
Computer Aided Digital Systems Design - EE 4743/6743

Sherif Abdelwahed

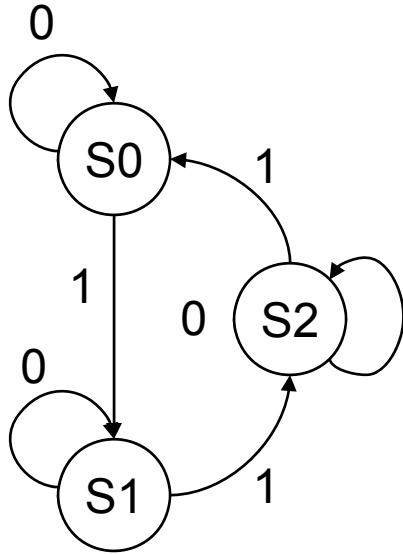
Finite State Machine Review

**Department of Electrical and Computer Engineering
Mississippi State University**

Finite State Machines (FSMs)

- Any Circuit with Memory Is a Finite State Machine
 - Even computers can be viewed as huge FSMs
 - Design of FSMs Involves
 - Defining states
 - Defining transitions between states
 - Defining outputs to transitions and/or states.
 - Above Approach Is Practical for Simple FSMs Only
 - The function of a state machine (or sequential circuit) can be represented by:
 - State table,
 - State diagram,
 - Flowchart,
 - Algorithmic State Machine (ASM)
-

Example State Machine Representations



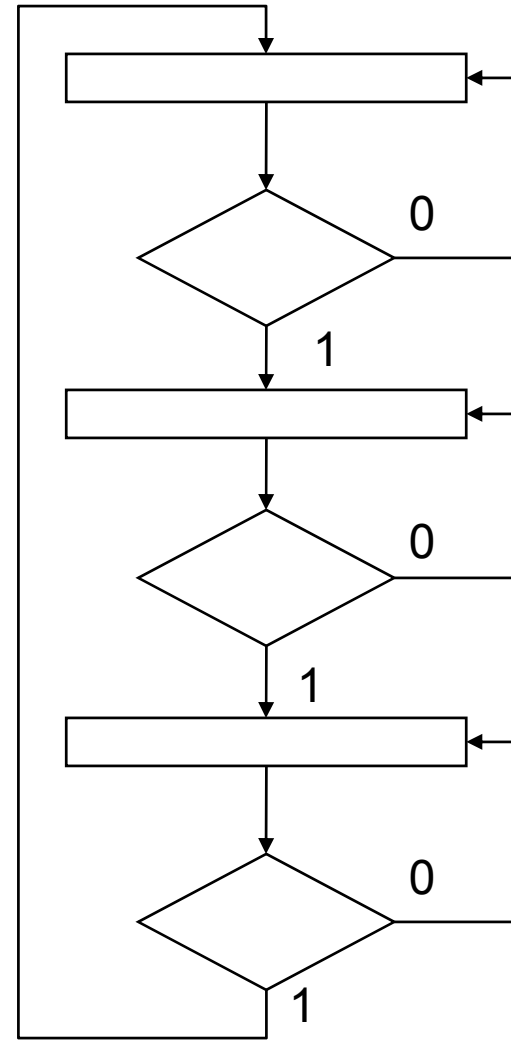
State diagram
(State transition graph)

S0

S1

S2

ASM Chart

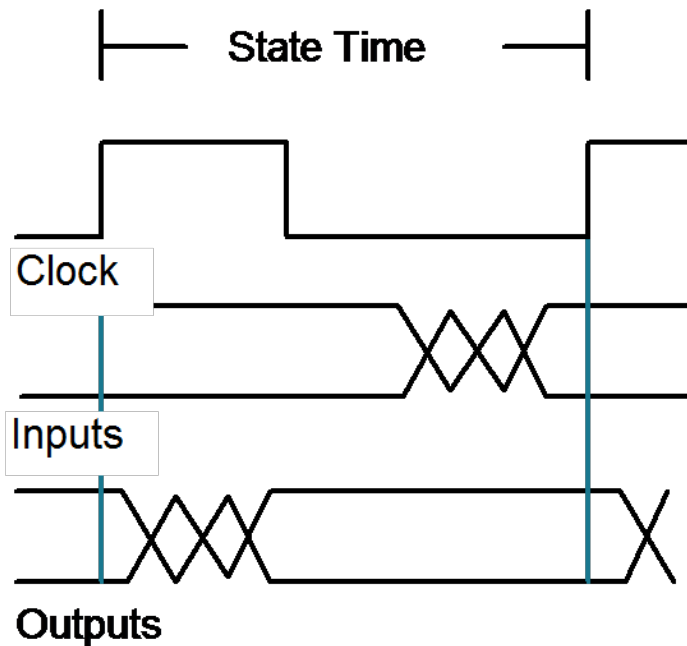


State Machine Timing

- **Timing Issues:**
 - When are inputs sampled, next state computed, outputs asserted?
 - **State Time:** Time between clocking events
 - Clocking event causes state/outputs to transition, based on inputs
 - For set-up/hold time considerations:
 - Inputs should be stable before clocking event
 - After propagation delay, Next State entered, Outputs are stable. Note:
 - Asynchronous signals take effect immediately
 - Synchronous signals take effect at the next clocking event
-

State Machine Timing

Example: Positive Edge Triggered Synchronous System



On rising edge, inputs sampled
outputs, next state computed

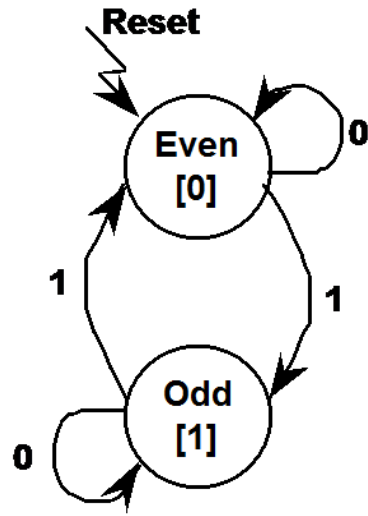
After propagation delay, outputs and
next state are stable

Immediate Outputs:
affect datapath immediately
could cause inputs from datapath to change

Delayed Outputs:
take effect on next clock edge
propagation delays must exceed hold times

Example FSM: Odd Parity Checker

Assert output whenever input bit stream has odd # of 1's



State Diagram

Present State	Input	Next State	Output
Even	0	Even	0
Even	1	Odd	0
Odd	0	Odd	1
Odd	1	Even	1

Symbolic State Transition Table

Present State	Input	Next State	Output
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Encoded State Transition Table

Design Procedure

- Design starts from a specification and results in a logic diagram or a list of Boolean functions.
 - The steps to be followed are:
 1. Derive a state diagram
 2. Reduce the number of states
 3. Assign binary value to states
 4. Obtain the binary coded state table
 5. Choose the type of flip flops to be used
 6. Derive the simplified flip flop input equations and output equations
 7. Draw the logic diagram
-

State Assignment

- State assignment is the binary coding used to represent the states
- Given N states, need at least $\log_2(N)$ FFs to encode the states (i.e. 3 states, need at least 2 FFs for state information).

$S_0 = 00$, $S_1 = 01$, $S_2 = 10$ (FSM is now a modulo 3 counter)

- Do not always have to use the fewest possible number of FFs. A common encoding is One-Hot encoding - use one FF per state.

$S_0 = 001$, $S_1 = 010$, $S_2 = 100$

- State assignment affects speed, gate count of FSM
-

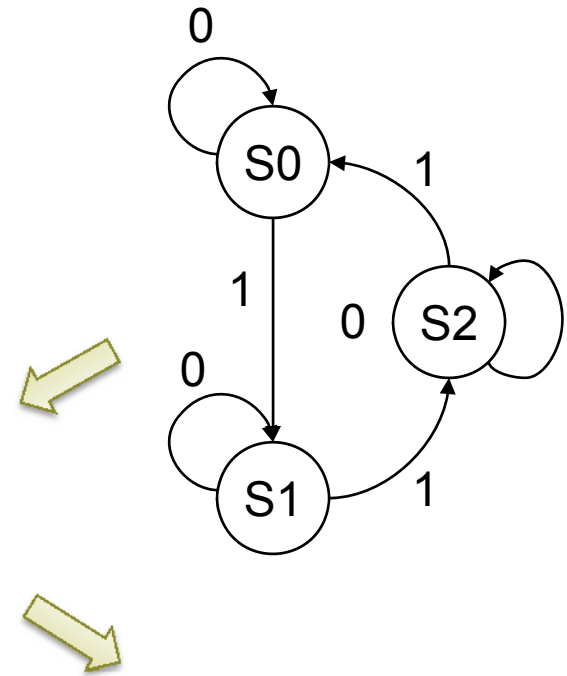
Example FSM Implementation

Use DFFs,

State assignment: S0 = 00, S1 = 01, S2 = 10

Input	CS		NS		FF Output	
	Q1	Q0	Q1+	Q0+	D1	D0
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	1	0	1	0
0	1	1	X	X	X	X
1	0	0	0	1	0	1
1	0	1	1	0	1	0
1	1	0	0	0	0	0
1	1	1	X	X	X	X

State Table



Equations

$$D1 = \text{Inc}'Q1Q0' + \text{Inc}Q1'Q0$$

$$D0 = \text{Inc}'Q1'Q0 + \text{Inc}Q1'Q0'$$

Minimize Equations

D1

		Q1Q0			
		00	01	11	10
Inc	0	0	0	x	1
	1	0	1	x	0

$$D1 = \text{Inc}' Q1 + \text{Inc} Q0$$

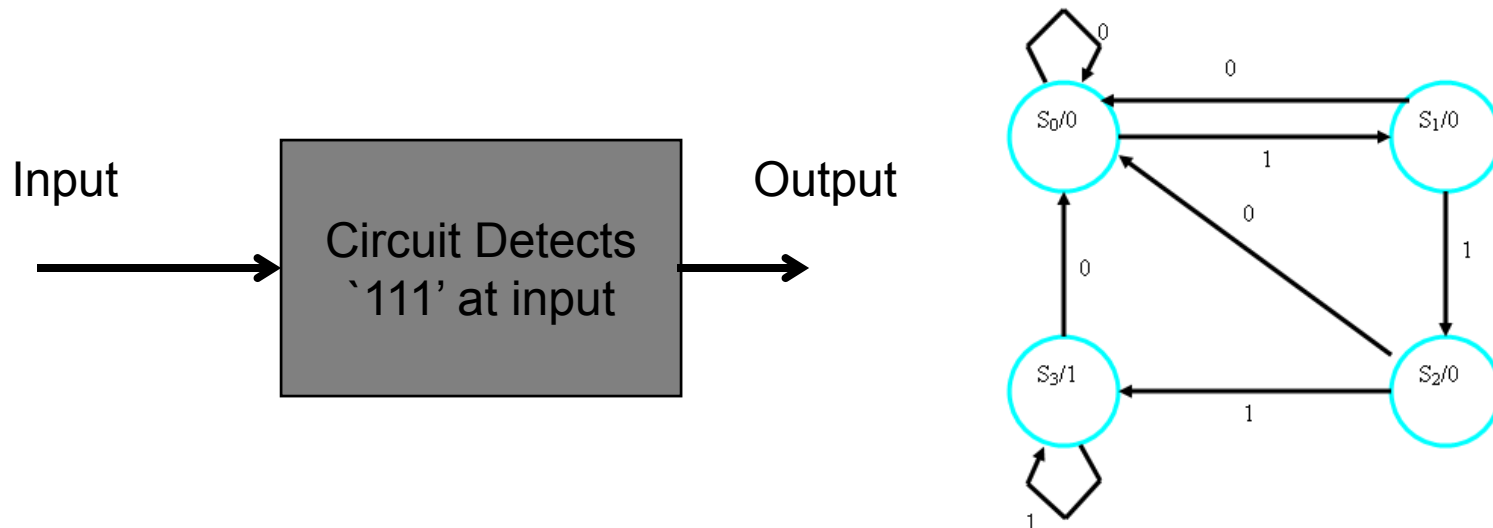
D0

		Q1Q0			
		00	01	11	10
Inc	0	0	1	x	0
	1	1	0	x	0

$$D1 = \text{Inc}' Q0 + \text{Inc} Q1'Q0'$$

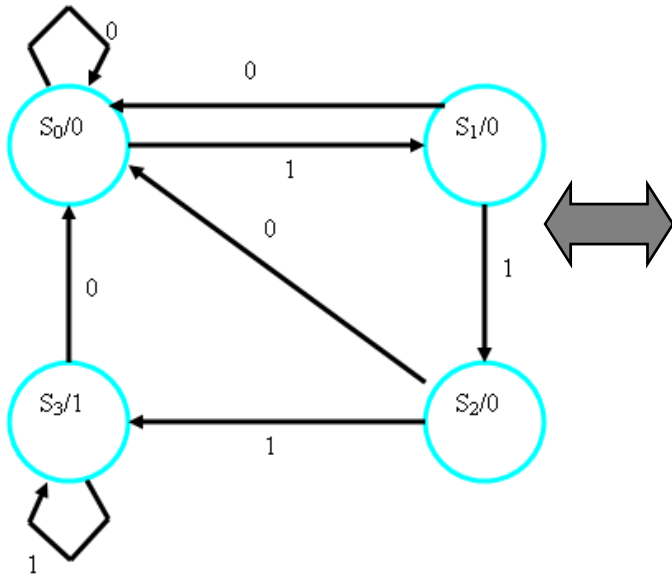
Example: A Sequence Detector

Design a circuit that detects a sequence of three ones.



- Create a state table and then select two D flip flops to represent the four states, labeling their outputs as A and B.
- There is one input, x , and one output, y , representing the input sequence and the output value respectively.
- The output y is one only when we detect the input sequence of '111'

State Table for Sequence Detector



Present State		Input x	Next State		Output y
A	B		A	B	
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

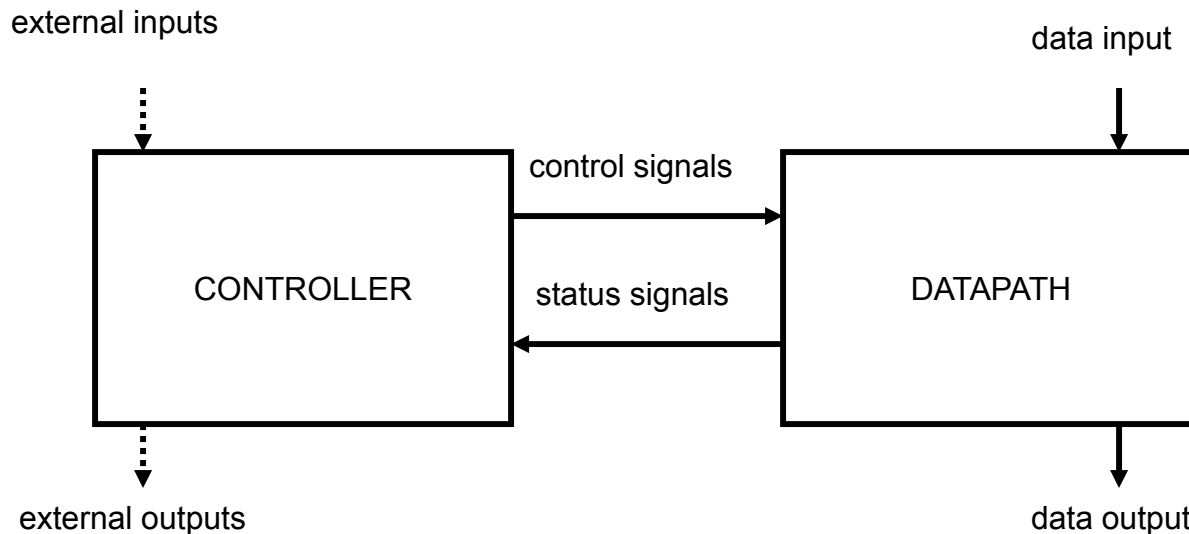
Input equations can be obtained directly from the table using minterms:

- $A(t + 1) = D_A(A, B, x) = \sum(3, 5, 7)$
- $B(t + 1) = D_B(A, B, x) = \sum(1, 5, 7)$
- $y(A, B, x) = \sum(6, 7)$

K-Maps can be used to minimize the input equations, resulting in

- $D_A = Ax + Bx$
- $D_B = Ax + B'x$
- $Y = AB$

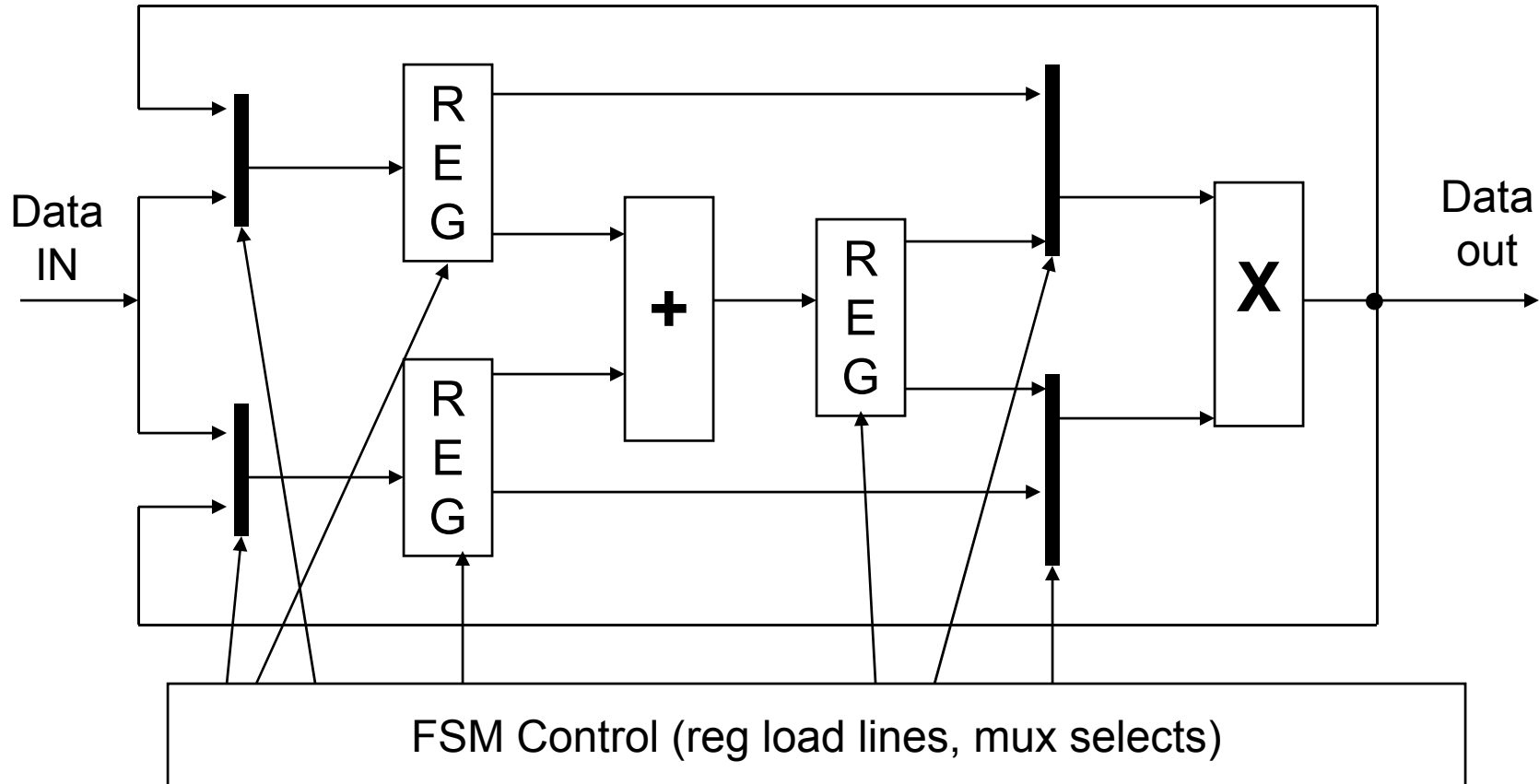
Complex Digital System Design



- Complex digital systems can be decomposed into a datapath and controller
 - Some refer to this as the register-transfer level (RTL) design method
- Datapath manipulates and processes data
 - To perform arithmetic, logic, shifting, and other data-processing tasks
 - These operations are implemented with ALUs, registers, multiplexers, adders, comparators, etc.
- Controller determines the enabling and sequencing of datapath operations
 - Controller provides signal to activate various processing in the datapath
 - Example: enable signals for registers
 - Example: control signals for muxes
 - Controller also determines the sequence the operations are performed

FSM as Controllers

The job of a finite state machine is to sequence operations on a datapath



Alternative State Machine Representations

- *Why State Diagrams Are Not Enough:*
 - Not flexible enough for describing very complex finite state machines
 - Not suitable for gradual refinement of finite state machine
 - Do not obviously describe an *algorithm*: that is, well specified sequence of actions based on input data
 - algorithm = sequencing + data manipulation
separation of control and data
 - *Gradual shift towards program-like representations:*
 - Algorithmic State Machine (ASM) Notation
 - Hardware Description Languages (e.g., VHDL)
-

Algorithmic State Machine (ASM)

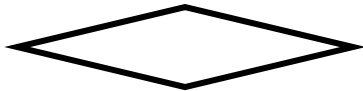
- Complex digital systems can be represented by algorithmic state machines
 - Simple finite state machines (expressed using state diagrams and state tables) good only for simple designs
 - Algorithmic State Machines (ASM) are
 - flow-chart type diagrams to represent finite state machines
 - suitable for a larger number of inputs and outputs compared to simple FSMs
-

Algorithmic State Chart (ASM)

- Only three action signals can appear within a basic ASM chart:



State box. Each box represents a state. Outputs within a state box is an UNCONDITIONAL output (always asserted in this state).

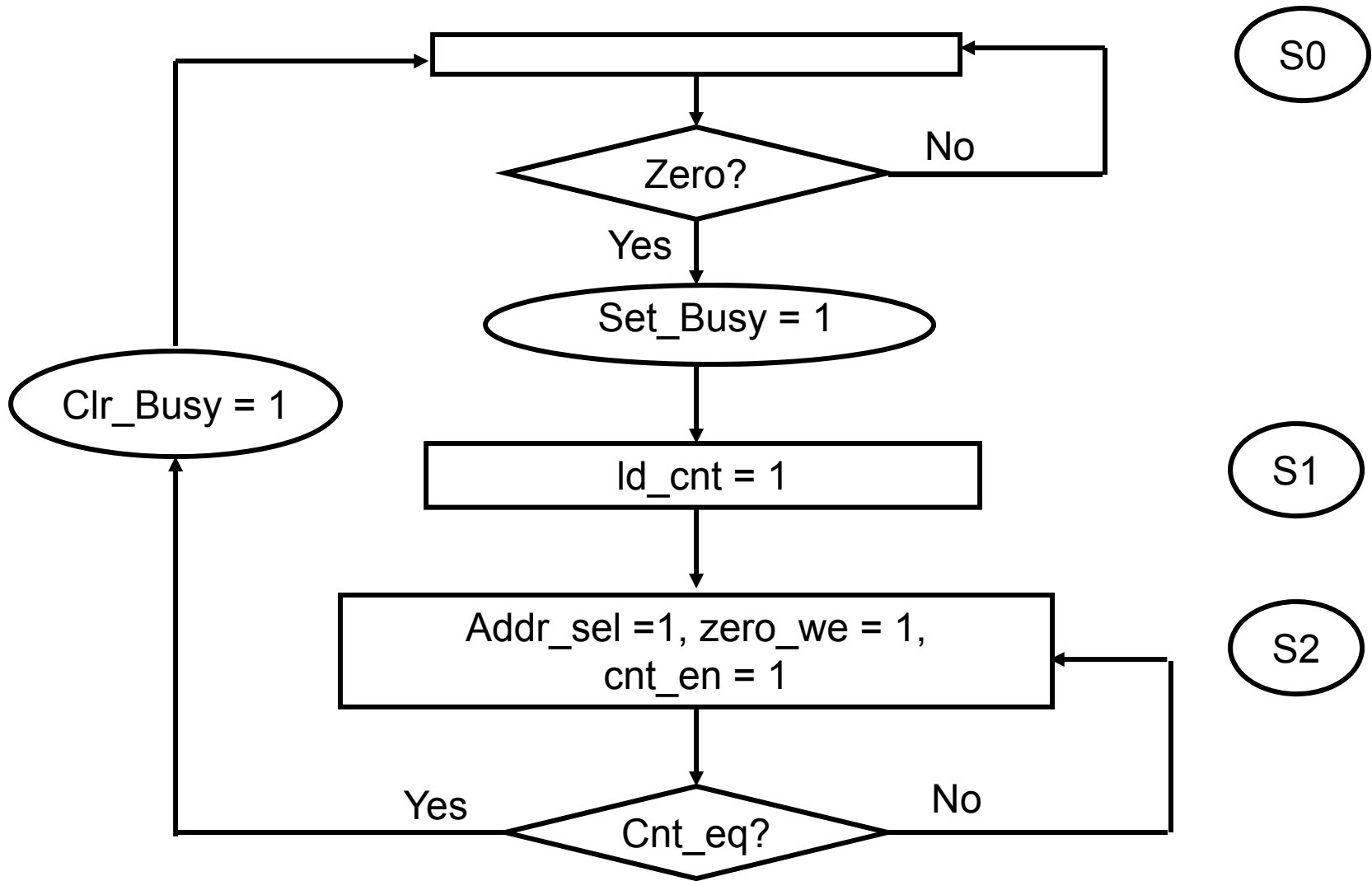


Decision box. A condition in this box will decide next state condition.



Conditional output box. If present, will always follow a decision box; output within it is conditional.

Example ASM Chart



Comments about ASM Example

- How many states?
 - Count the state boxes
 - _____ states
 - How many inputs?
 - Inputs ALWAYS appear within decision boxes
 - Count signals within decision boxes
 - _____ inputs
 - How many outputs?
 - _____ unconditional outputs (count signals within state boxes)
 - _____ conditional output (count signals within conditional output boxes)
 - Outputs ALWAYS appear in either state boxes or conditional output boxes.
-

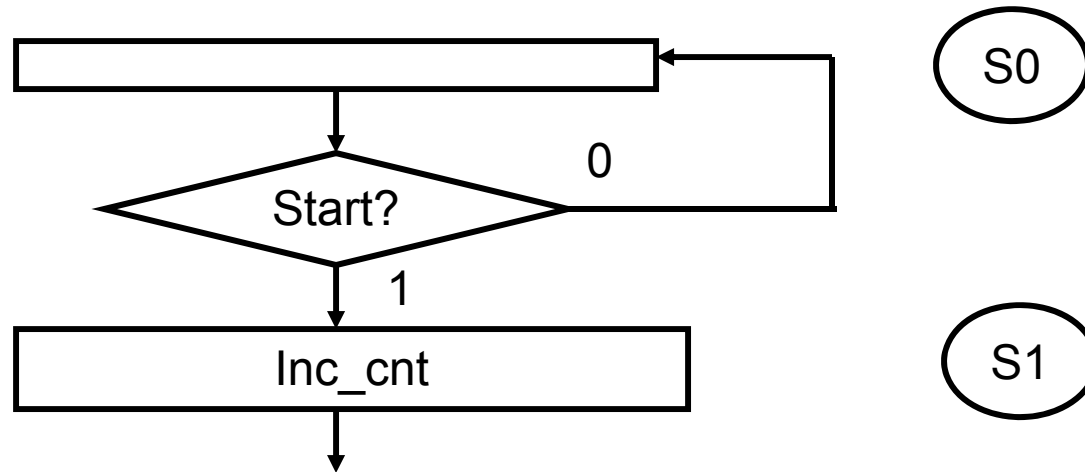
Comments about ASM Example

- How many states?
 - Count the state boxes
 - 3 states
 - How many inputs?
 - Inputs ALWAYS appear within decision boxes
 - Count signals within decision boxes
 - 2 inputs (Zero, Cnt_eq)
 - How many outputs?
 - 4 unconditional outputs (count signals within state boxes)
 - 2 conditional output (count signals within conditional output boxes)
 - Outputs ALWAYS appear in either state boxes or conditional output boxes
-

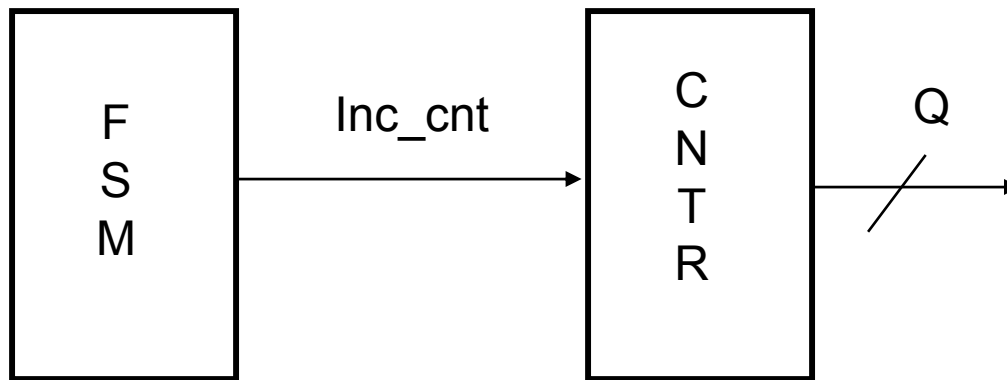
State Assignment

- Minimal encoding for example is three FFs
 - $S_0 = 00$, $S_1 = 01$, $S_2 = 10$ (counting order)
 - $S_0 = 00$, $S_1 = 01$, $S_2 = 11$, (Gray code for $S_0 \rightarrow S_1 \rightarrow S_2$)
Gray code usually faster, less logic than counting order
 - One Hot encoding, one FF per state
 - $S_0 = 001$, $S_1 = 010$, $S_2 = 100$
 - For large FSMs (> 16 states), one hot can be faster than minimal encoding
-

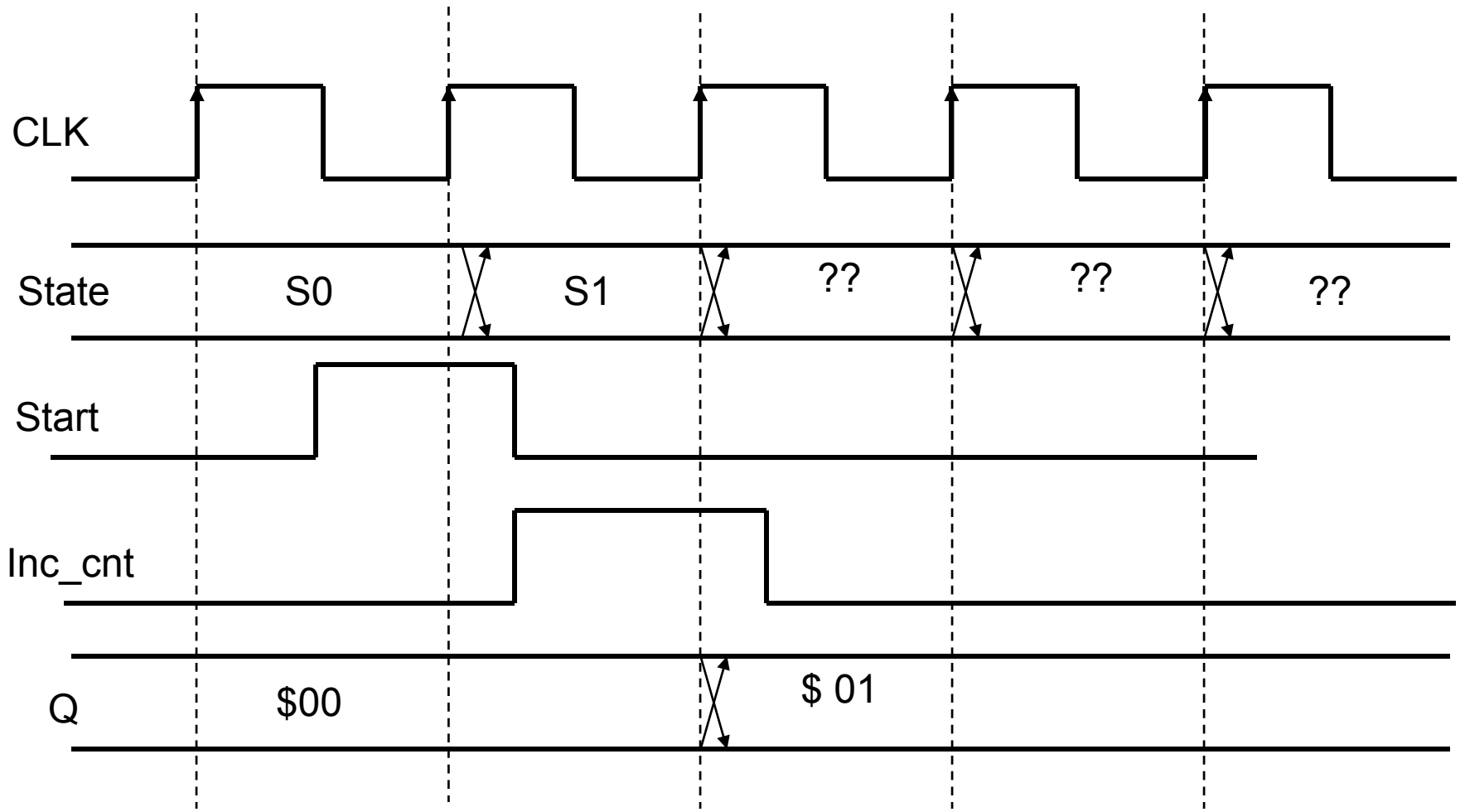
FSM Timing Examples



What does timing look like?

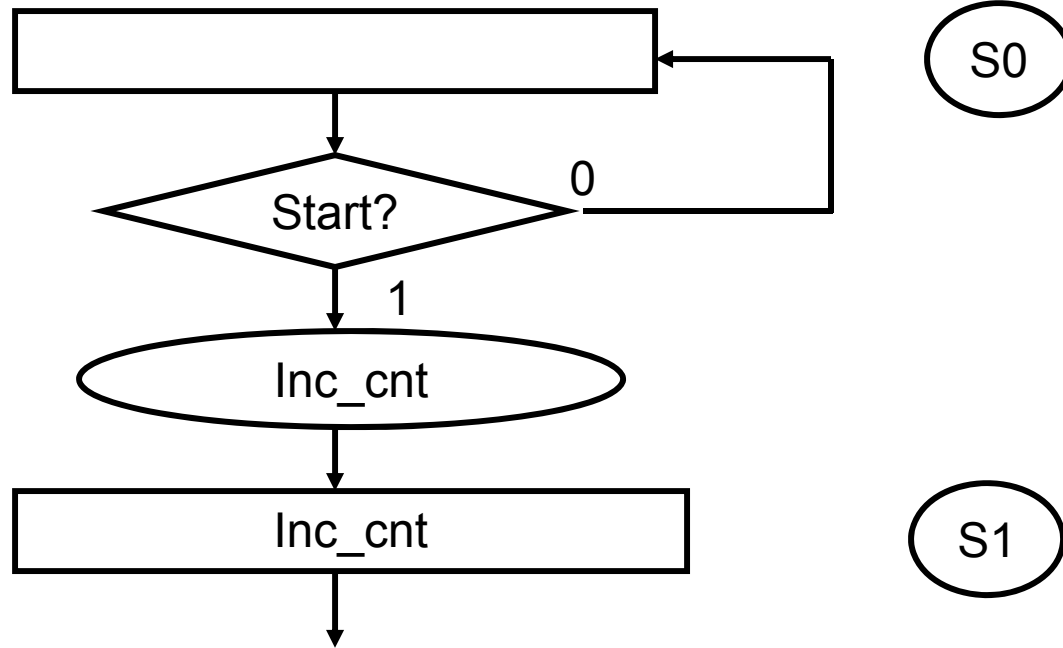


Timing

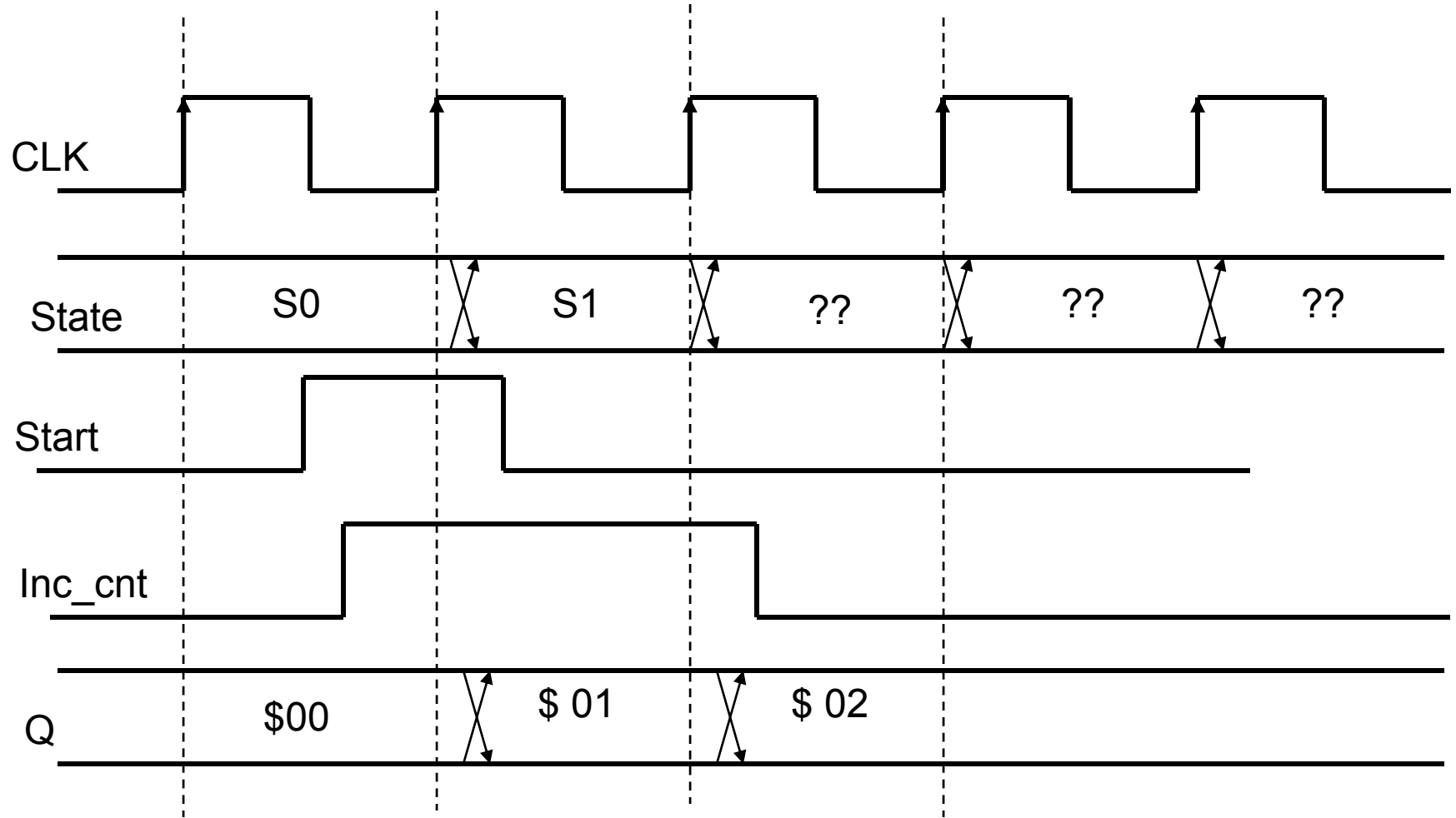


1st edge after Start, Cntr=0; 2nd edge Cntr=0; 3rd edge Cntr=1

How is this different?

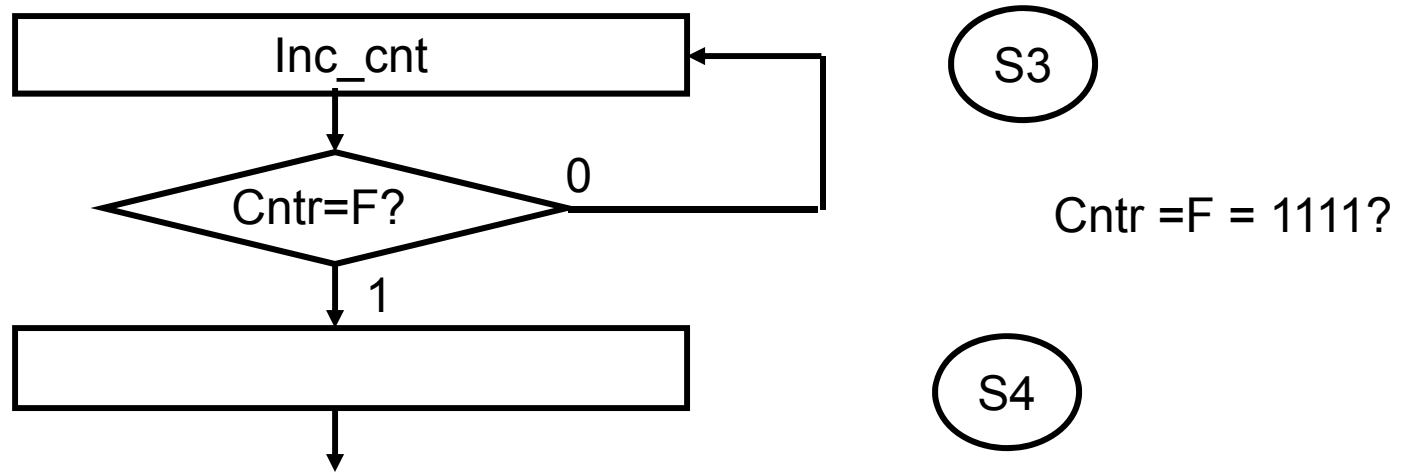


Timing

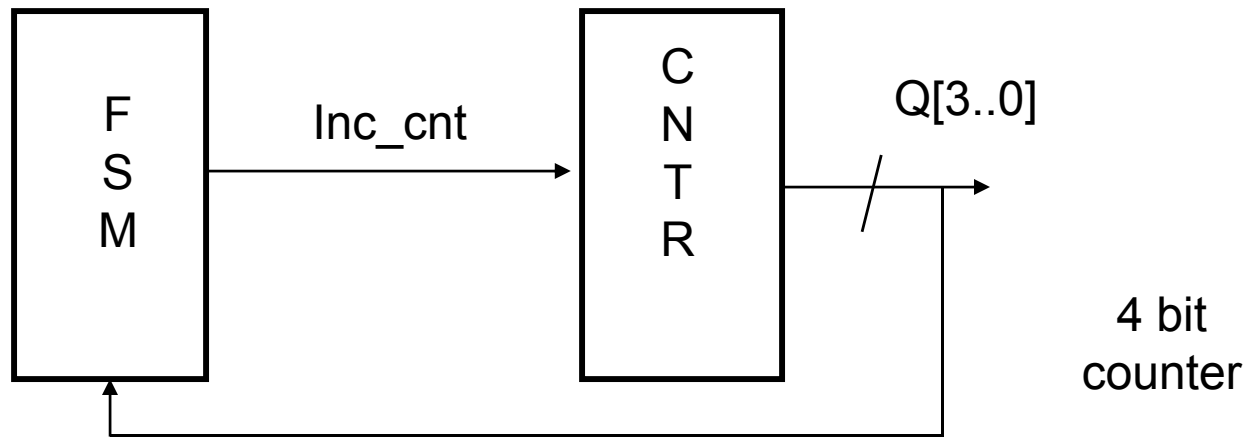


1st edge after Start, Cntr=0; 2nd edge Cntr=1

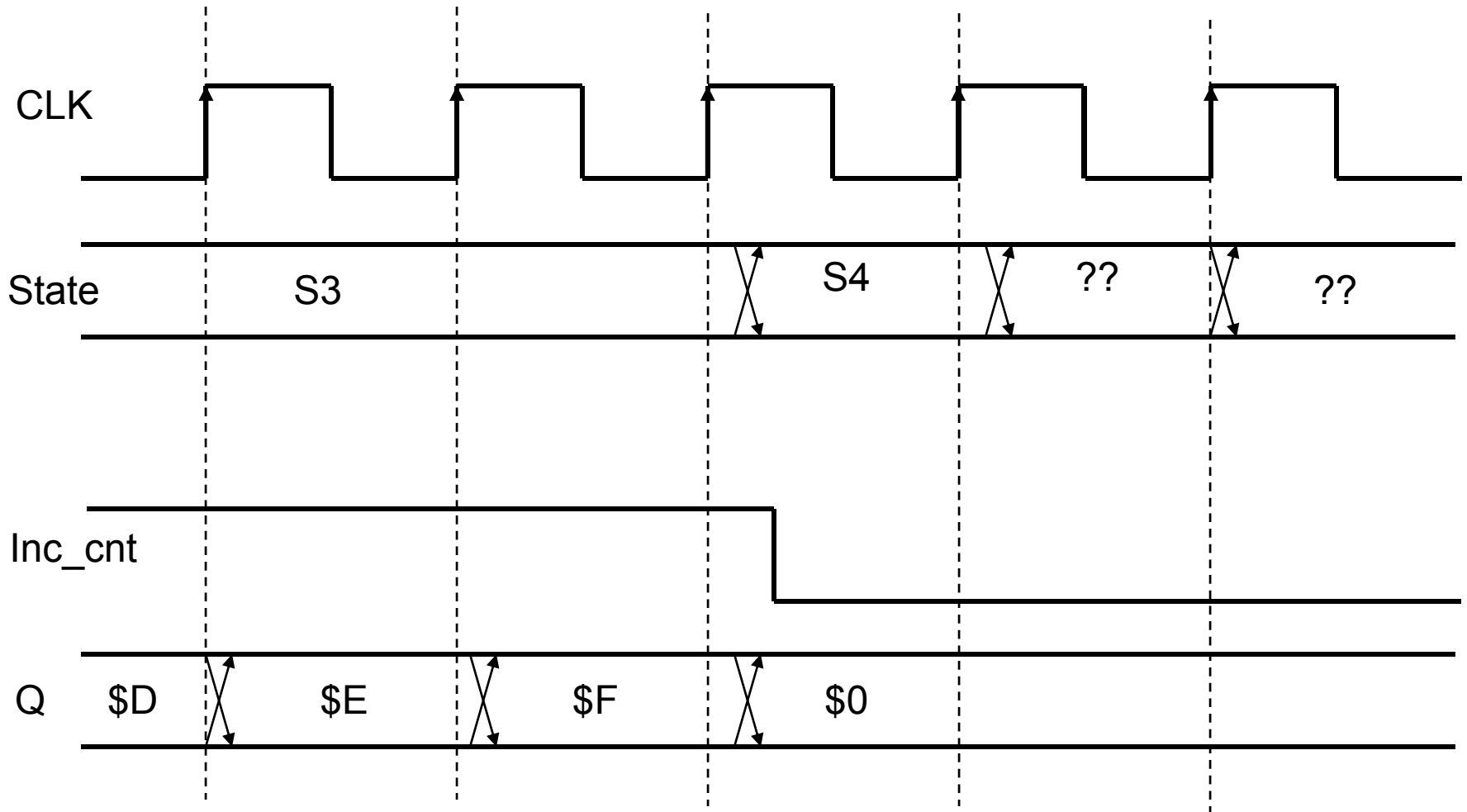
Checking for End of Count



What does timing look like?

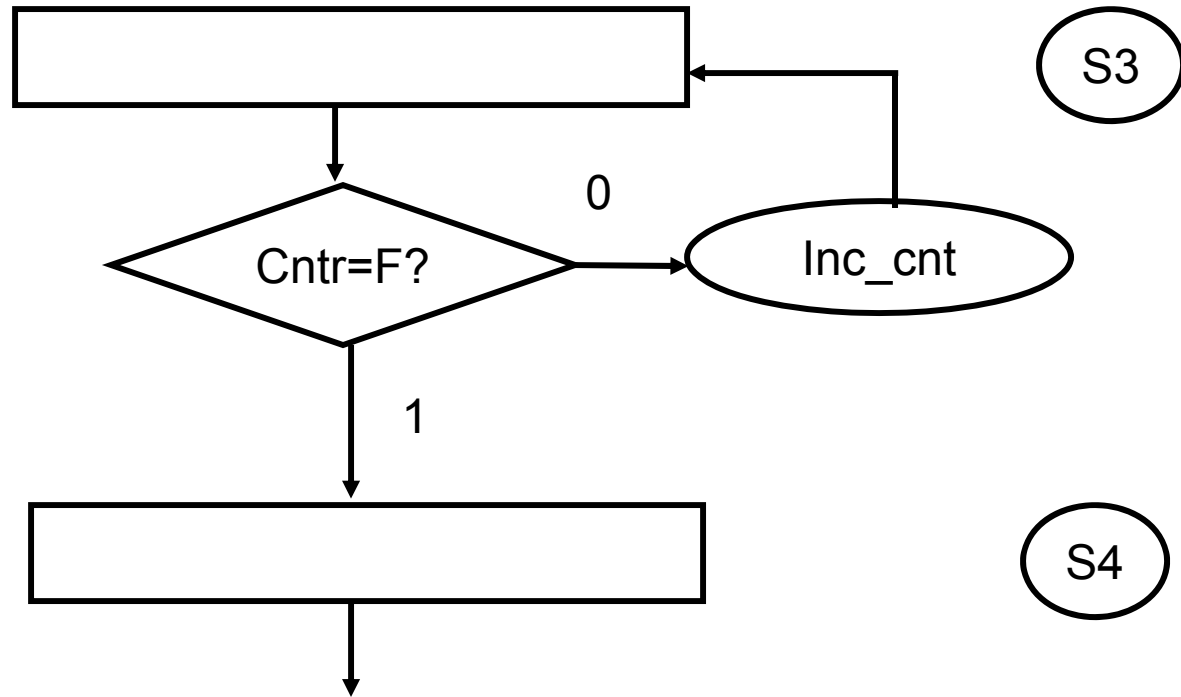


Timing

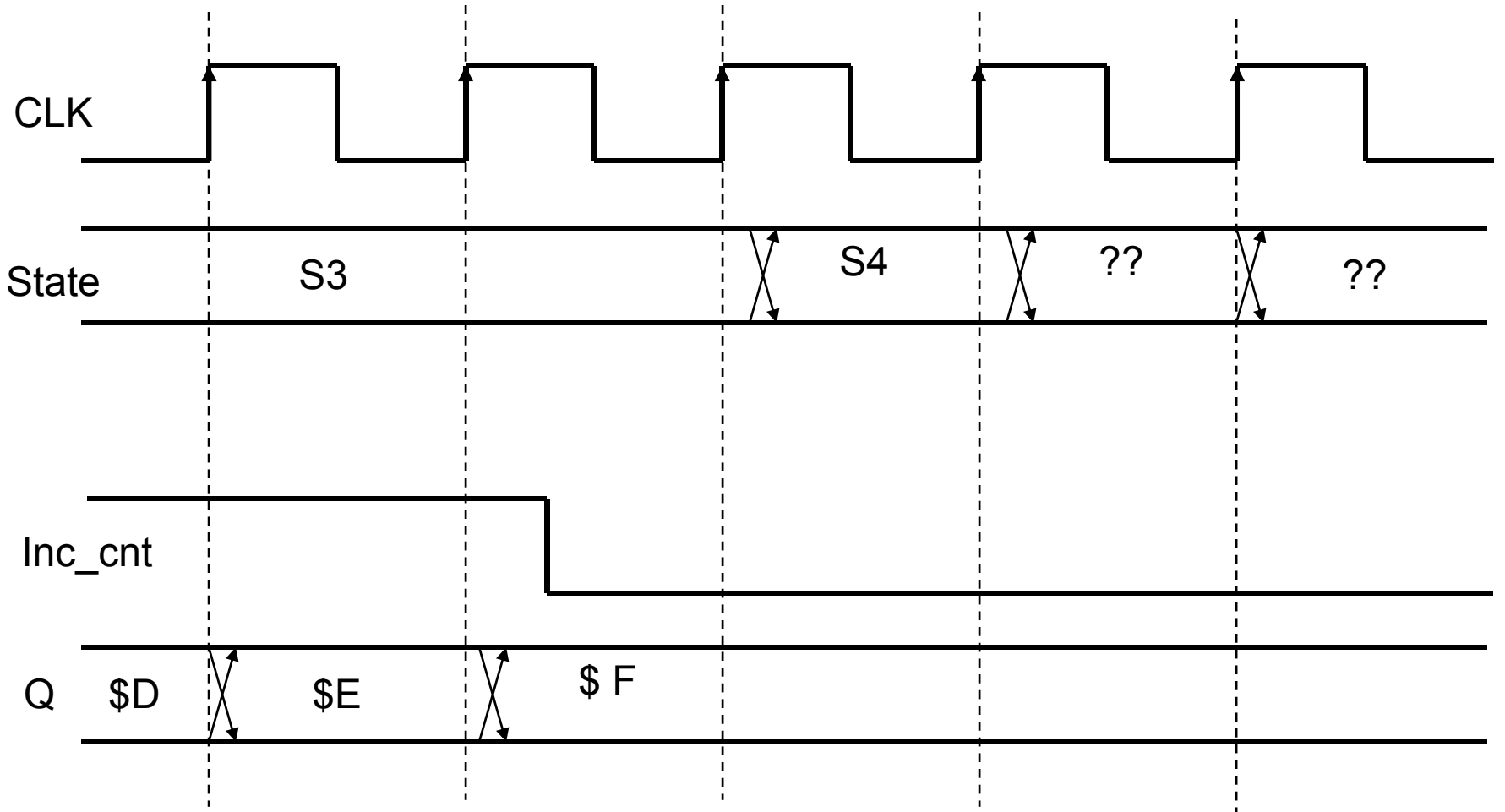


Final value of 4-bit counter is 0

How is this different?

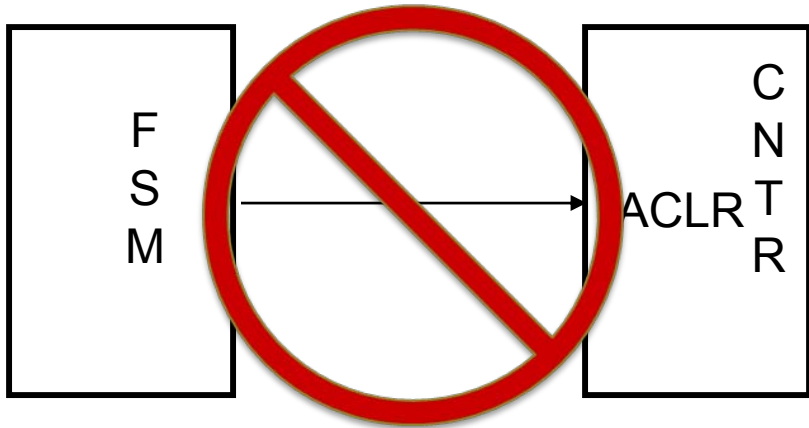


Timing



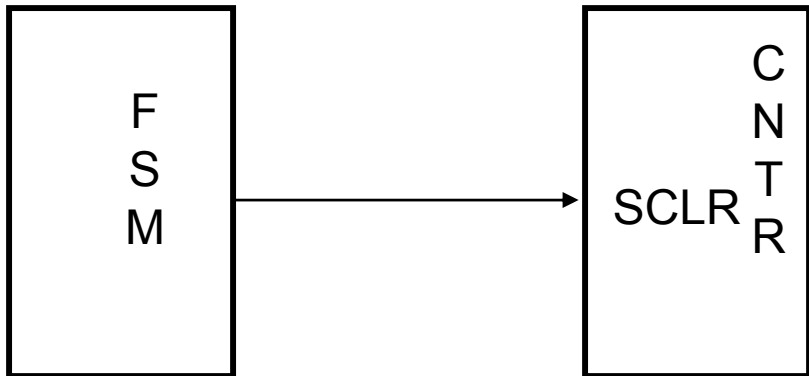
Final value of 4-bit counter is F.

FSM/Datapath Design



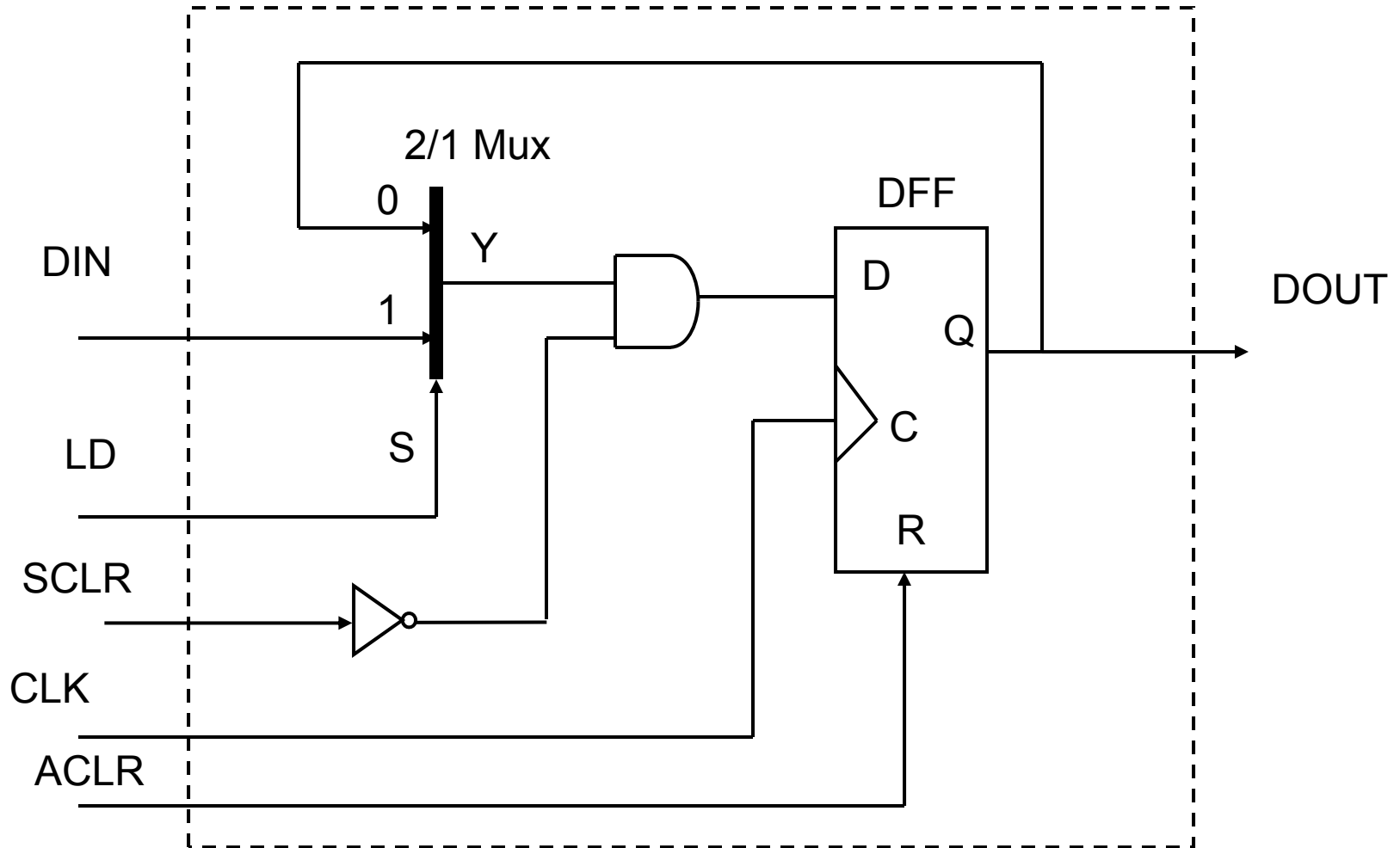
Control of asynchronous clear by FSM - **bad!!**

Glitch on ACLR line by FSM logic can cause inadvertent clear operation! Also, generally want synchronous behavior of CNTR during FSM operation (all outputs changing on clock edges).



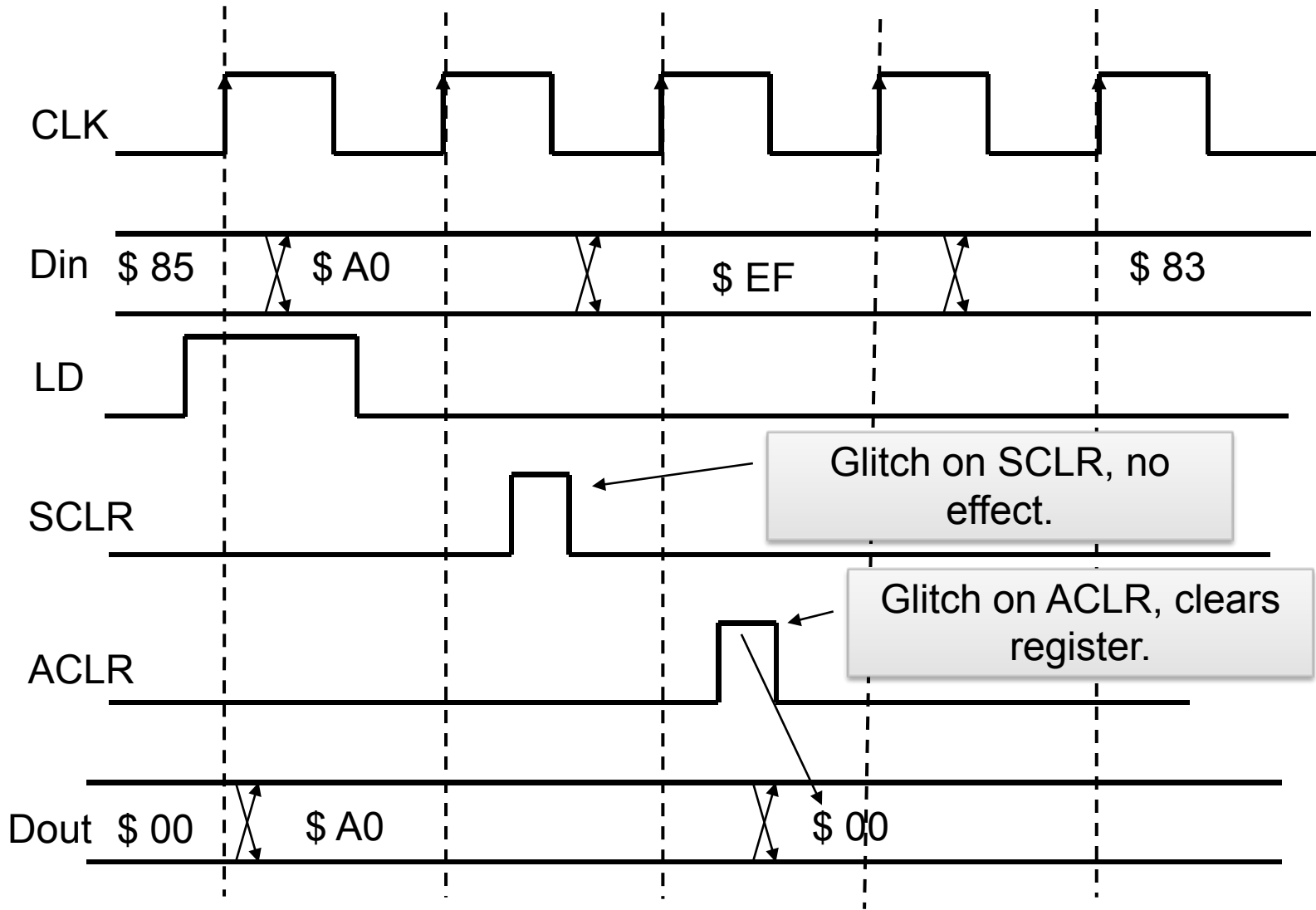
Control of synchronous clear by FSM. **Good!**

1 Bit Register with Synchronous Clear



Note that $SCLR = 1$ will set $DFF=0$ on next active clock edge.

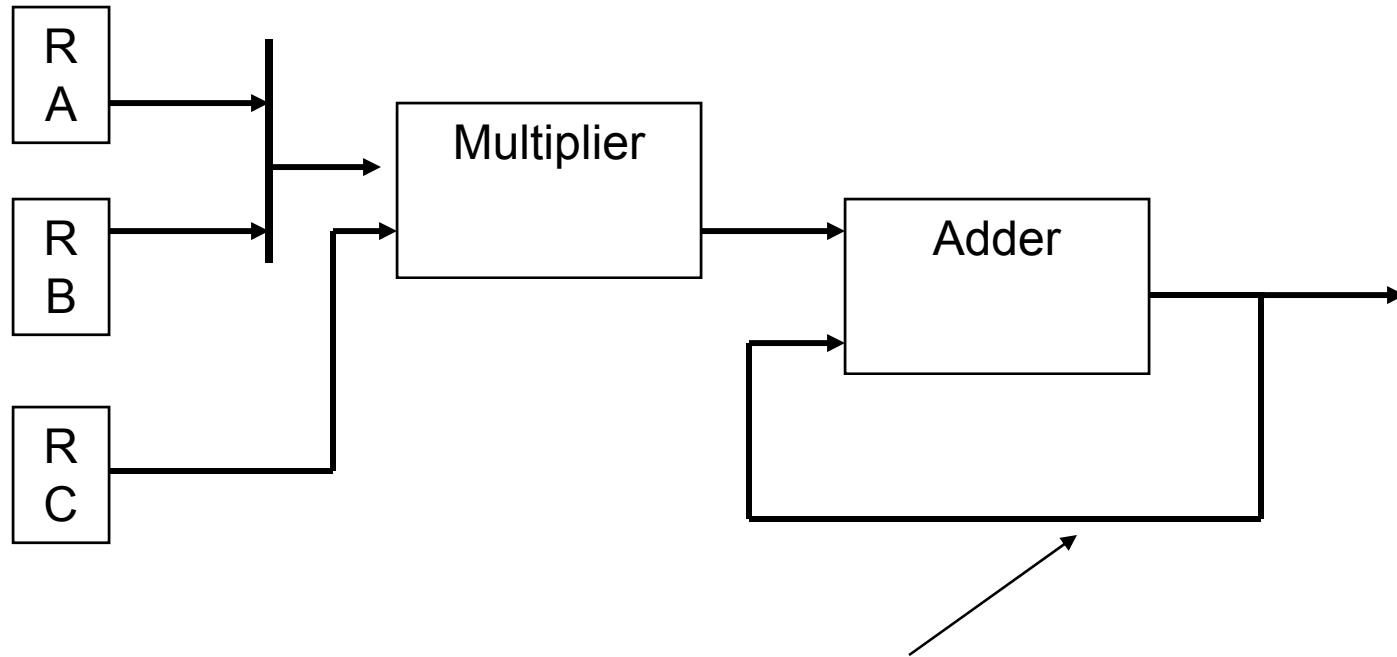
Register Timing (8 Bit register)



What Causes Glitches on outputs of FSMs?

- Many combinational paths in logic that defines next state.
 - If these paths have unequal numbers of gates, or gates have different delays, then glitches can occur.
 - We normally don't care about these glitches as long as the output lines are STABLE before the next clock edge (satisfy the setup time requirement)
 - If output lines are connected to asynchronous control inputs, then glitches can be a BIG problem!
 - Solution: Don't connect asynchronous controls lines to FSM outputs or guarantee FSM outputs are glitch free (come directly from a FF).
-

FSM/Datapath Design

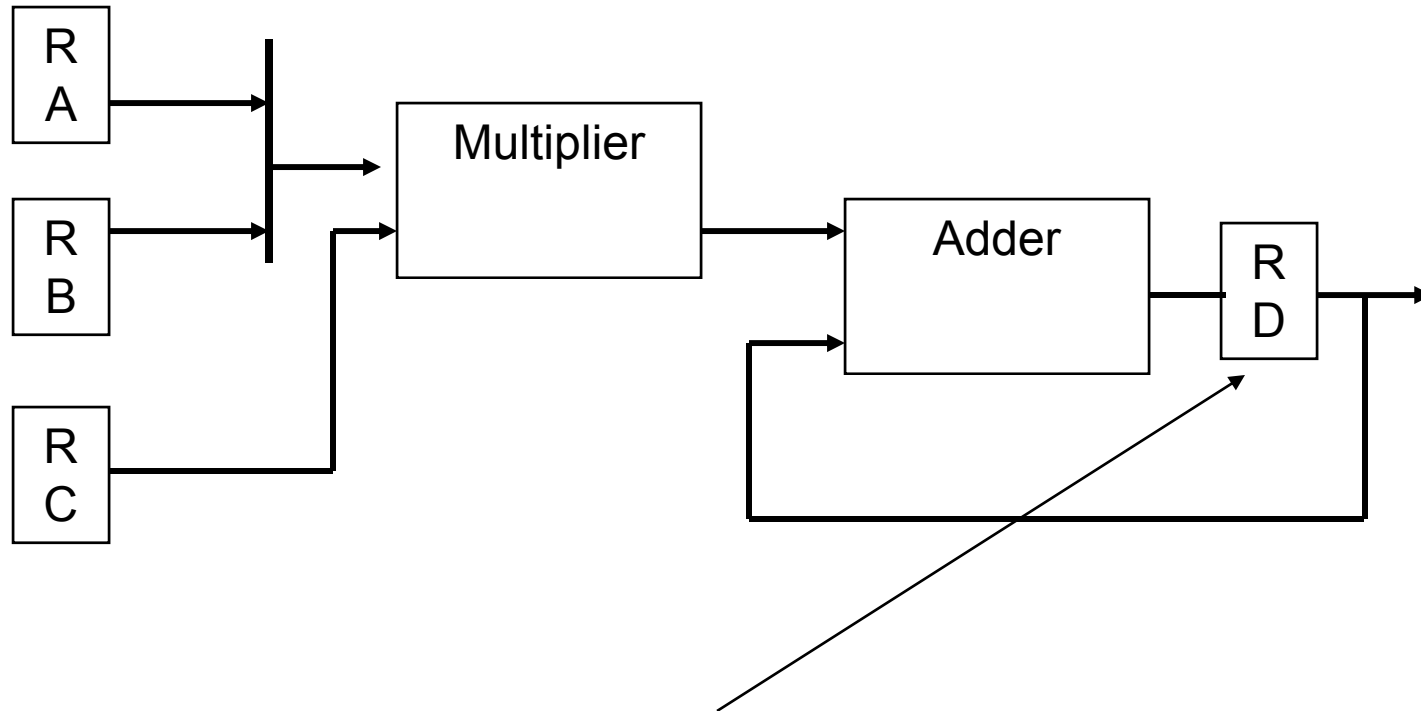


This datapath is trying to accumulate a multiply/add from data values in registers RA, RB, RC.

The adder has a COMBINATIONAL LOOP!! Will oscillate!!!

Bad!!!

FSM/Datapath Design



Must use a Register to hold value of multiply/add for next multiply/add operation. **Good!!!**