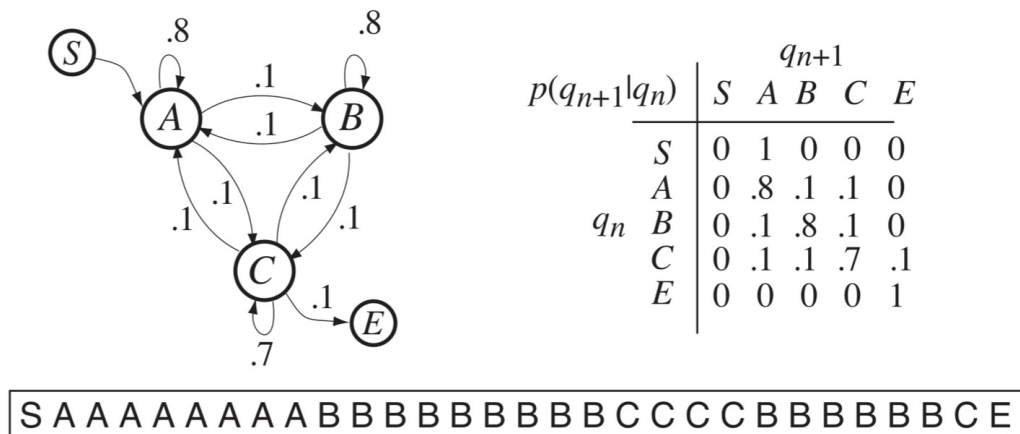
- For example, shift register
 - input value shown on transition arcs
 - output values shown within state node



Moore Finite State Machine (FSM)

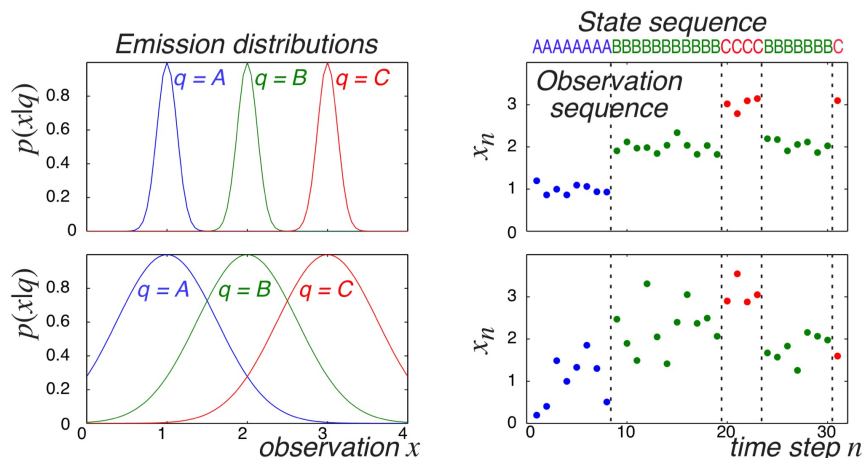
Markov Model

- A (first order) Markov model is a finite-state system whose behavior depends only on the current state
 - e.g. generative Markov model



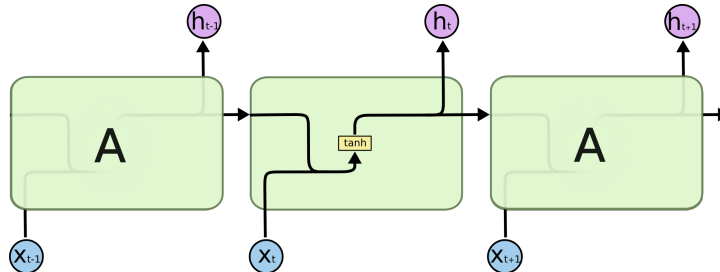
Hidden Markov Model (HMM)

- Markov model where state sequence $Q = \{q_n\}$ is not directly observable (= 'hidden')
- But, observations X do depend on Q
 - x_n is rv that depends on current state: $p(x|q)$

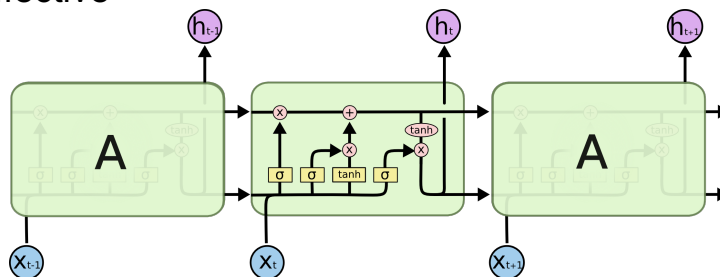


Recurrent Neural Networks (RNN) for Learning Sequential Behaviors – States in a (Long) Vector

- Standard RNN contains a single layer



- Long Short Term Memory (LSTM) contains four interacting layers and is more effective



<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

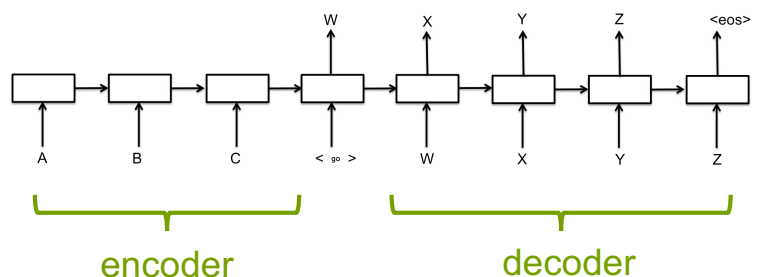
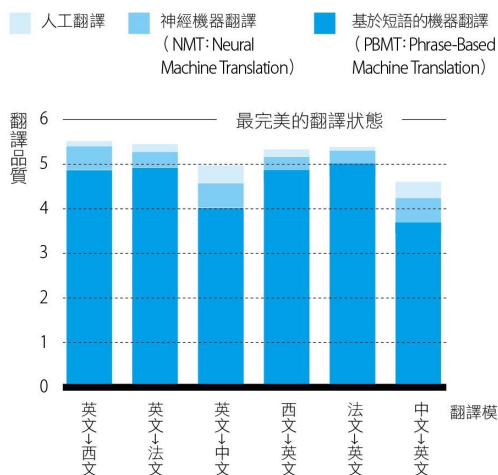
5

IC, Fall 2022 – Winston Hsu

Sequence to Sequence Model

- 「一次翻譯一整個句子，而非把句子切成數個字詞來翻譯」 – i.e.,
- “Google’s Translation App Is About To Get Much Better”
– Times Magazine

Google 翻譯整合神經網絡，翻譯品質接近人工筆譯

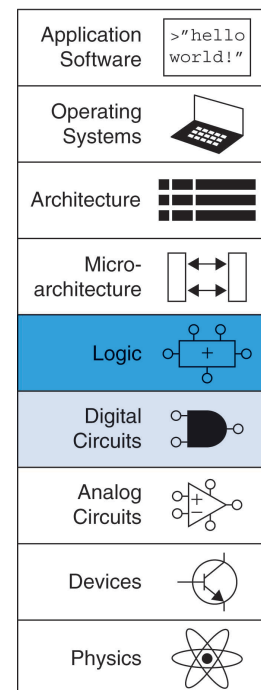


6

IC, Fall 2022 – Winston Hsu

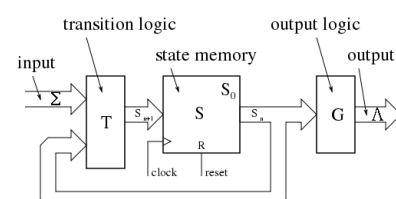
Topics

- Introduction
- Latches and Flip-Flops
- Synchronous Logic Design
- Finite State Machines



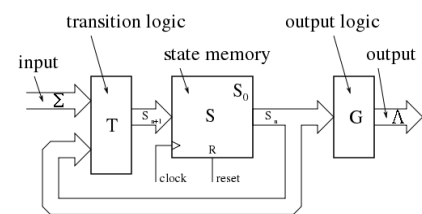
Introduction

- Outputs of sequential logic depend on current *and* prior input values – it has **memory**.
- Some definitions:
 - **State**: all the information about a circuit necessary to explain its future behavior
 - **Latches and flip-flops**: state elements that store one bit of state
 - **Synchronous sequential circuits**: combinational logic followed by a bank of flip-flops



Sequential Circuits

- Give sequence of events
- Have memory (short-term)
- Use feedback from output to input to store information

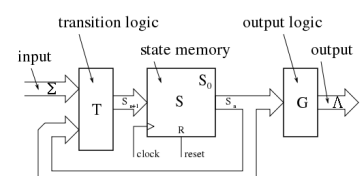


9

IC, Fall 2022 – Winston Hsu

State Elements

- The state of a circuit influences its future behavior
- State elements store state
 - Bistable circuit
 - SR Latch
 - D Latch
 - D Flip-flop

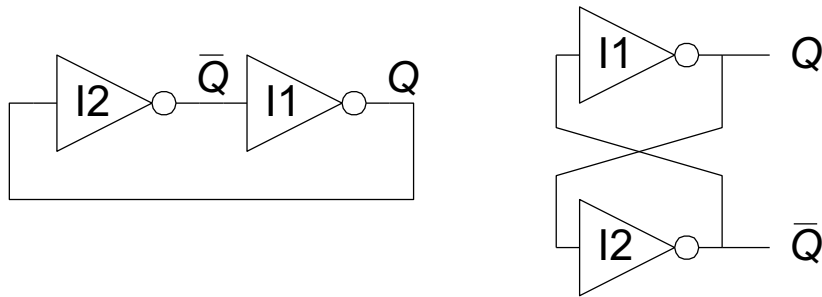


10

IC, Fall 2022 – Winston Hsu

Bistable Circuit

- Fundamental building block of other state elements
- Two outputs: Q , \bar{Q}
- No inputs

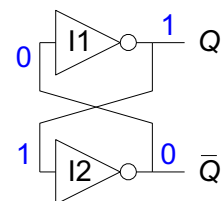
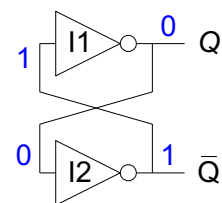


Bistable Circuit Analysis

- Consider the two possible cases:

– $Q = 0$:
then $Q' = 1$, $Q = 0$ (consistent)

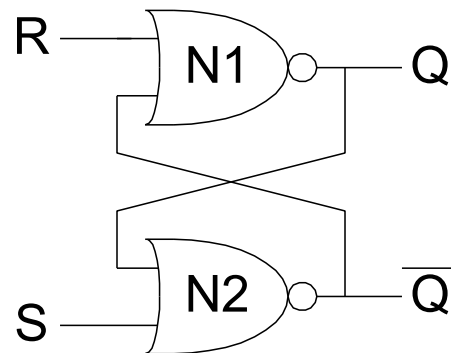
– $Q = 1$:
then $Q' = 0$, $Q = 1$ (consistent)



- Stores 1 bit of state in the state variable, Q (or Q')
- But there are **no inputs to control the state**

SR (Set/Reset) Latch

- SR Latch



- Consider the four possible cases:

- $S = 1, R = 0$
- $S = 0, R = 1$
- $S = 0, R = 0$
- $S = 1, R = 1$

NOR



$$Y = \overline{A + B}$$

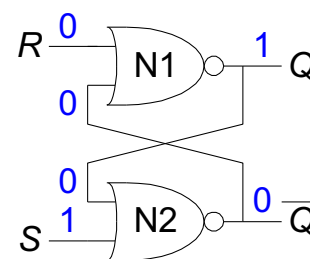
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

13

IC, Fall 2022 – Winston Hsu

SR Latch Analysis

- $S = 1, R = 0$:
then $Q = 1$ and $Q' = 0$



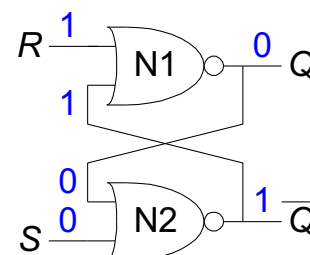
NOR



$$Y = \overline{A + B}$$

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

- $S = 0, R = 1$:
then $Q = 0$ and $Q' = 1$




14

IC, Fall 2022 – Winston Hsu

SR Latch Analysis

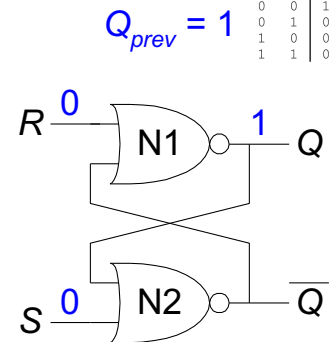
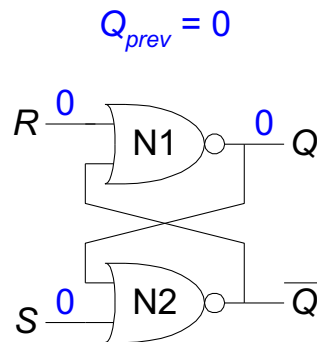
NOR



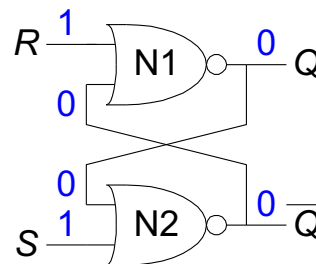
$$Y = \overline{A + B}$$

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

- $S = 0, R = 0$:
then $Q = Q_{prev}$



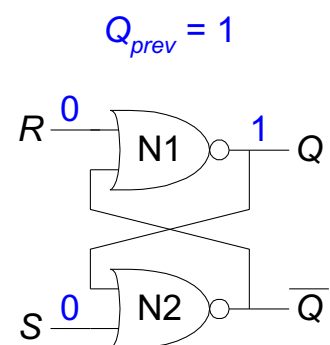
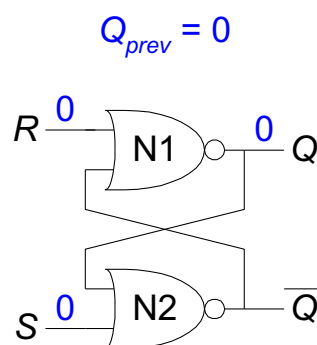
- $S = 1, R = 1$:
then $Q = 0, \bar{Q} = 0$



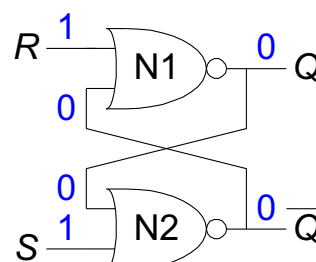
SR Latch Analysis

- $S = 0, R = 0$:
then $Q = Q_{prev}$

– **Memory!**



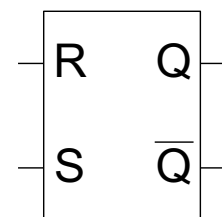
- $S = 1, R = 1$:
then $Q = 0, \bar{Q} = 0$
Invalid State
 $\bar{Q} \neq \text{NOT } Q$



SR Latch Symbol

- SR stands for Set/Reset Latch
 - Stores one bit of state (Q)
- Control what value is being stored with S , R inputs
 - **Set:** Make the output 1
($S = 1, R = 0, Q = 1$)
 - **Reset:** Make the output 0
($S = 0, R = 1, Q = 0$)
- **Must do something to avoid invalid state (when $S = R = 1$)**

SR Latch Symbol



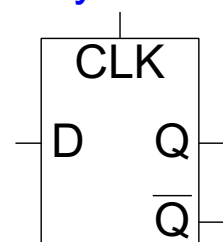
17

IC, Fall 2022 – Winston Hsu

D Latch

- Two inputs: CLK , D
 - **CLK:** controls *when* the output changes
 - **D** (the data input): controls *what* the output changes to
- Function
 - When $CLK = 1$,
 D passes through to Q (transparent)
 - When $CLK = 0$,
 Q holds its previous value (opaque)
- Avoids invalid case when
 $Q \neq \text{NOT } \overline{Q}$

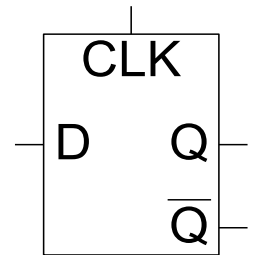
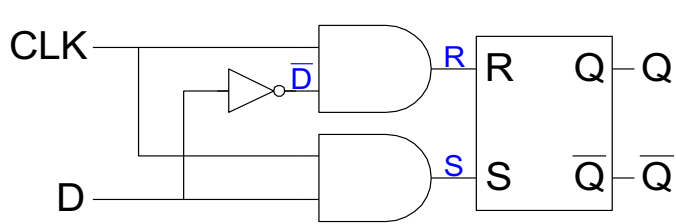
D Latch Symbol



18

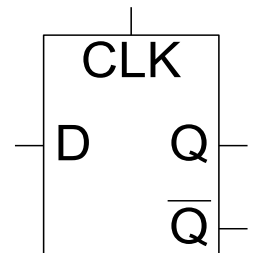
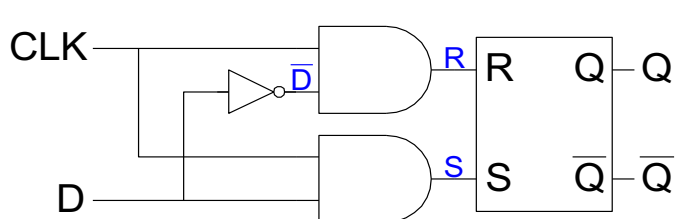
IC, Fall 2022 – Winston Hsu

D Latch Internal Circuit



CLK	D	\overline{D}	S	R	Q	\overline{Q}
0	X					
1	0					
1	1					

D Latch Internal Circuit



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

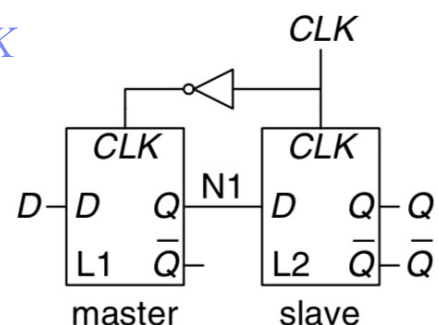
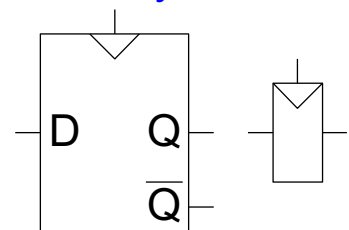
Flip-Flop

- A small and useful sequential circuit
 - e.g., synchronization with clock
- Abstraction that remembers one bit
- Basis of important computer components for
 - register
 - memory
 - counter
- There are several flavors

D Flip-Flop

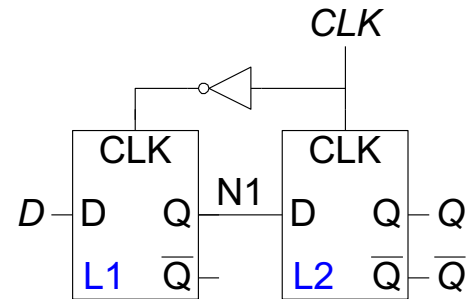
- **Inputs:** CLK , D
- **Function**
 - Samples D on rising edge of CLK
 - When CLK rises from 0 to 1, D passes through to Q
 - Otherwise, Q holds its previous value
 - Q changes only on rising edge of CLK
- Called *edge-triggered*
- Activated on the clock edge

D Flip-Flop Symbols

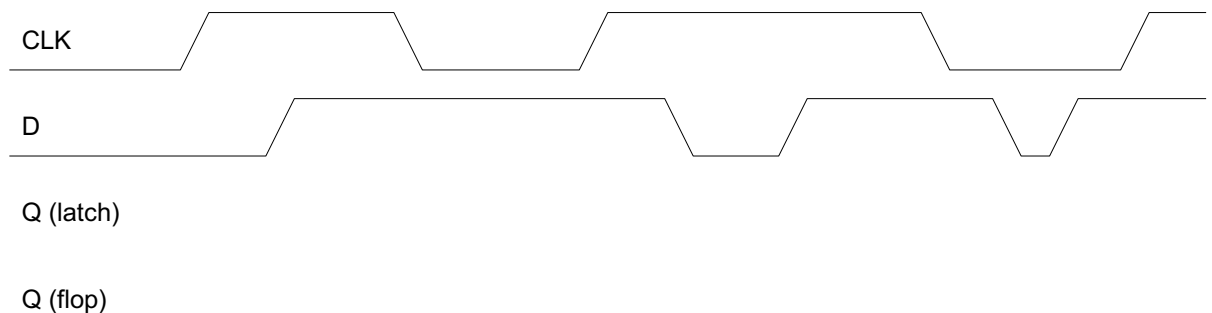
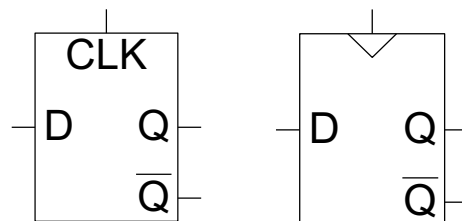


D Flip-Flop Internal Circuit

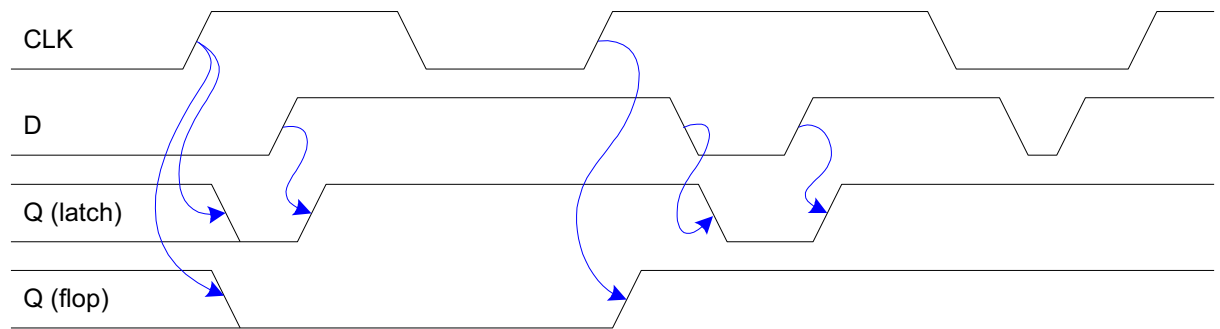
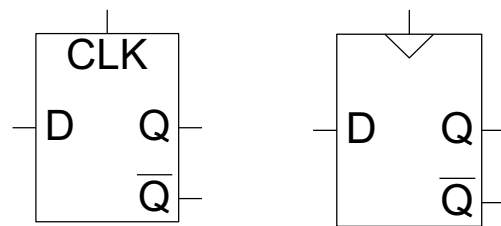
- Two back-to-back latches (L1 and L2) controlled by complementary clocks
- When $CLK = 0$
 - L1 is transparent
 - L2 is opaque
 - D passes through to N1 (node)
- When $CLK = 1$
 - L2 is transparent
 - L1 is opaque
 - N1 passes through to Q
- Thus, on the edge of the clock (when CLK rises from 0 \rightarrow 1)
 - D passes through to Q



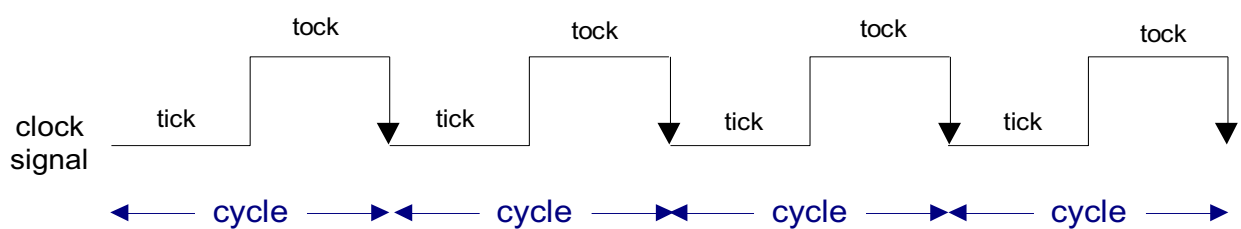
D Latch vs. D Flip-Flop



D Latch vs. D Flip-Flop



The Clock



- In our jargon, a clock cycle = *tick*-phase (low), followed by a *tock*-phase (high)
- In real hardware, the clock is implemented by an oscillator

How Much does it Hertz?

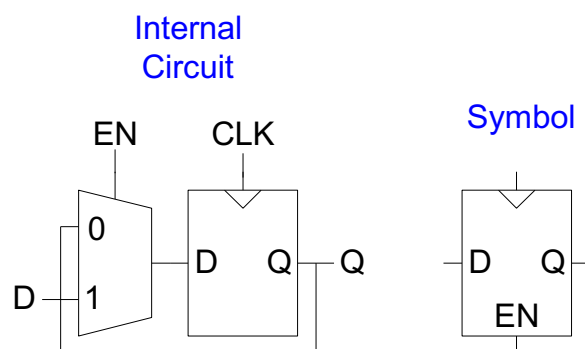
- Frequency is inverse of cycle time.
 - Expressed in hertz.
 - Frequency of 1 Hz means that there is 1 cycle per second.
 - 1 kilohertz (kHz) means 1000 cycles/sec.
 - 1 megahertz (MHz) means 1 million cycles/sec.
 - 1 gigahertz (GHz) means 1 billion cycles/sec.
 - 1 terahertz (THz) means 1 trillion cycles/sec.



Heinrich Rudolf Hertz
(1857-1894)

Enabled Flip-Flops

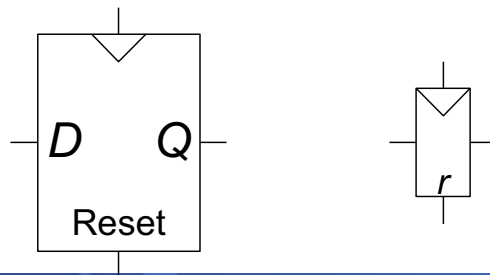
- **Inputs:** CLK , D , EN
 - The enable input (EN) controls when new data (D) is stored
- **Function**
 - $EN = 1$: D passes through to Q on the clock edge
 - $EN = 0$: the flip-flop retains its previous state



Resettable Flip-Flops

- **Inputs:** CLK , D , $Reset$
- **Function:**
 - **Reset = 1:** Q is forced to 0
 - **Reset = 0:** flip-flop behaves as ordinary D flip-flop

Symbols

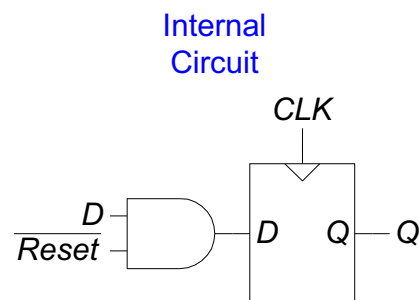


Resettable Flip-Flops

- Two types:
 - **Synchronous:** resets at the clock edge only
 - **Asynchronous:** resets immediately when $Reset = 1$
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop
- Synchronously resettable flip-flop?

Resettable Flip-Flops

- Two types:
 - Synchronous**: resets at the clock edge only
 - Asynchronous**: resets immediately when Reset = 1
- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop
- Synchronously resettable flip-flop?



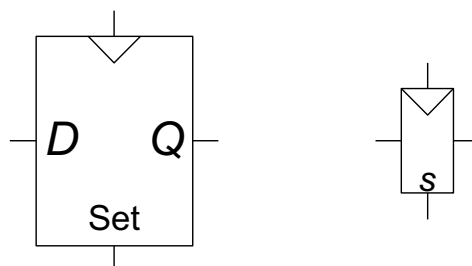
31

IC, Fall 2022 – Winston Hsu

Settable Flip-Flops

- Inputs:** *CLK*, *D*, *Set*
- Function:**
 - Set = 1**: *Q* is set to 1
 - Set = 0**: the flip-flop behaves as ordinary D flip-flop

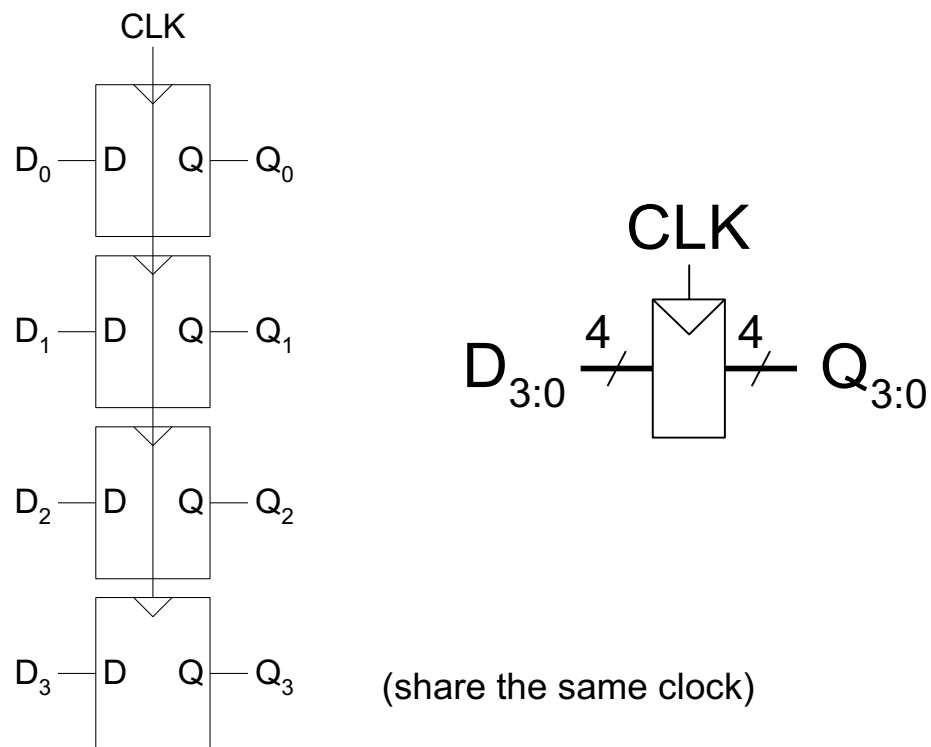
Symbols



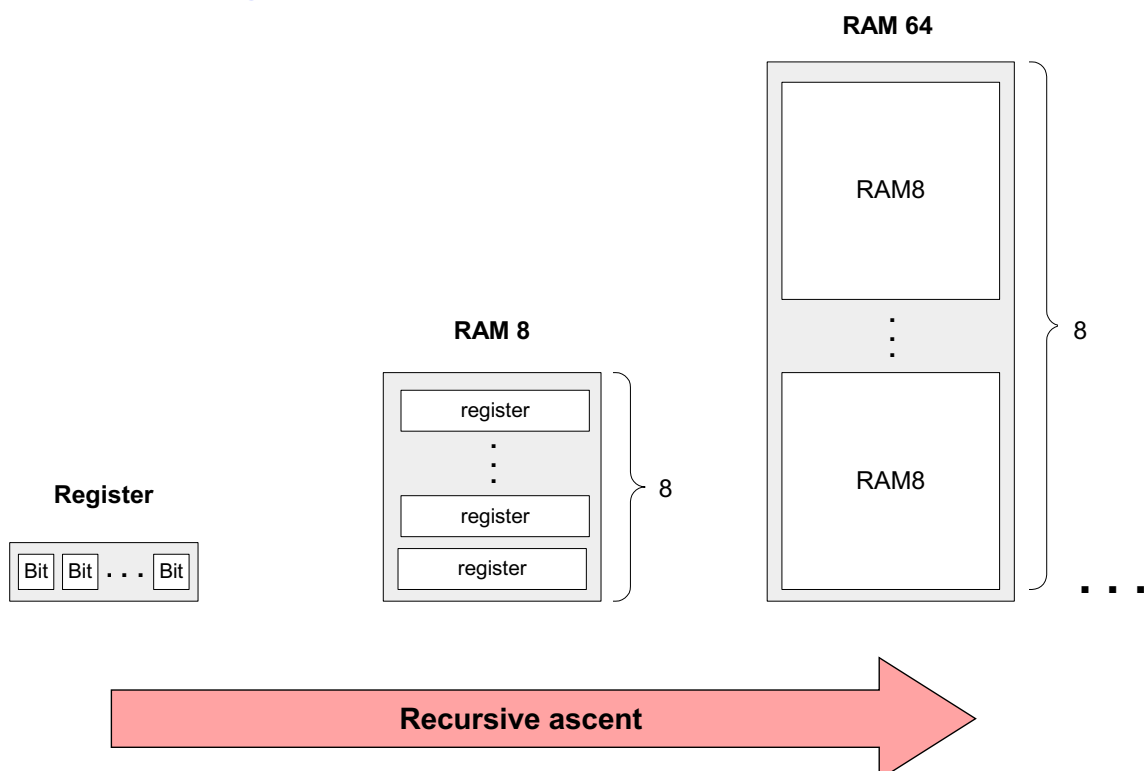
32

IC, Fall 2022 – Winston Hsu

Registers (4 Bit)



RAM Anatomy



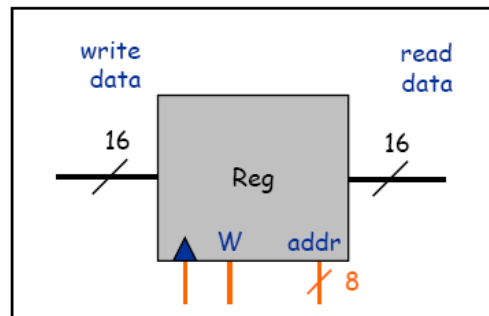
Register file interface

$n \times k$ register file.

- Bank of n registers; each stores k bits.
- Read and write information to *one* of n registers.
 - $\log_2 n$ address inputs specifies which one
- Addressed bits always appear on output.
- If write enable and clock are asserted, k input bits are copied into addressed register.

Examples.

- TOY registers: $n = 16$, $k = 16$.
- TOY main memory: $n = 256$, $k = 16$.
- Real computer: $n = 256$ million, $k = 32$.
 - 1 GB memory
 - 1 byte = 8 bits

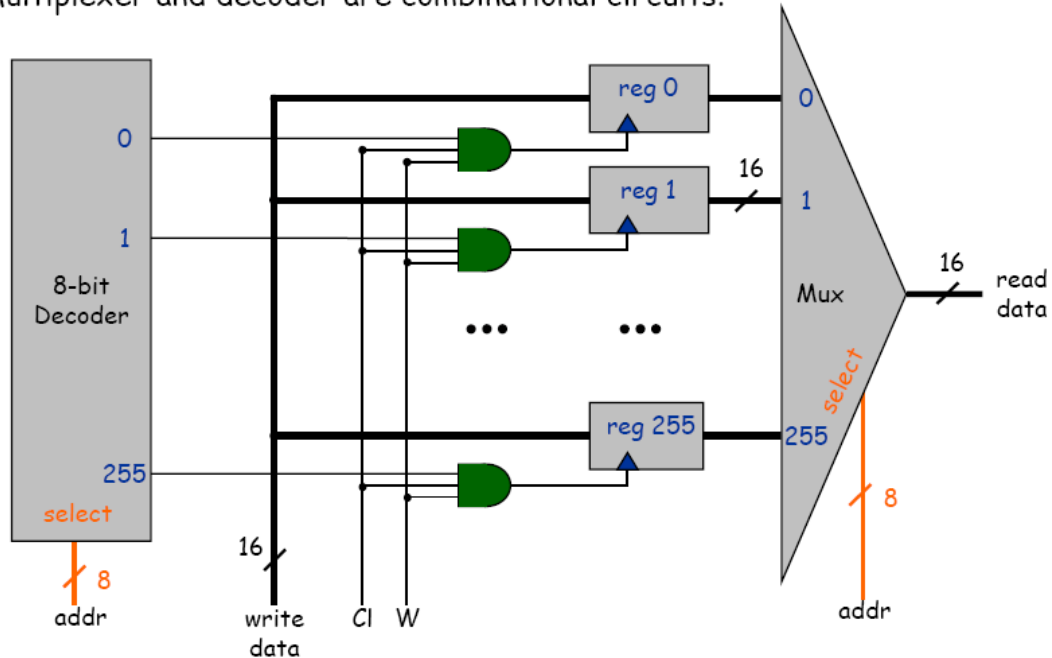


256 x 16 Register File Interface

Register file implementation

Implementation example: TOY main memory.

- Use 256 16-bit registers.
- Multiplexer and decoder are combinational circuits.

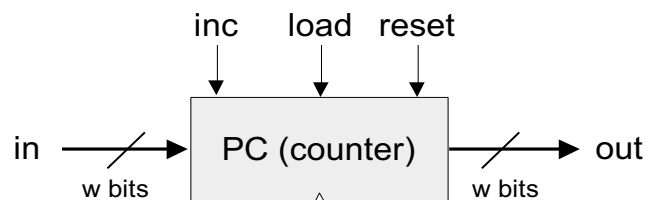


37

IC, Fall 2022 – Winston Hsu

Counter

- Needed: a storage device that can:
 - (a) set its state to some base value
 - (b) increment the state in every clock cycle
 - (c) maintain its state (stop incrementing) over clock cycles
 - (d) reset its state



```

If reset(t-1) then out(t)=0
else if load(t-1) then out(t)=in(t-1)
else if inc(t-1) then out(t)=out(t-1)+1
else out(t)=out(t-1)
    
```

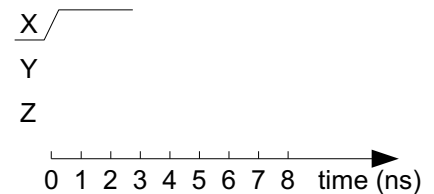
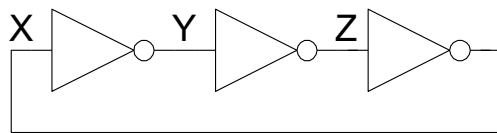
- Typical function: *program counter*
- Implementation: register chip + some combinational logic.

38

IC, Fall 2022 – Winston Hsu

Sequential Logic

- Sequential circuits: all circuits that aren't combinational
- A problematic circuit:
 - assuming 1 ns delay

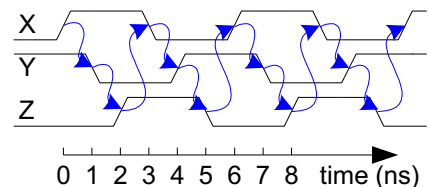
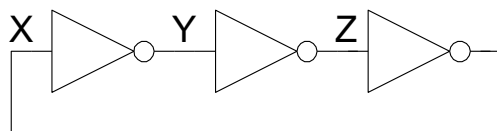


39

IC, Fall 2022 – Winston Hsu

Sequential Logic

- Sequential circuits: all circuits that aren't combinational
- A problematic circuit:



- No inputs and 1-3 outputs
- Astable circuit, oscillates
- Period depends on inverter delay
- It has a **cyclic path**: output fed back to input

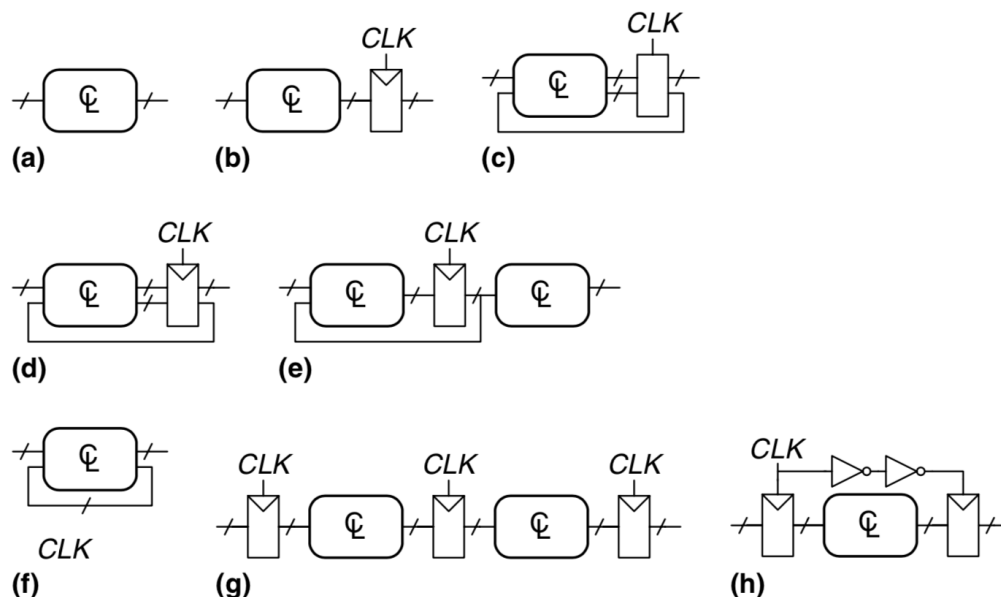
40

IC, Fall 2022 – Winston Hsu

Synchronous Sequential Logic Design

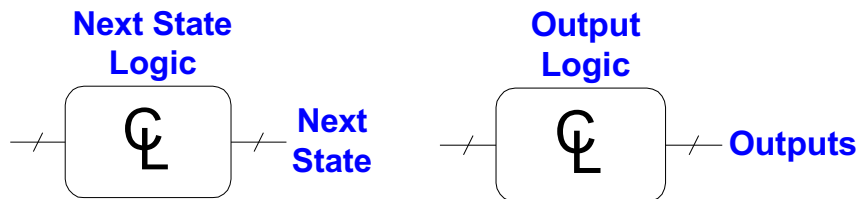
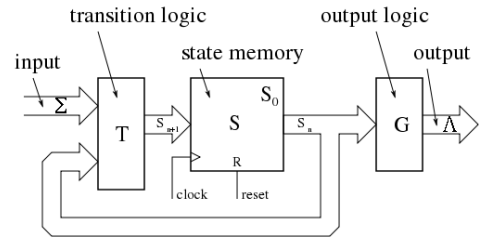
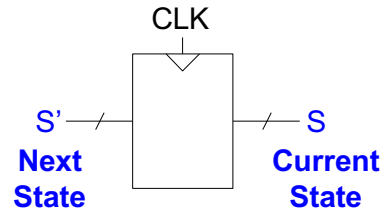
- Breaks cyclic paths by **inserting registers**
- Registers contain **state** of the system
- **State changes at clock edge**: system **synchronized** to the clock
- **Rules** of synchronous sequential circuit composition:
 - Every circuit element is **either** a *register* or a *combinational circuit*
 - At least one circuit element is a register
 - All registers receive the same clock signal
 - Every cyclic path contains at least one register
- Two common synchronous sequential circuits
 - Finite State Machines (FSMs)
 - Pipelines

Example – Which Are Synchronous Sequential Circuits?



Finite State Machine (FSM)

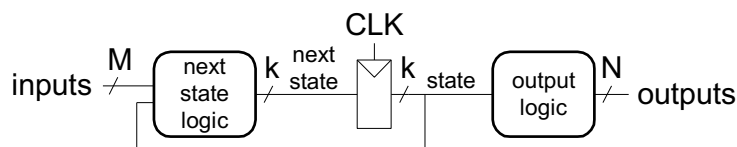
- Consists of:
 - State register**
 - Stores current state
 - Loads next state at clock edge
 - Combinational logic**
 - Computes the next state
 - Computes the outputs



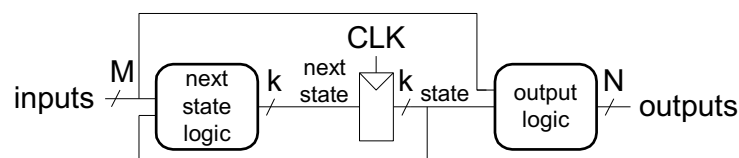
Finite State Machine (FSM)

- Next state determined by current state and inputs
- Two types of finite state machines differ in output logic:
 - Moore FSM:** outputs depend only on current state
 - Mealy FSM:** outputs depend on current state *and* inputs

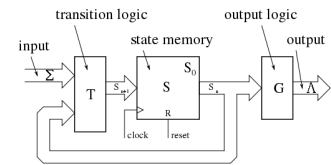
Moore FSM



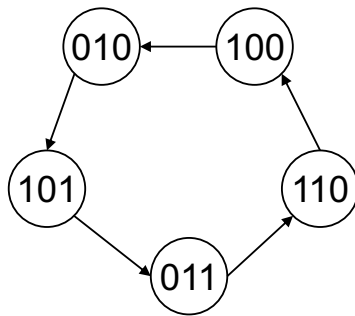
Mealy FSM



FSM Example (Simple)



- Implement simple count sequence: 000, 010, 011, 101, 110
- Derive the state transition table from the state transition diagram



Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	x	x	x
0	0	1	x	x	x
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	x	x	x

note the don't care conditions that arise from the unused state codes

Don't cares in FSMs (cont'd)

- Synthesize logic for next state functions derive input equations for flip-flops

C^+		C			
A	X	1	1	0	
	X	1	X	0	
		B			

B^+		C		
A	X	0	0	1
	X	1	X	1
		B		

A^+		C		
A	X	1	0	0
	X	0	X	1
		B		

$$C^+ = B$$

$$B^+ = A + B' C$$

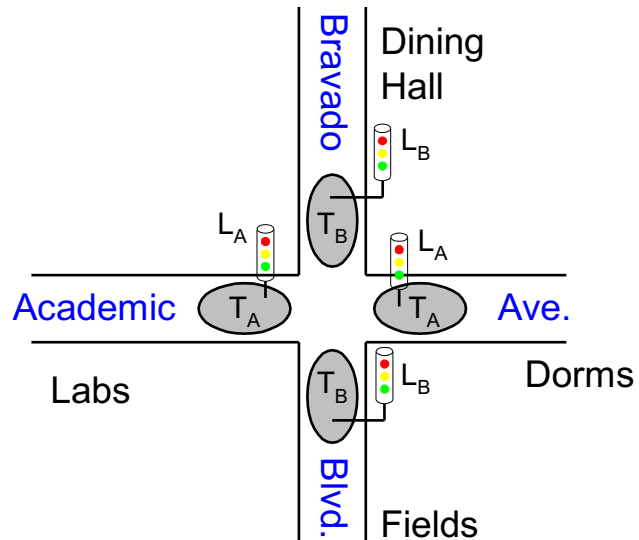
$$A^+ = A' C' + AC$$

FSM Example

- Traffic light controller

(input) — Traffic sensors: T_A , T_B (TRUE when there's traffic)

(output) — Lights: L_A , L_B

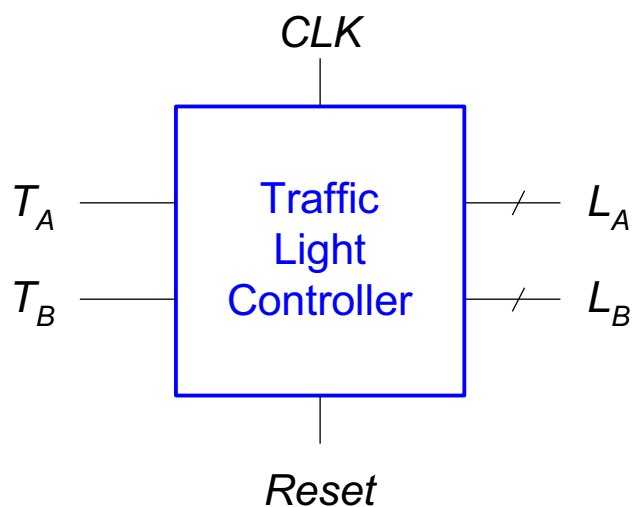
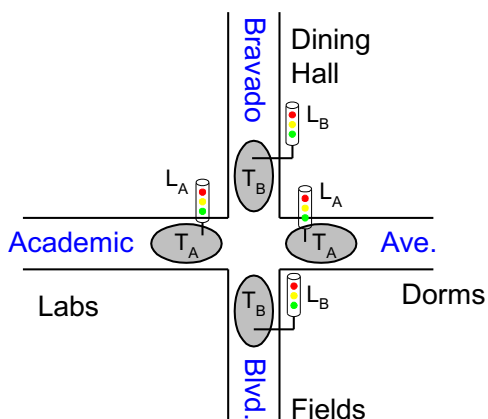


47

IC, Fall 2022 – Winston Hsu

FSM Black Box

- Inputs: CLK , $Reset$, T_A , T_B
- Outputs: L_A , L_B

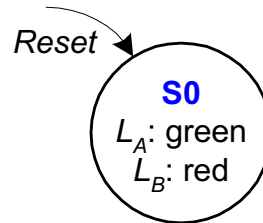
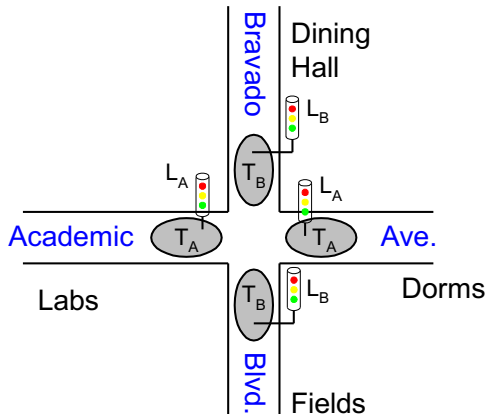


48

IC, Fall 2022 – Winston Hsu

FSM State Transition Diagram

- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs

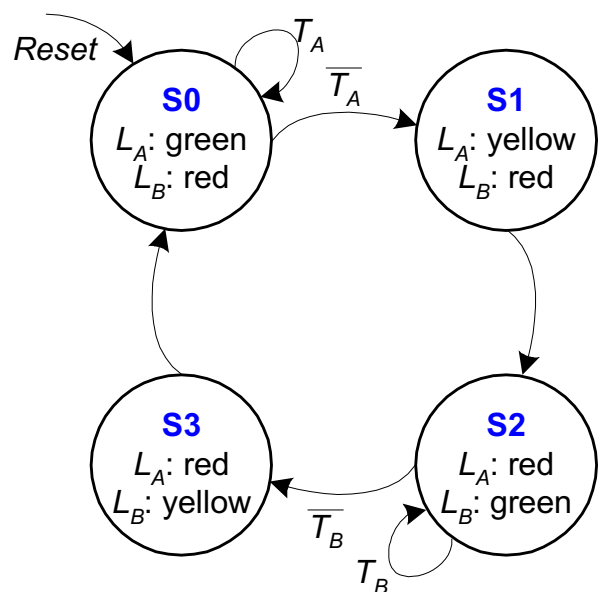
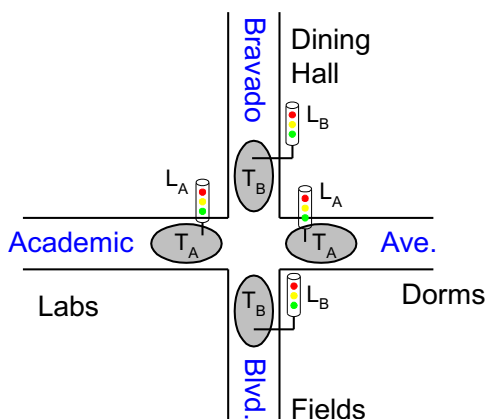


49

IC, Fall 2022 – Winston Hsu

FSM State Transition Diagram

- **Moore FSM:** outputs labeled in each state
- **States:** Circles
- **Transitions:** Arcs

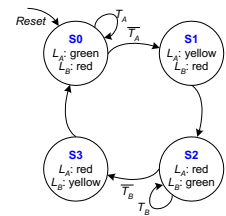


50

IC, Fall 2022 – Winston Hsu

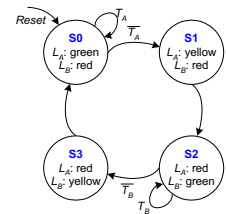
FSM State Transition Table

Current State S	Inputs T_A T_B		Next State S'
S0	0	X	
S0	1	X	
S1	X	X	
S2	X	0	
S2	X	1	
S3	X	X	



FSM State Transition Table

Current State S	Inputs T_A T_B		Next State S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0



FSM Encoded State Transition Table

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X		
0	0	1	X		
0	1	X	X		
1	0	X	0		
1	0	X	1		
1	1	X	X		

State	Encoding
S0	00
S1	01
S2	10
S3	11

FSM Encoded State Transition Table

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

State	Encoding
S0	00
S1	01
S2	10
S3	11

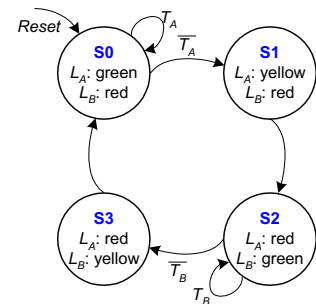
$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S_1} \overline{S_0} \overline{T_A} + S_1 \overline{S_0} \overline{T_B}$$

FSM Output Table

Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0				
0	1				
1	0				
1	1				

Output	Encoding
green	00
yellow	01
red	10

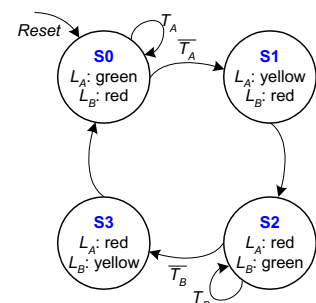


FSM Output Table

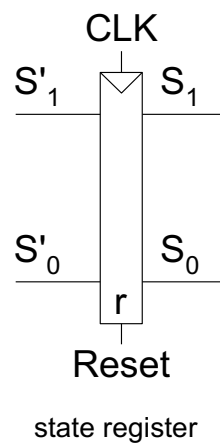
Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

$$\begin{aligned}
 L_{A1} &= S_1 \\
 L_{A0} &= \overline{S_1} S_0 \\
 L_{B1} &= \overline{S_1} \\
 L_{B0} &= S_1 S_0
 \end{aligned}$$

Output	Encoding
green	00
yellow	01
red	10



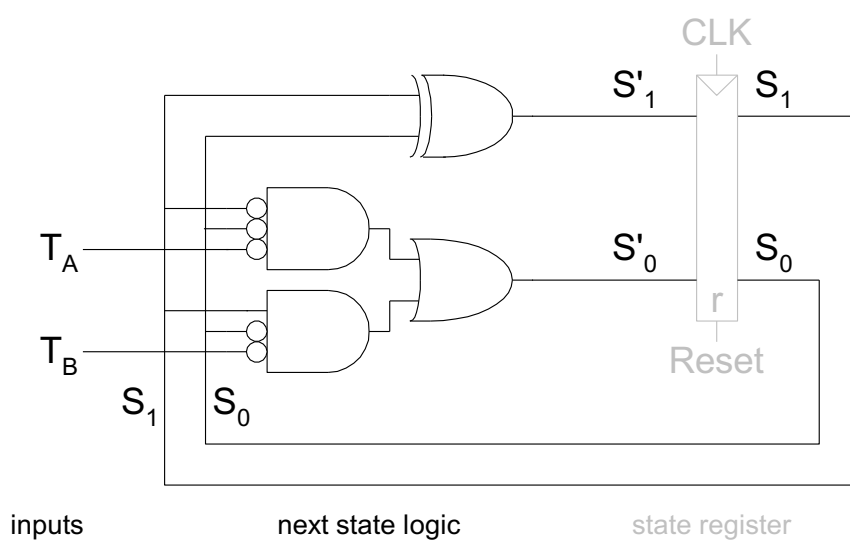
FSM Schematic: State Register



57

IC, Fall 2022 – Winston Hsu

FSM Schematic: Next State Logic



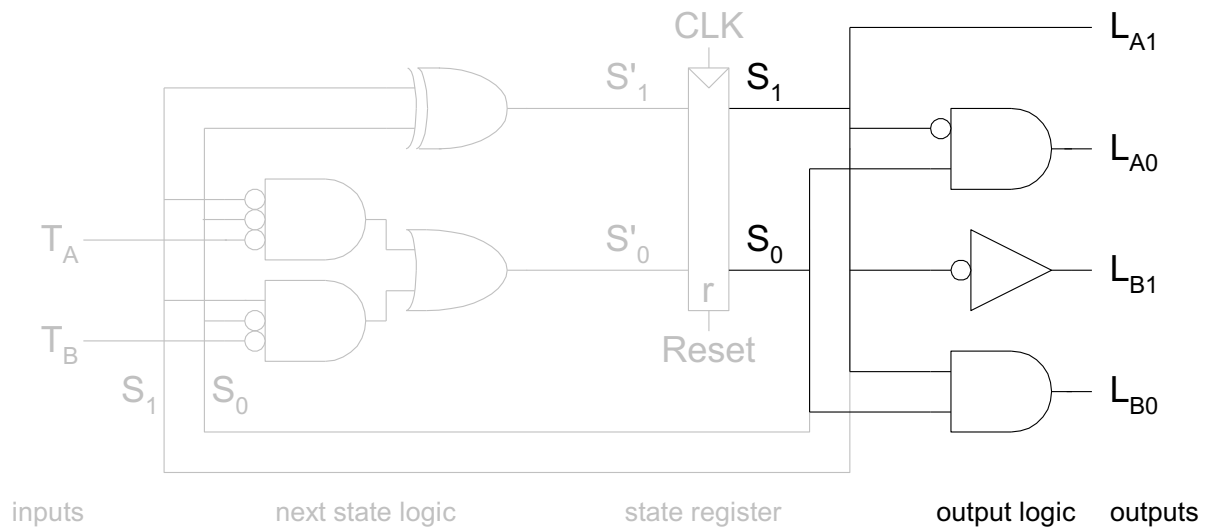
$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S_1} \overline{S_0} \overline{T_A} + S_1 \overline{S_0} \overline{T_B}$$

58

IC, Fall 2022 – Winston Hsu

FSM Schematic: Output Logic



$$L_{A1} = S_1$$

$$L_{A0} = S_1 S_0$$

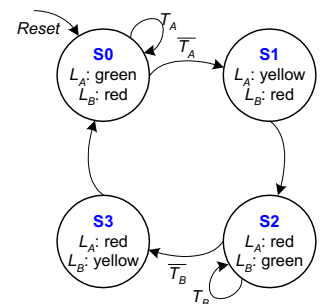
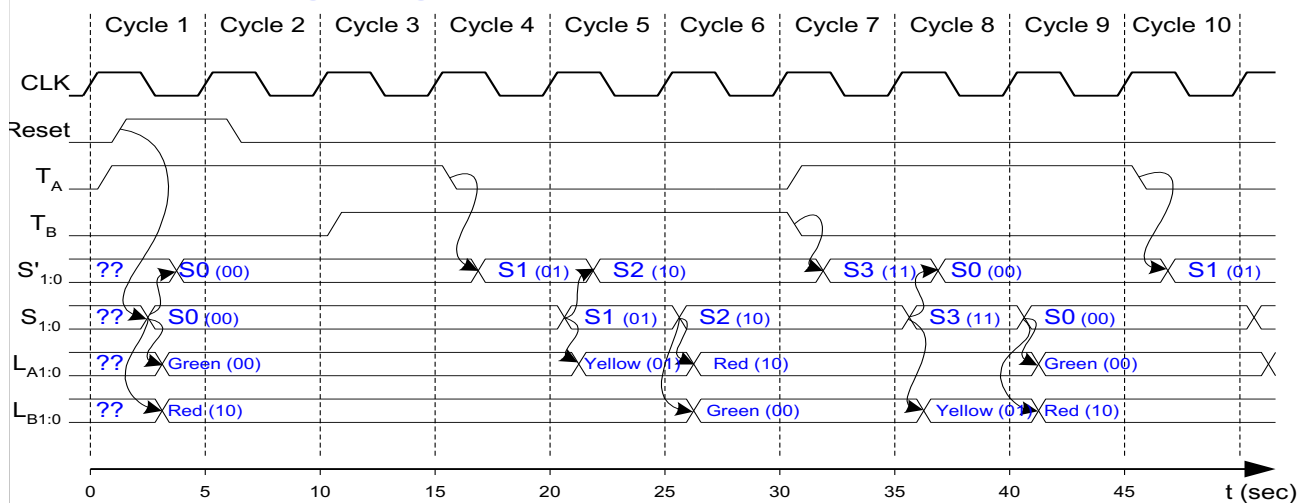
$$L_{B1} = S_1$$

$$L_{B0} = S_1 S_0$$

59

IC, Fall 2022 – Winston Hsu

FSM Timing Diagram



60

IC, Fall 2022 – Winston Hsu

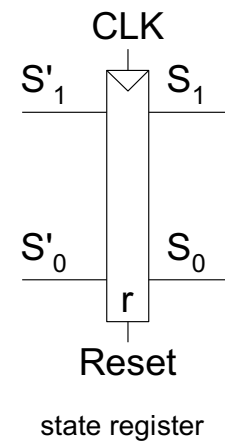
FSM State Encoding

- **Binary encoding:**

- i.e., for four states, 00, 01, 10, 11

- **One-hot encoding**

- One state bit per state
- Only one state bit HIGH at once
- i.e., for 4 states, 0001, 0010, 0100, 1000
- Requires more flip-flops
- Often next state and output logic is simpler



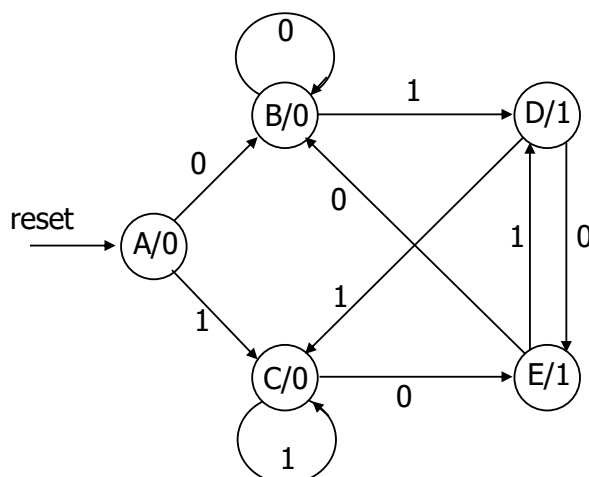
61

IC, Fall 2022 – Winston Hsu

Specifying outputs for a Moore machine

- Output is only function of state

- specify in state bubble in state diagram
- Example (what does it do?)



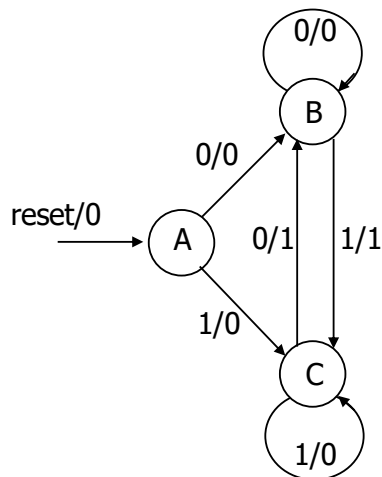
reset	input	current state	next state	output
1	–	–	A	
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	D	0
0	0	C	E	0
0	1	C	C	0
0	0	D	E	1
0	1	D	C	1
0	0	E	B	1
0	1	E	D	1

62

IC, Fall 2022 – Winston Hsu

Specifying outputs for a Mealy machine

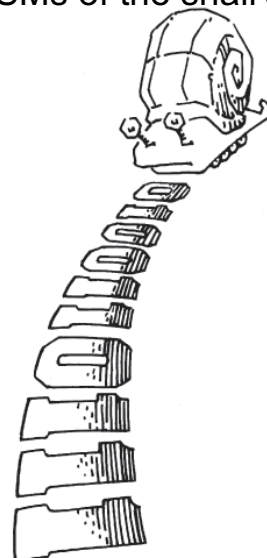
- Output is function of state and inputs
 - specify output on transition arc between states



reset	input	current	next	output
		state	state	
1	—	—	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

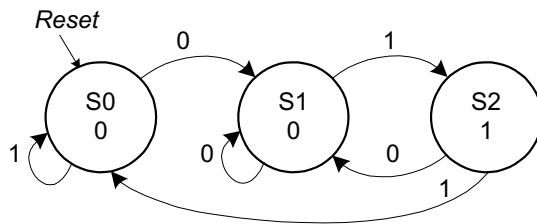
Moore vs. Mealy FSM

- Alyssa P. Hacker has a snail that crawls down a paper tape with 1's and 0's on it. The snail smiles whenever the last two digits it has crawled over are 01. Design Moore and Mealy FSMs of the snail's brain.

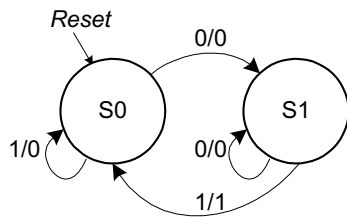


State Transition Diagrams

Moore FSM



Mealy FSM



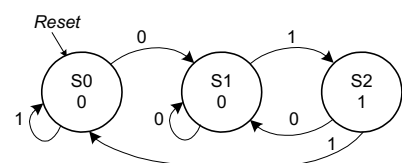
Mealy FSM: arcs indicate input/output

Moore FSM State Transition Table

Current State		Inputs	Next State	
S_1	S_0		S'_1	S'_0
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		

State	Encoding
S0	00
S1	01
S2	10

Moore FSM



Moore FSM State Transition Table

Current State		Inputs A	Next State	
S_1	S_0		S'_1	S'_0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

State	Encoding
S0	00
S1	01
S2	10

$$S'_1 = S_0 A$$

$$S'_0 = \overline{A}$$

Moore FSM Output Table

Current State		Output
S_1	S_0	Y
0	0	
0	1	
1	0	

$$Y = S_1$$

Moore FSM Output Table

Current State		Output
S_1	S_0	Y
0	0	0
0	1	0
1	0	1

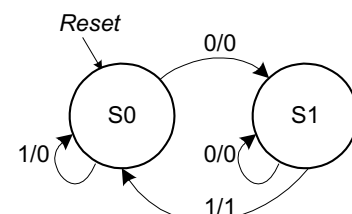
$$Y = S_1$$

Mealy FSM State Transition & Output Table

Current State	Input	Next State	Output
S_0	A	S'_0	Y
0	0		
0	1		
1	0		
1	1		

State	Encoding
S0	00
S1	01

Mealy FSM

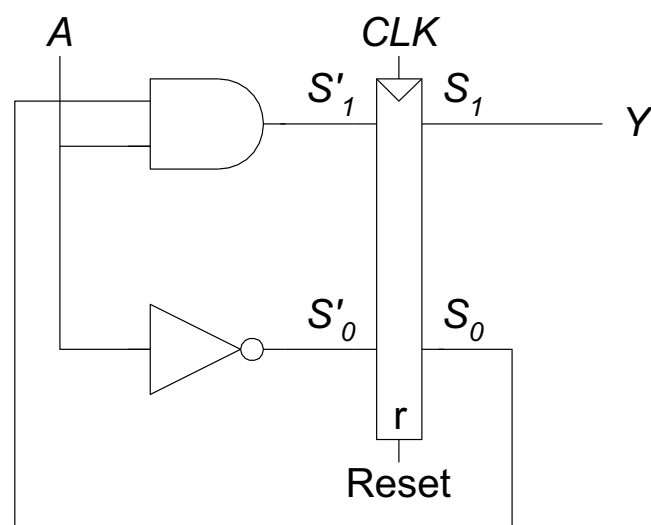


Mealy FSM State Transition & Output Table

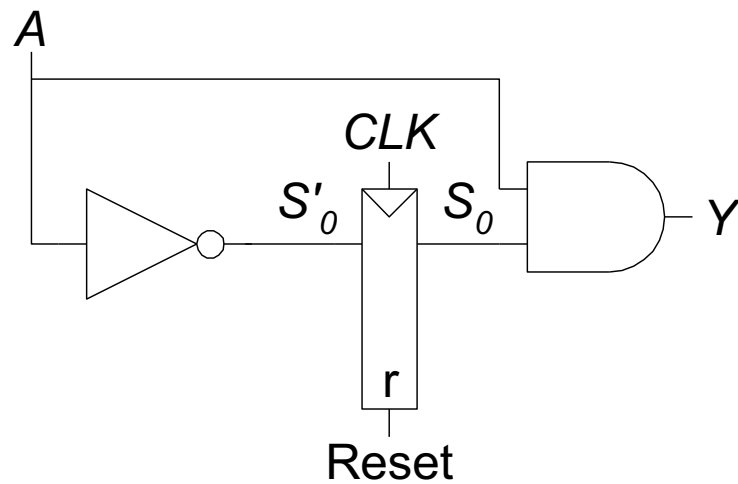
Current State	Input	Next State	Output
S_0	A	S'_0	Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

State	Encoding
S0	00
S1	01

Moore FSM Schematic



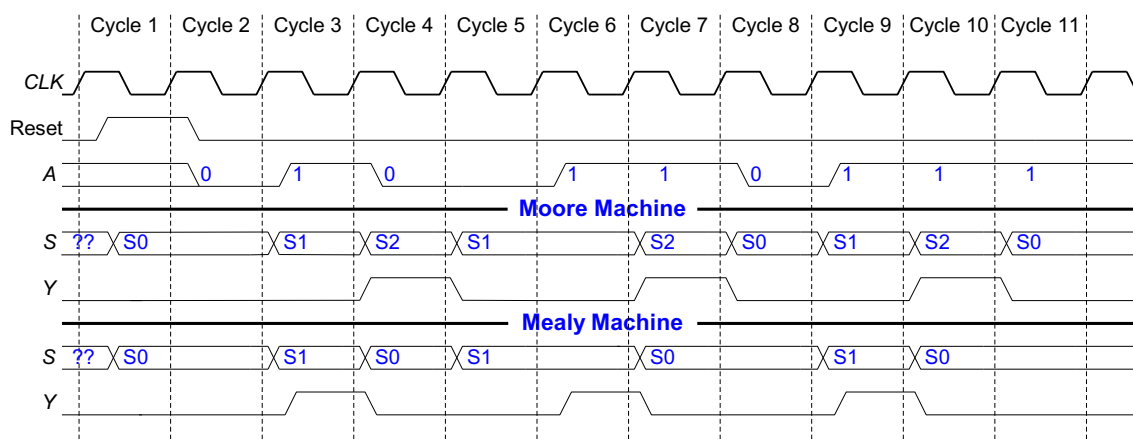
Mealy FSM Schematic



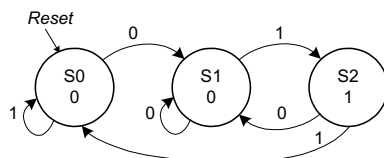
73

IC, Fall 2022 – Winston Hsu

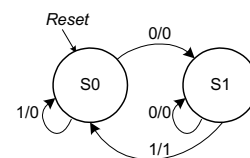
Moore & Mealy Timing Diagram



Moore FSM



Mealy FSM

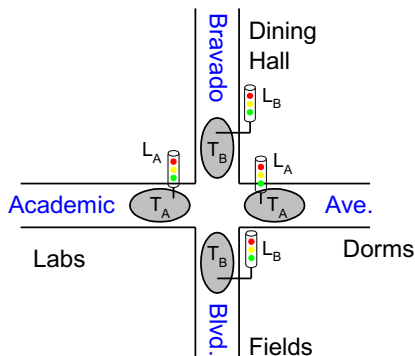


74

IC, Fall 2022 – Winston Hsu

Factoring State Machines

- Break complex FSMs into smaller interacting FSMs
- Example: Modify traffic light controller to have Parade Mode.
 - Two more inputs: P , R
 - When $P = 1$, enter Parade Mode & Bravado Blvd light stays green
 - When $R = 1$, leave Parade Mode



75

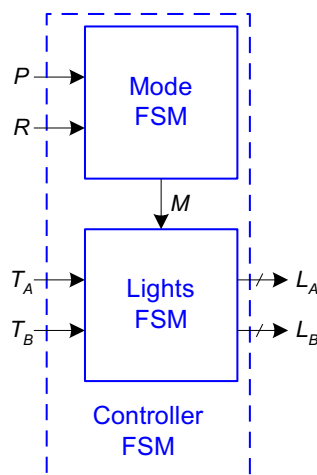
IC, Fall 2022 – Winston Hsu

Parade FSM

Unfactored FSM



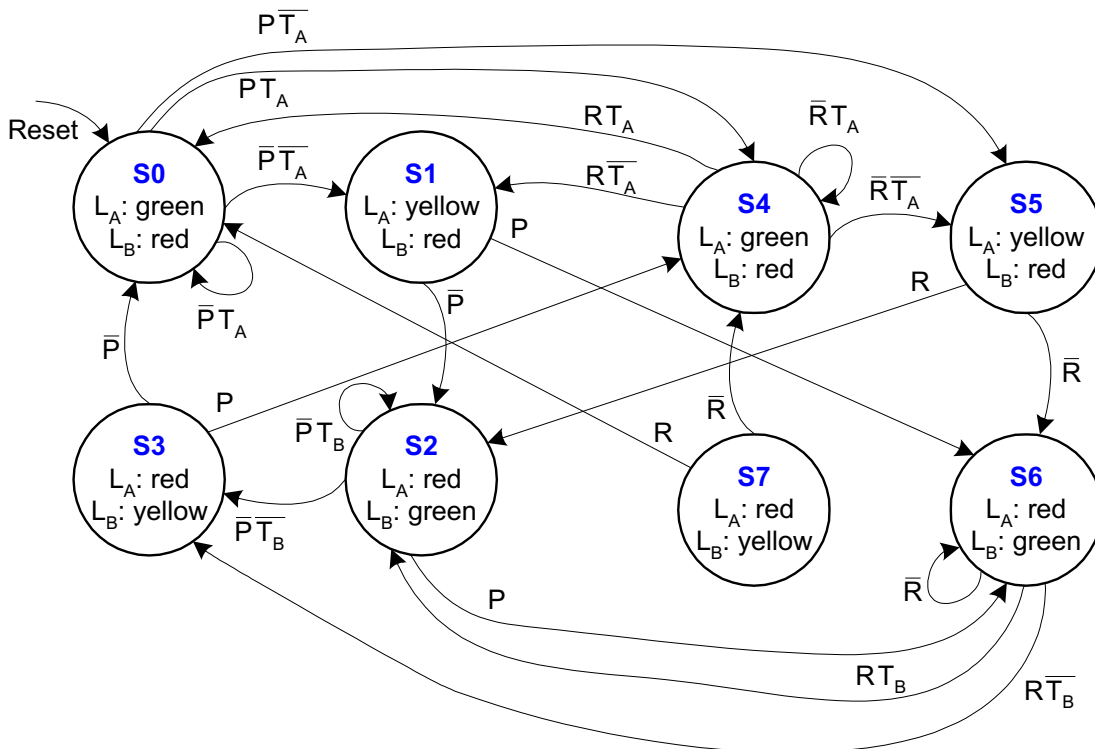
Factored FSM



76

IC, Fall 2022 – Winston Hsu

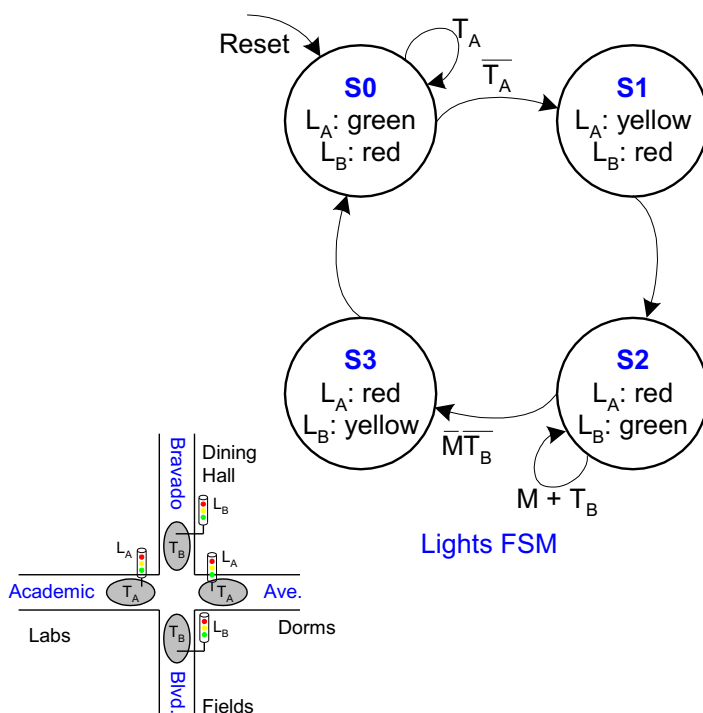
Unfactored FSM



77

IC, Fall 2022 – Winston Hsu

Factored FSM



Lights FSM

Mode FSM

78

IC, Fall 2022 – Winston Hsu

FSM Design Procedure

- Identify inputs and outputs
- Sketch state transition diagram
- Write state transition table
- Select state encodings
- For Moore machine:
 - Rewrite state transition table with state encodings
 - Write output table
- For a Mealy machine:
 - Rewrite combined state transition and output table with state encodings
- Write Boolean equations for next state and output logic
- Sketch the circuit schematic

To-Do Items

- Today's lecture sections
 - Chapter 3 (parts) from Digital Design and Computer Architecture, 2nd Edition, David Harris and Sarah Harris (except. Sec. 3.2.7., Sec. 3.5)