SEE 3243/4243
FSM Modelling & Systematic Realization I

Week 9

- Finite State Machine Concept
- Basic Design Procedure
- JASM (Just Another State Machine) Example
- Parity Checker Example
- Counter with Enable Example
- Complex Counter Example

Finite State Machines

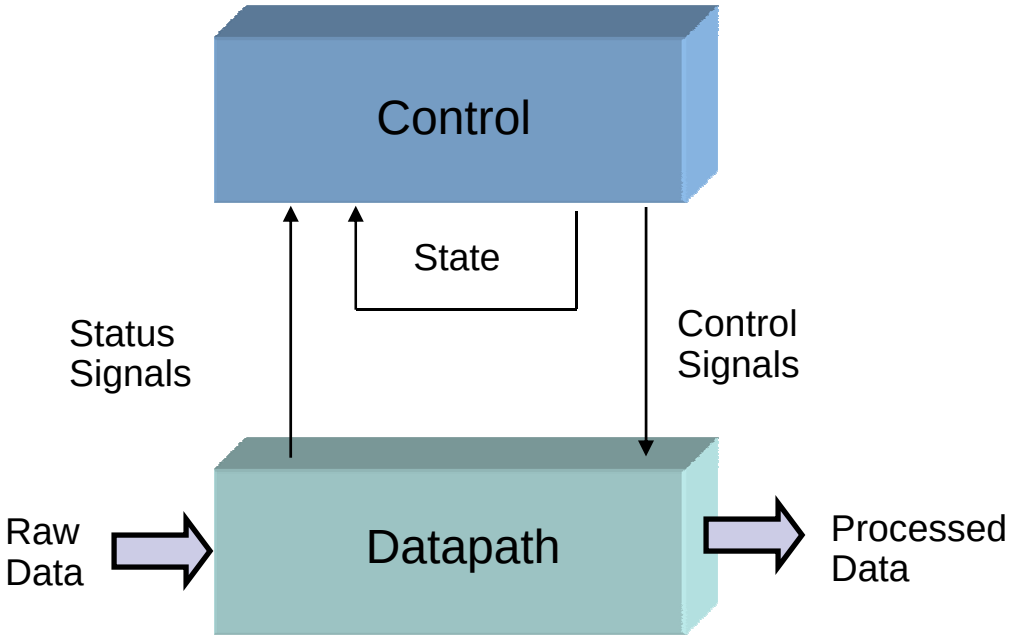
- **State**: collection of state variables containing all information from past needed to predict future behavior
- **Finite state machines** (FSMs): circuits that can be in only a fixed number of possible states
- The counters simple finite state machines.
  - State = output
  - No choice of sequence
- More generally, in FSM:
  - Next State = function of input and present state
  - Outputs = function of input and present state
  - More complex behavior than counters.
- Finite state machines perform decision-making logic

Concept of the State Machine

Computer Hardware = Datapath + Control

- FSM generating sequences of control signals
- Instructs datapath what to do next

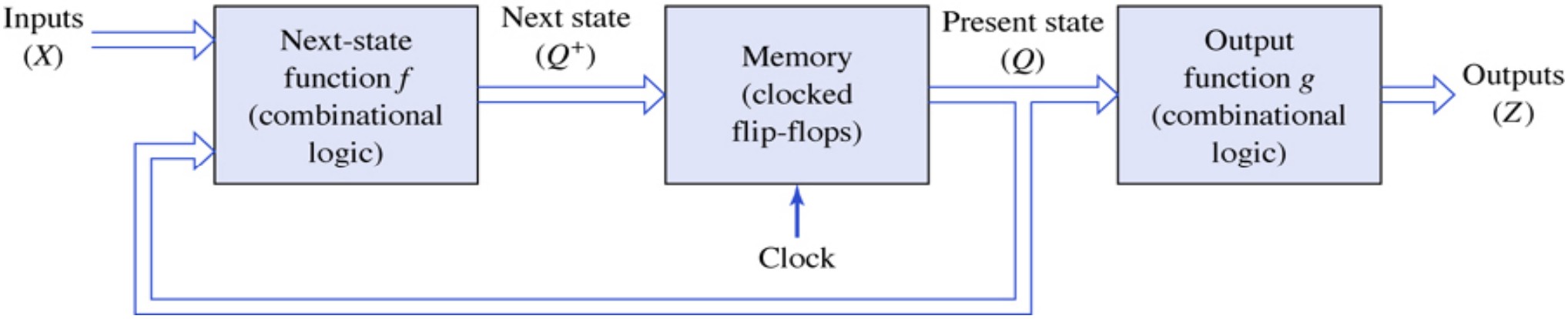
- Registers
- Combinational Functional Units (e.g., ALU)
- Busses



State Machine Structure

- **State memory**:
  - n FFs to store current states. All FFs are connected to a common clock signal.
- **Next-state logic**:
  - determine the next state when state changes occur
- **Output logic**:
  - determines the output as a function of current state and input
- There are three models for Finite State Machine (FSM)
  - Moore model
  - Mealy model
  - Synchronous Mealy model
- What are the differences between all these three models?

Moore Machine

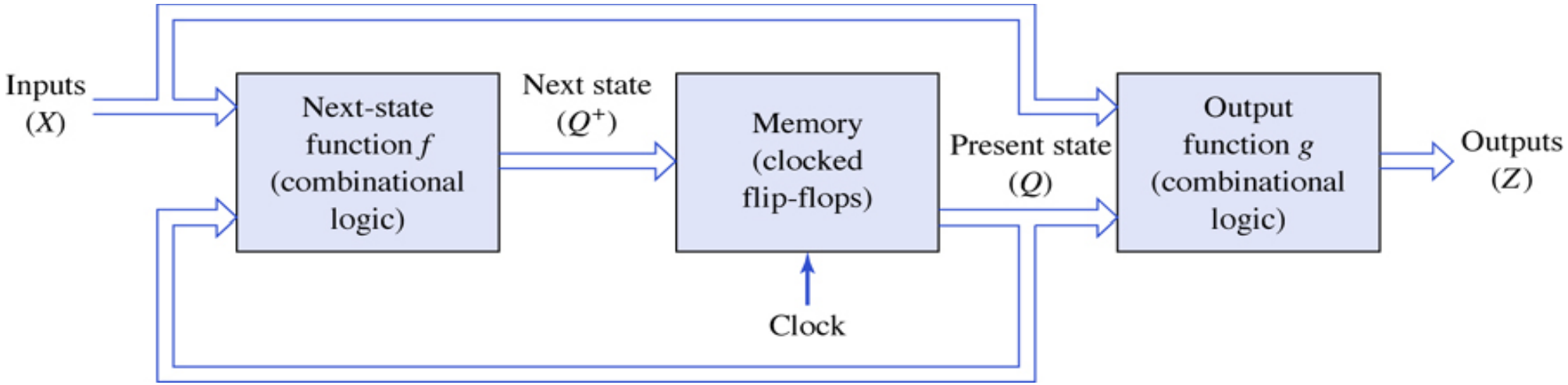


Moore Machine

Outputs are function solely of the current state

Outputs change synchronously with state changes

Mealy Machine



Mealy Machine

Outputs depend on state AND inputs

Asynchronous signals: Input change causes an immediate output change

Moore vs Mealy

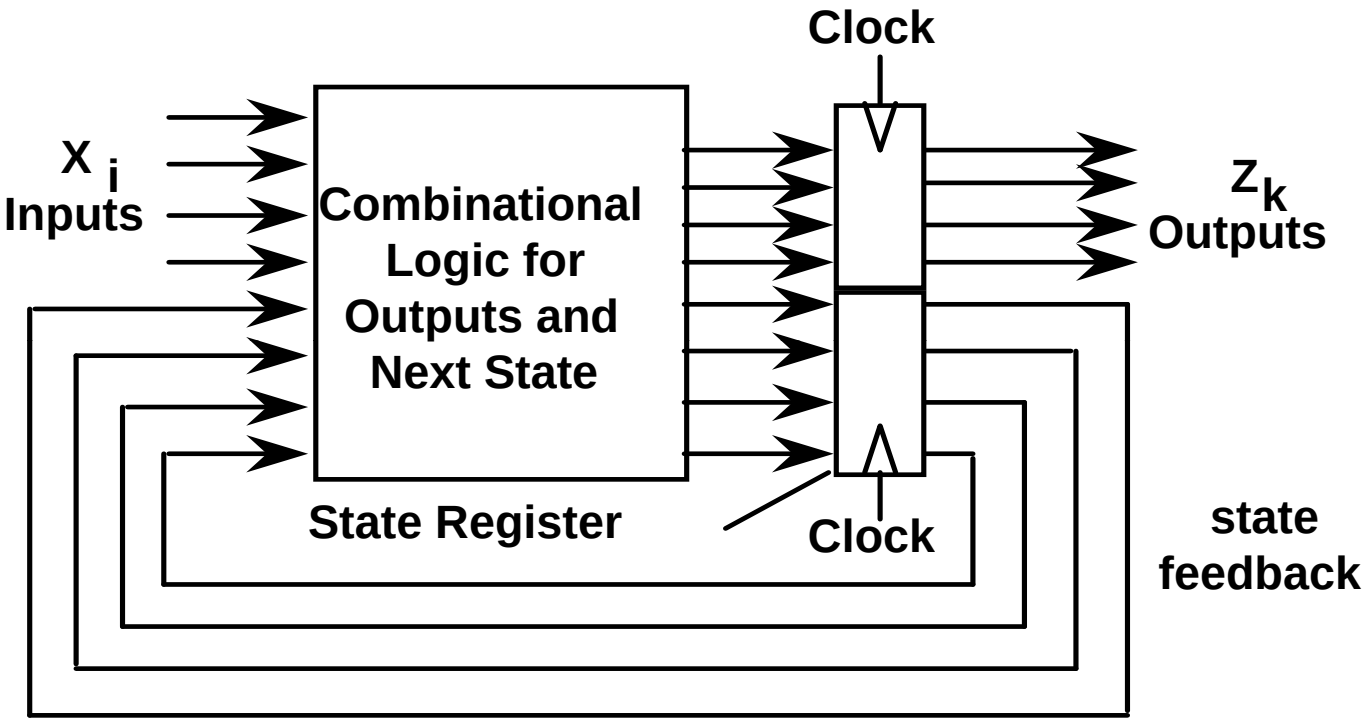
- **Moore**:
  - Generally **more** states required to solve a given problem
  - Easier to understand
  - Synchronous output (changes only with a clock pulse) -- The output is delayed in a Moore machine. Output does not occur until the next state change
  - Typically take more gates
  - Generally easier clocked (generally able to clock faster)
  - Easier to simulate using Max+Plus II
- **Mealy**:
  - Generally same or less states required
  - Slightly more complex to analyze
  - Asynchronous output (output can change any time an input changes) may lead to false outputs due to output changing after state changes
  - Generally requires less logic

Conclusion: Must know both, but learn Moore first

Synchronous Mealy

- Mealy model tend to has glitches in the output.
  - This is due to the asynchronous nature of the Mealy machine.
- Glitches are undesirable in real hardware controllers.
  - But because Mealy machines encode control in fewer states, saving on state register flip-flops, it is still desirable to use them.
- This leads to alternative synchronous design styles for Mealy machines.
- Simply stated, the way to construct a synchronous Mealy machine is to break the direct connection between inputs and outputs by introducing storage elements.

Synchronous Mealy Machine

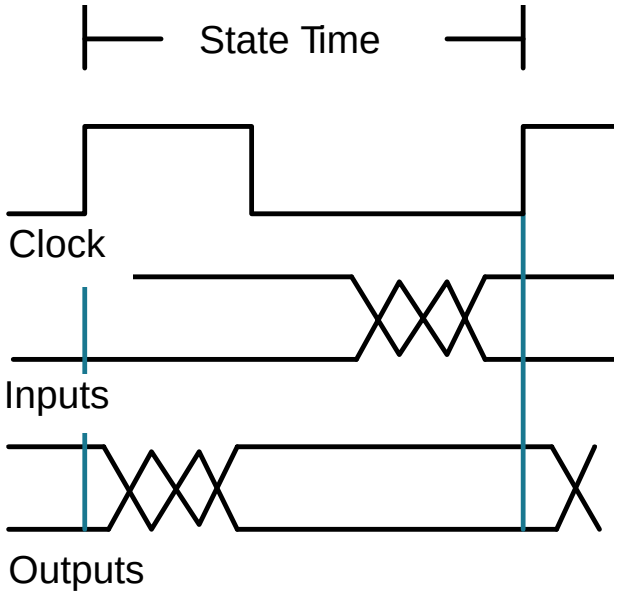


Combination of best ideas of Moore and Mealy:
Less logic + synchronous output

latched state AND outputs

avoids glitchy outputs!

State Machine Timing



- State Time:
  - Time between clocking events
- Clocking event:
  - inputs sampled
  - outputs, next state computed
- After propagation delay
  - outputs stable
  - next state entered
- Moore vs Mealy:
  - Asynchronous signals take effect immediately
  - Synchronous signals take effect at the next clocking event
- Immediate Outputs affect datapath immediately
- Delayed Outputs take effect on next clock edge
  - Important for synchronous Mealy
- For set-up/hold time considerations:
  - Inputs should be stable before clocking event

Basic Design Approach

- Eight Step Process (or just Six for this Week)
  1. Understand the statement of the Specification
  2. Draw a state diagram
  3. Convert state diagram to state table
  4. Optionally, perform state minimization
  5. Perform state assignment
  6. Obtain next state and output equations
  7. Optionally, choose a flip flop type other than DFF and derive the flip flop input maps or tables.
  8. Implement (Draw circuit realization, enter design & verify)

- 1-3 covered Week 9 & 11; 4, 5 & 7 covered Week 12;
- 6,7 generalized from the counter design procedure

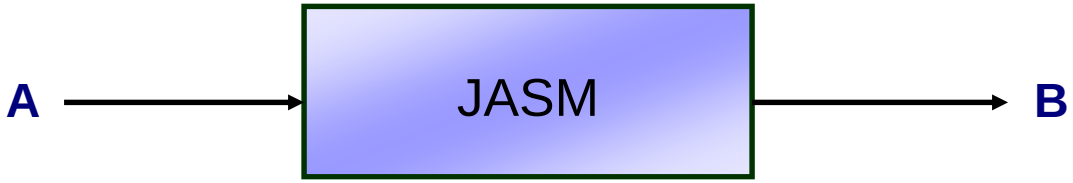
Example 1: JASM (Just Another State Machine)

- The specification:
  - An idle system is activated when an input, **A** is given. Then, an output, **B** is produced after two interval time or cycles later. Next, the system will be back to the idle state, waiting for the next triggering input **A** .

- Step 1: Understand the specs.
  - Get a sample input/output relationship. More may be needed later.
    - Sample input/output relationship:

      A : 001001110

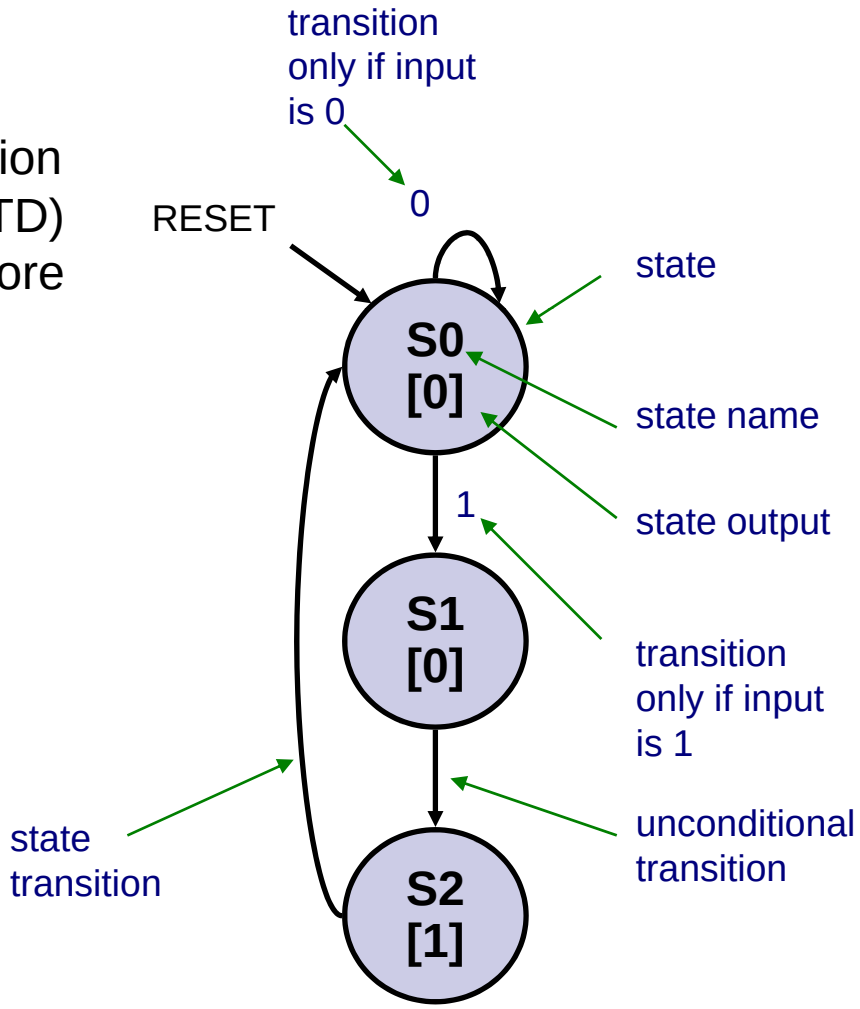
      B : 000010010
  - Draw a simple block diagram.



JASM State Transition Diagram

Step 2: Draw State diagram

Some call it state transition diagram (STD) Choose Moore or Mealy



- Highlights:
  - An **oval** represents a condition or state
  - The state name and output is written inside the state
  - An **arc** or arrow represents a transition from a state to another state
  - An arrow is labeled if a certain is applied for the transition to occur

JASM Symbolic State Table

- Step 3: get symbolic state table.
  - To proceed to logic design, the state diagram is converted to a state table.
  - There are 3 different states denoted by S0, S1 and S2.
  - A symbolic state table uses state names, as used in the state diagram.

Present State	Input A	Next State	Output B	Comments
S0	0	S0	0	Remain in idle state if input does not change
	1	S1		Go to next state if input is 1
S1	0	S2	0	Go to next state no matter what is the input.
	1	S2		
S2	0	S0	1	Go to S0 no matter what is the input. Output is high in this state.
	10	S0		

- Step 4: Perform state minimization:
  - Not necessary here… too few states already. But will be needed later.

JASM Encoded State Table

- Step 5: Perform state assignment:
  - Use of "simple" binary encoding gives us: S0 = 00, S1 = 01 and S2 = 10.
  - Must also add in code 11 to take care of don't cares.
  - Here, if we somehow get to state 11, next state & output are don't cares.

Present State	Input	Next State	Output
PS_1PS_0	A	NS_1NS_0	B
00	0	00	0
	1	01	
01	0	10	0
	1	10	
10	0	00	1
	1	00	
11	0	11	X
	1	11	

Alternative State Assignments

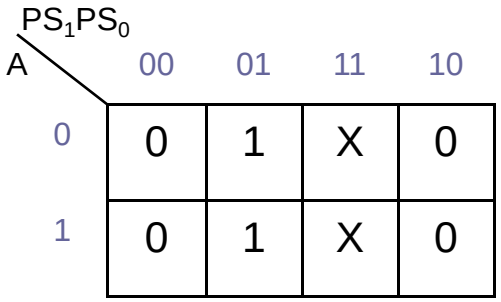
NO	Simple	Gray	Johnson	One-Hot	Almost One-hot
0	000	000	0000	00000001	0000000
1	001	001	0001	00000010	0000001
2	010	011	0011	00000100	0000010
3	011	010	0111	00001000	0000100
4	100	110	1111	00010000	0001000
5	101	111	1110	00100000	0010000
6	110	101	1100	01000000	0100000
7	111	100	1000	10000000	1000000

- We'll use simple state assignment for this week.

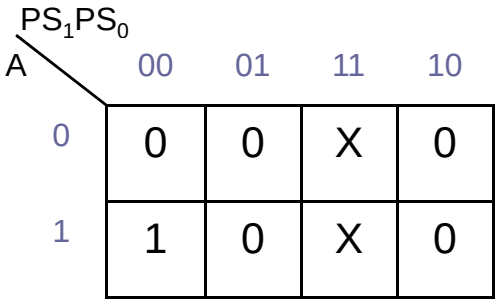
Get Logic Equations

- Step 6: Solve the next state & output equations.

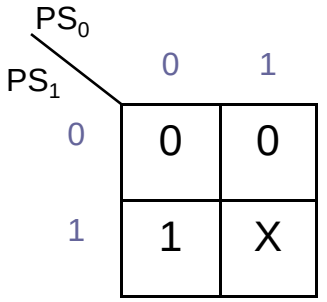
Present State		Input	Next State		Output
PS_1	PS_0	A	NS_1	NS_0	B
0	0	0	0	0	0
0	0	1	0	1	
0	1	0	1	0	0
0	1	1	1	0	
1	0	0	0	0	1
1	0	1	0	0	
1	1	0	X	X	X
1	1	1	X	X	



$NS_1 = PS_0$

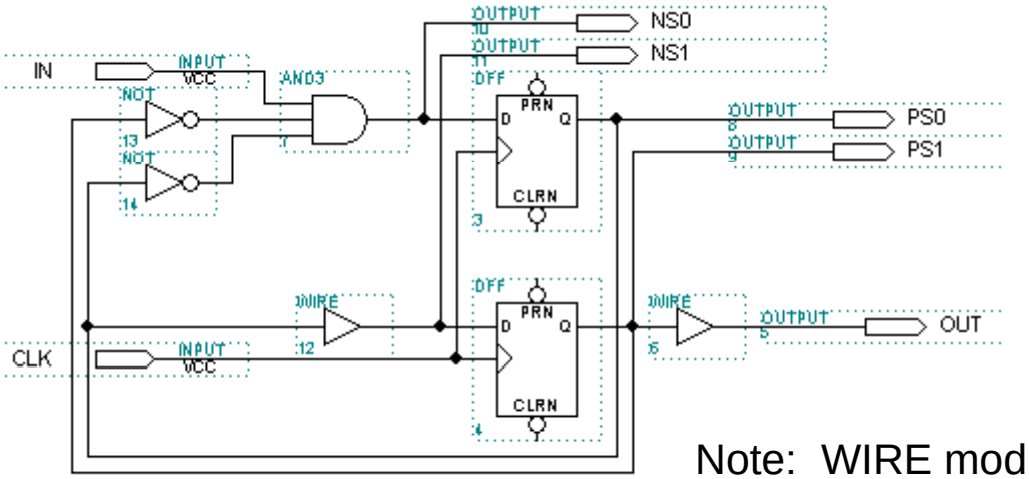


$NS_0 = PS_1' \bullet PS_0' \bullet A$

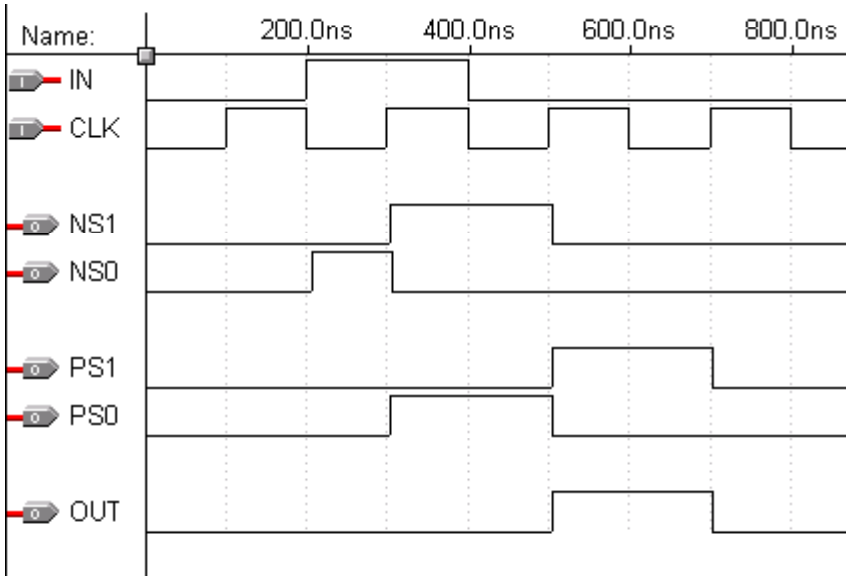


$NS_0 = PS_1$

Moore Model Implementation of JASM



Note: WIRE module has no effect on logic



Example 2: Bit Sequence Detector (BSD)

- The specification:
  - An input is used to detect a sequence or a series of inputs, 110. When the specific sequence is detected, an output high is produced for a cycle. Then, the system will continue detect for the next sequence inputs.
- Motivation
  - The sequence detector circuit has a practical application in code encoding and decoding such as Huffman Codes

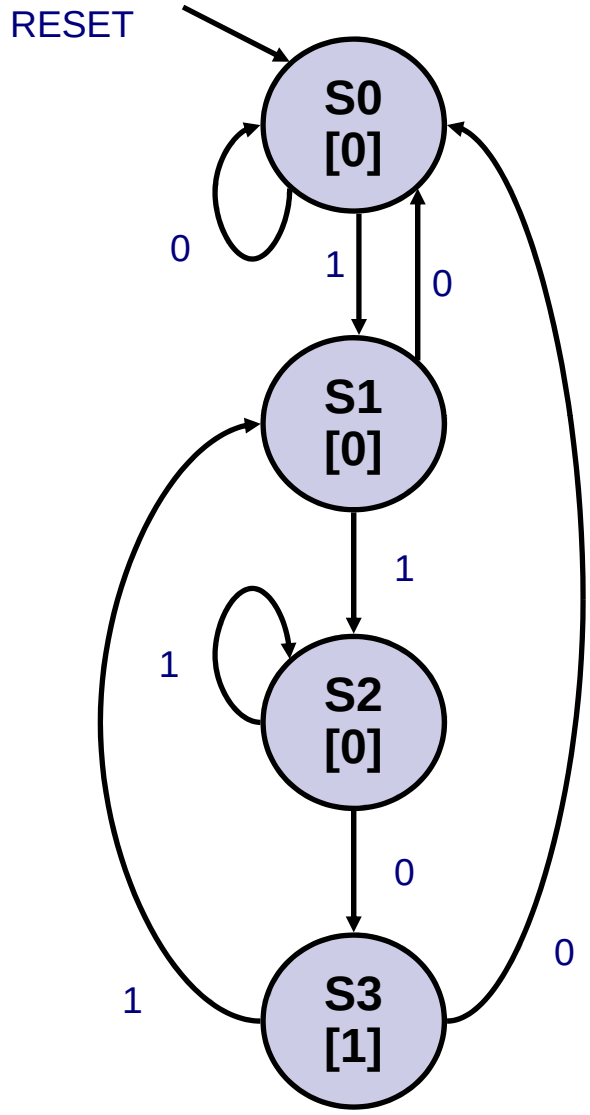
- Step 1: Understand the specs.
  - Get a sample input/output relationship.
    - Sample input/output relationship:

      IN : 1100011011110…

      OUT : 0010000100001…

110 BSD State Diagram

- Step 2: Get state diagram
  - Start with the expected sequence first
  - S0 means 0 bit found, S1 = 1 bit found, and so on
  - In S3, all three bits have been detected and output becomes 1
  - After completing S0-S1-S2-S3 transitions, add all remaining arrows.



BSD Symbolic State Transition Table

Step 3: Symbolic state table

Present States	Input	Next States	Output	Comments
	IN		OUT	
S0	0	S0	0	Remain in idle state if start sequence is not detected
	1	S1		Go to next state if start sequence is detected
S1	0	S0	0	Go back to starting state if wrong sequence
	1	S2		Go to next state if correct sequence is detected
S2	0	S3	0	Complete sequence is detected
	1	S2		Sequence is not completed yet, wait until ‘0’ appear
S3	0	S0	1	Go back to starting state if start sequence is wrong
	1	S1		Go to next sequence if start sequence is correct

Step 4: State table minimization --> not necessary

BSD Encoded State Table

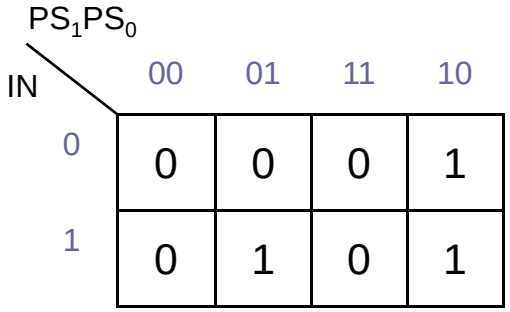
- Step 5: Perform state assignment:
  - Use “simple” binary encoding:
    - S0 = 00
    - S1 = 01
    - S2 = 10
    - S3 = 11

Present State		Input	Next State		Output
PS_1	PS_0	IN	NS_1	NS_0	OUT
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	1	1	0
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	0	1	1

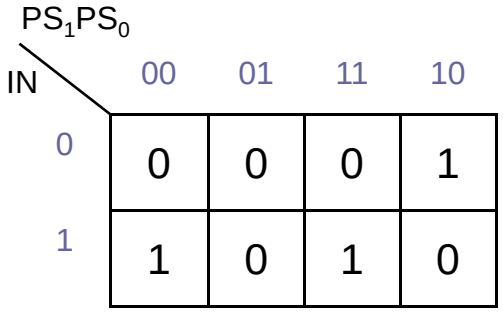
BSD Next State & Output Equations

Step 6:

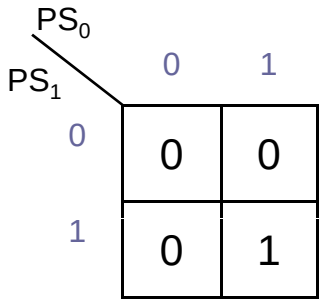
Present State		Input	Next State		Output
PS_1	PS_0	IN	NS_1	NS_0	OUT
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	1	1	0
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	0	1	1



$NS1 = PS_1 \bullet PS_0' + PS_1' \bullet PS_0 \bullet IN$



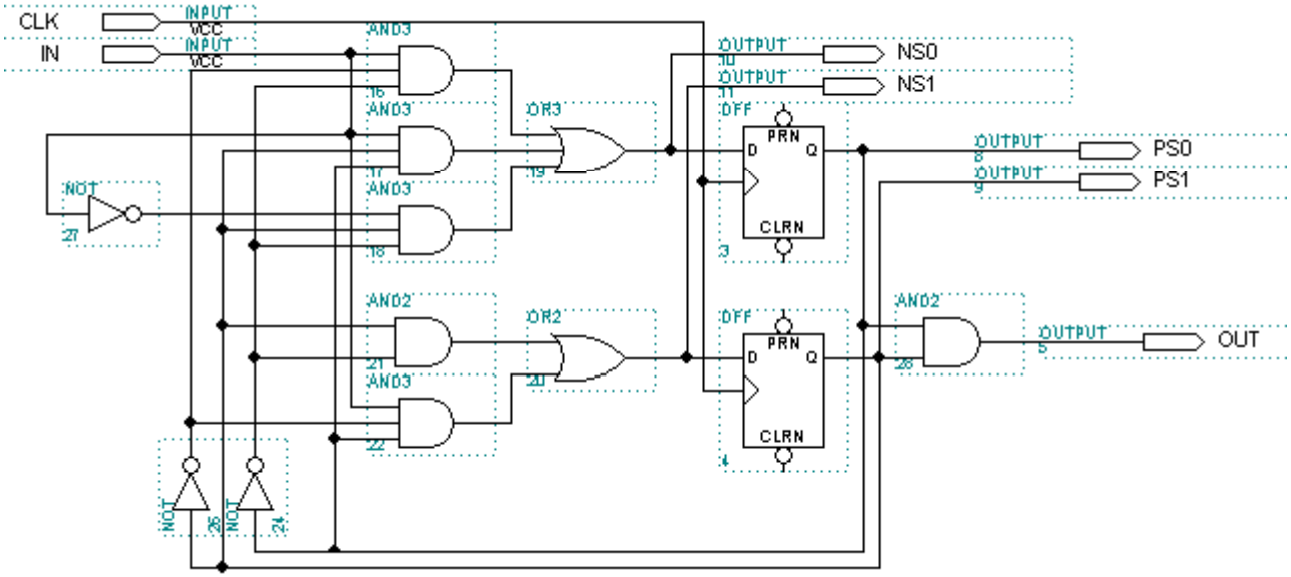
$NS0 = PS_1' \bullet PS_0' \bullet IN + PS_1 \bullet PS_0' \bullet IN' + PS_1 \bullet PS_0 \bullet IN$

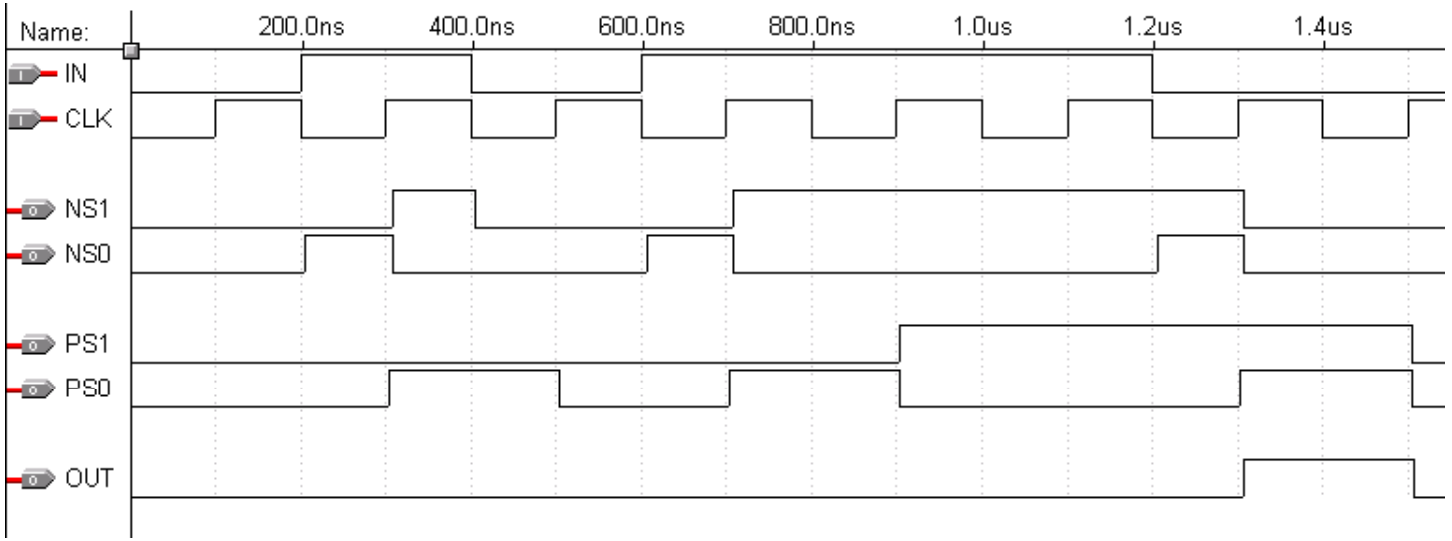


$B = PS_1\, PS_0$

BSD Moore Circuit

Step
(7,) 8:





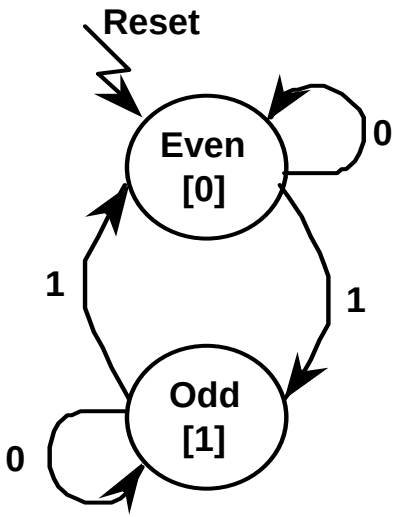
Example 3: Odd Parity Checker

- **The specification:**
  - Assert output whenever input bit stream has odd # of 1's

- **Step 1: Understand the specs.**
  - Get a sample input/output relationship.
    - A : 0
    - B : 1 1 because 0 (even) # of 1's detected
    - A : 01
    - B : 0 0 because 1 (odd) # of 1's detected
    - A : 011
    - B : 1 1 because 2 (even) # of 1's detected
    - A : 0110
    - B : 1 …ditto… (same as above)
    - A : 01101
    - B : 0 0 because 3 (odd) # of 1's
    - A : 011010
    - B : 0 …ditto…

Odd Parity Checker

- Steps 2,3,(4,)5: State Diagram, symbolic state table, (minimization) & encoded state table



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

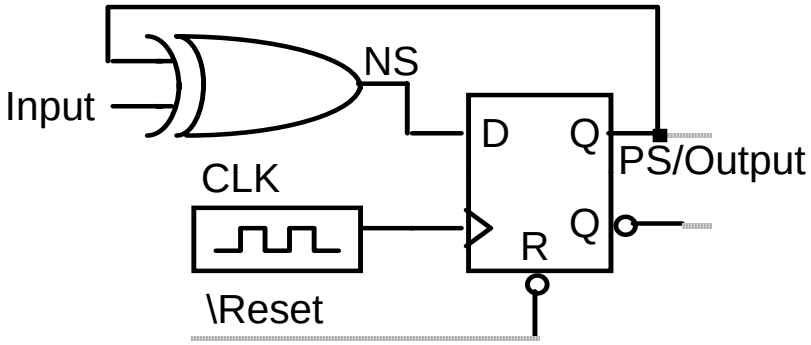
Encoded State Transition Table

Step 6: Next state & Output Equations

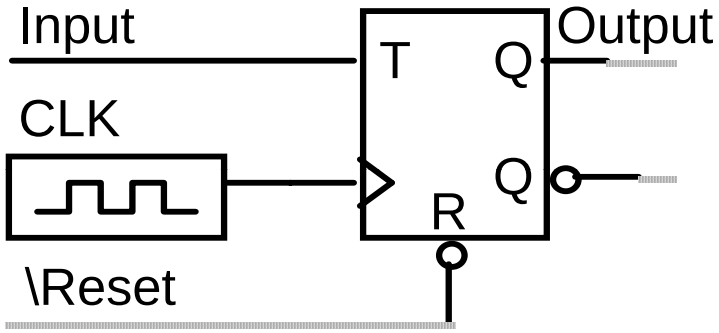
NS = PS xor PI; OUT = PS

Odd Parity Checker

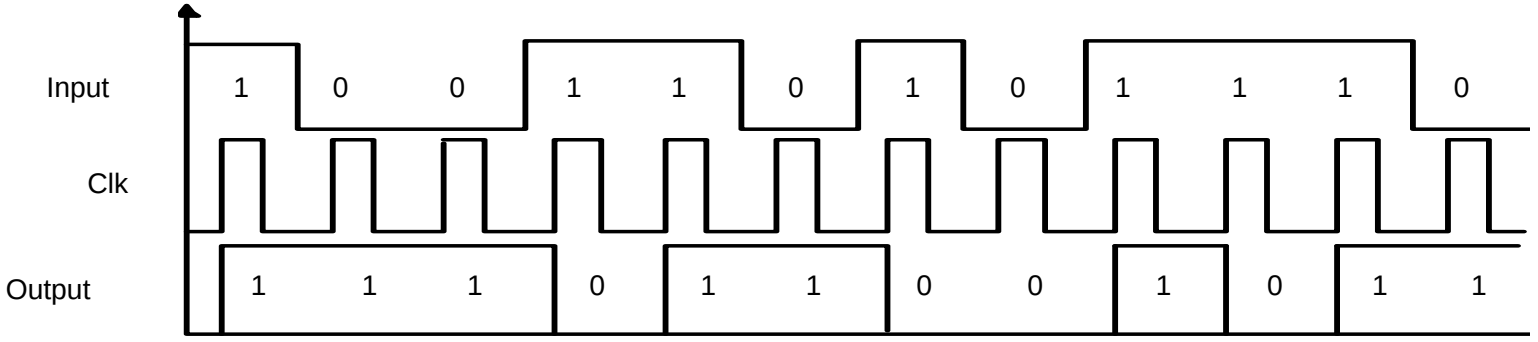
Steps 7 & 8: Implementation (DFF & TFF)



D FF Implementation



T FF Implementation



Timing Behavior: Input 1 0 0 1 1 0 1 0 1 1 1 0

Example 4: Dual-Mode Counter

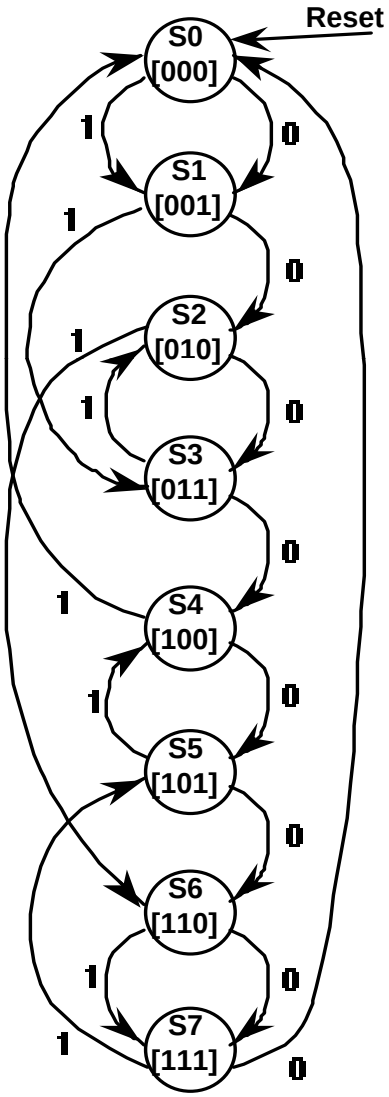
- A sync. 3 bit counter has a mode control M. When M = 0, the counter counts up in the binary sequence. When M = 1, the counter advances through the Gray code sequence.

Step 1

- List possible sequences to understand the problem.
  - Binary: 000, 001, 010, 011, 100, 101, 110, 111
  - Gray: 000, 001, 011, 010, 110, 111, 101, 100

Mode Input M	Current State	Next State (CBA)
0	000	001
0	001	010
1	010	110
1	110	111
1	111	101
0	101	110
0	110	111

Dual-Mode Counter



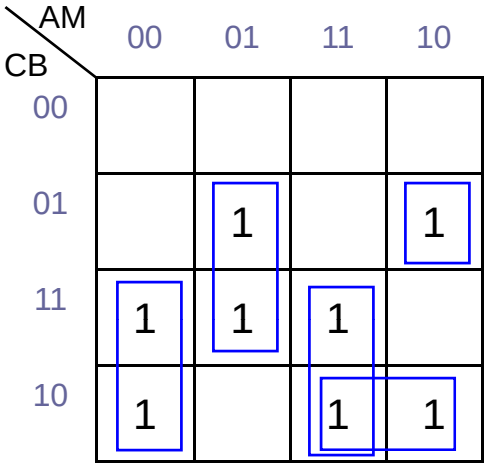
Step 2

One state for each output combination
Add appropriate arcs for the mode control

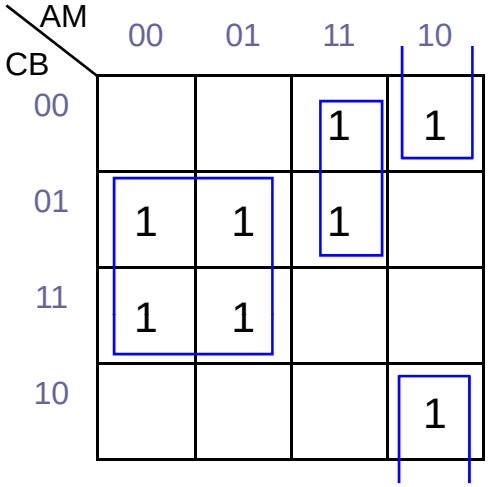
Step 3,4,5

Present State			Input	Next State		
C	B	A	M	DC	DB	DA
0	0	0	0	0	0	1
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	1	1
0	1	0	1	1	1	0
0	1	1	0	1	0	0
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	0	1	0	0	0
1	0	1	0	1	1	0
1	0	1	1	1	0	0
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	0	0	0
1	1	1	1	1	0	1

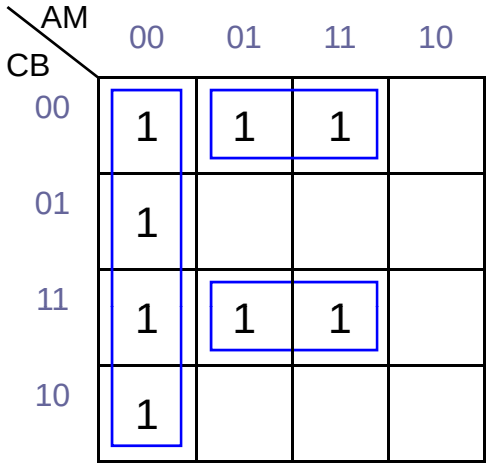
Dual-Mode Counter



DC = CA'M' + BA'M + CAM + CB'A + C'BAM'

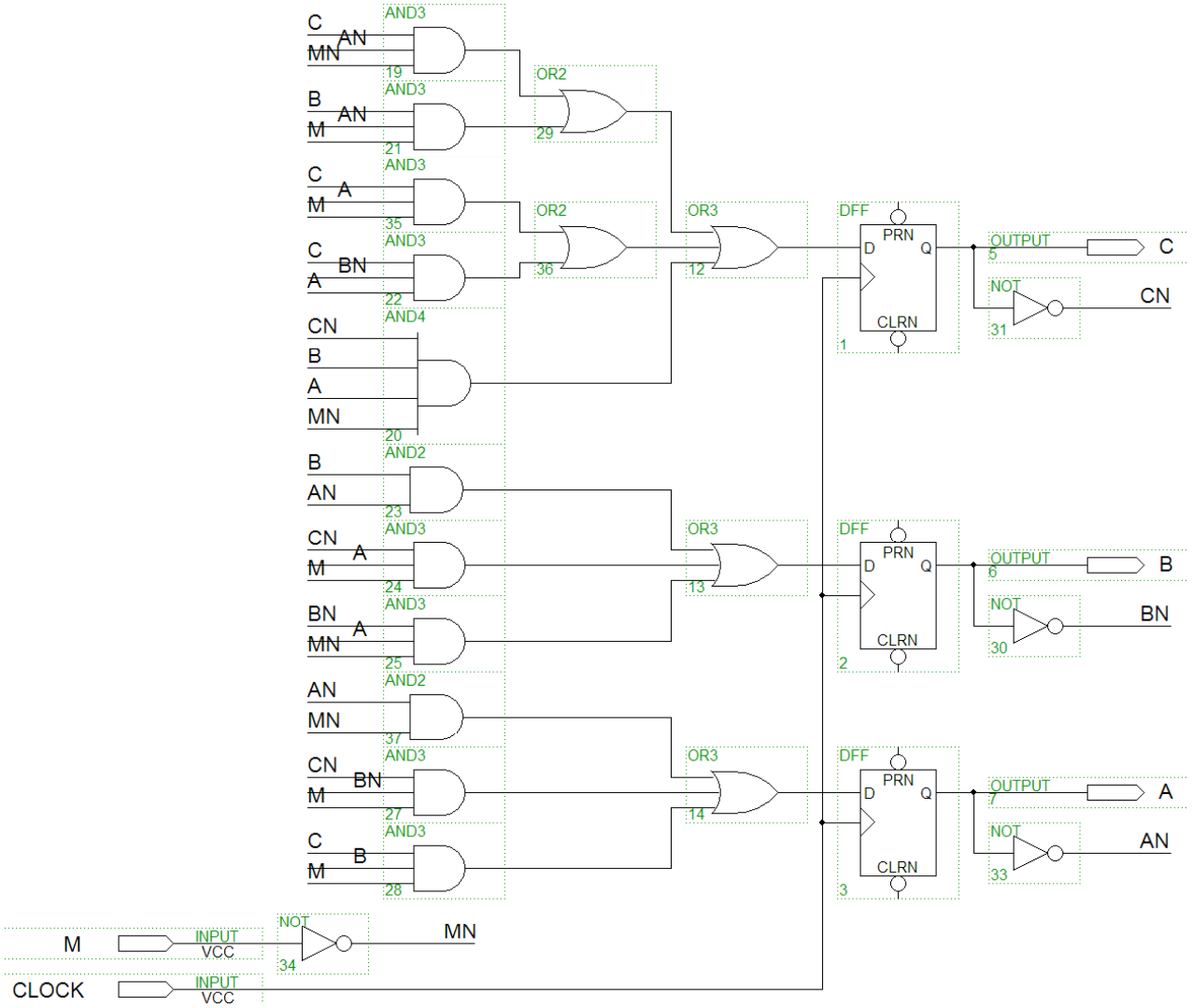


DC = BA' + C'AM + B'AM'



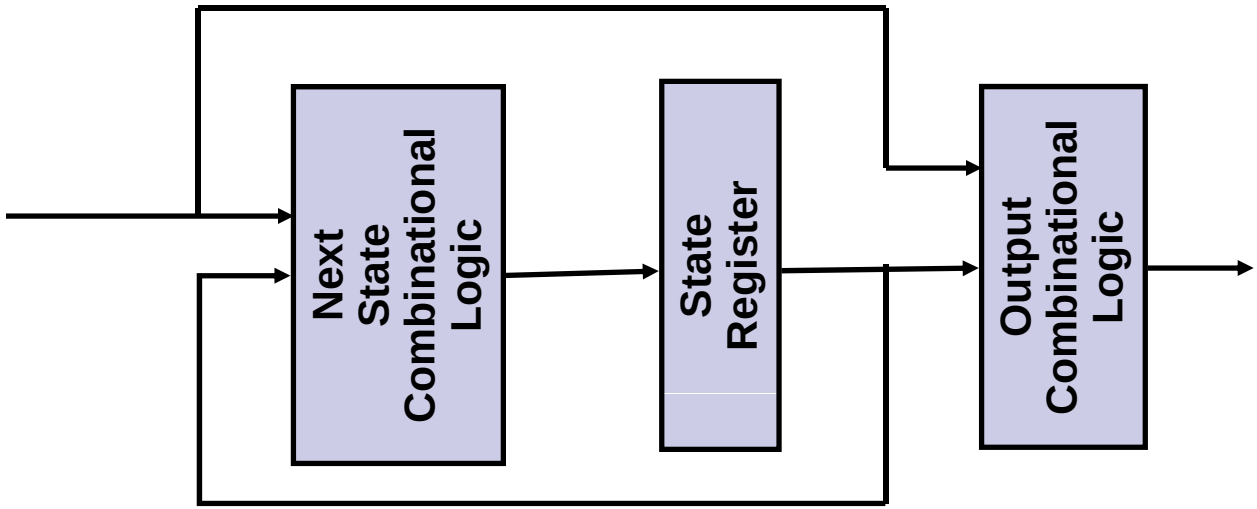
DC = A'M' + C'B'M + CBM

Dual-mode Counter Circuit



JASM Using Mealy Model

- The specifications (still remember?):
  - An idle system is activated when an input, **A** is given. Then, an output, **B** is produced after two interval time or cycles later. Next, the system will be back to the idle state, waiting for the next triggering input **A**.
- Step 1: Understand the specs.
  - Been there, done that!
  - Another view of Mealy Model. Notice: output = f(input,present state)

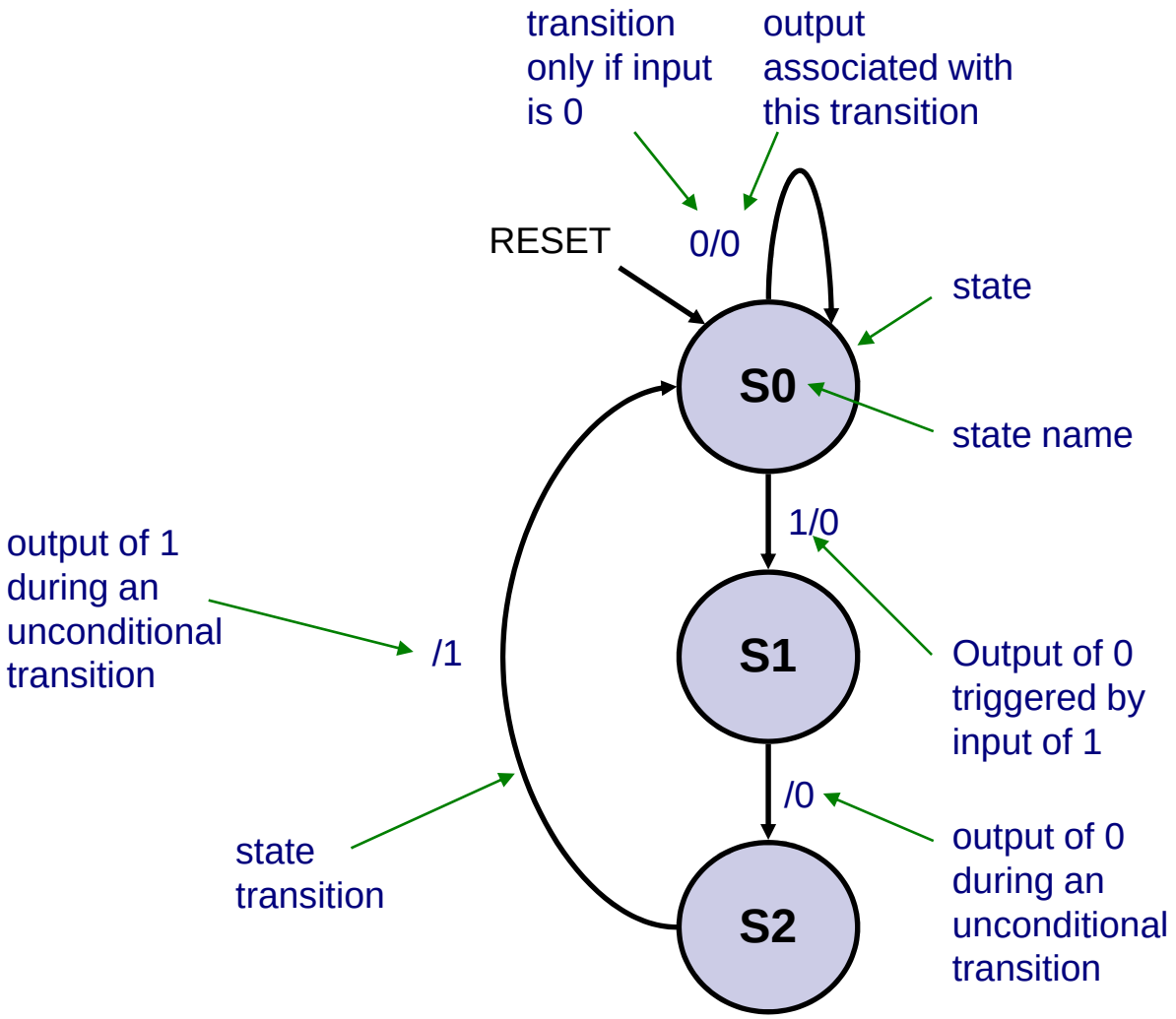


JASM Mealy State Transition Diagram

Step 2: Draw State diagram

Mealy state diagram is slightly different than Moore

Outputs are associated with state transitions (**arcs)** instead of state



JASM Symbolic State Table

- Step 3: get symbolic state table.
  - Now output is a function of both present state and input.

Present State	Input	Next State	Output
	A		B
S0	0	S0	0
	1	S1	0
S1	0	S2	0
	1	S2	0
S2	0	S0	1
	1	S0	1

Step 4: Perform state minimization.

Not necessary here… yet

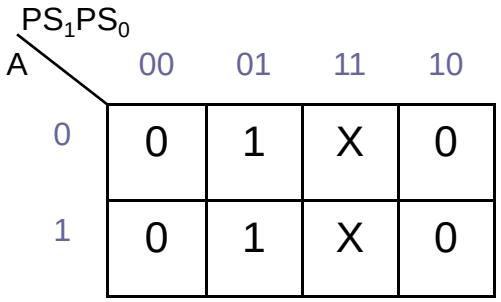
Step 5: Get encoded state table.

Present State	Input	Next State	Output
PS_1PS_0	A	NS_1NS_0	B
00	0	00	0
	1	01	
01	0	10	0
	1	10	
10	0	00	1
	1	00	
11	0	XX	X
	1	XX	

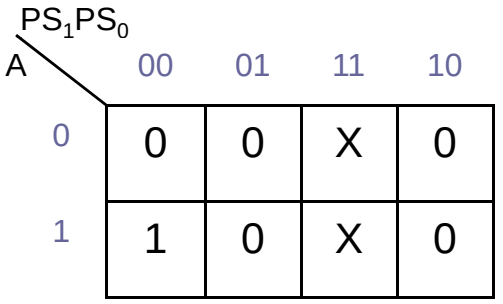
Get Logic Equations

- Step 6: Solve the next state & output equations. Remember output is a function of both present state and input.
- Step 6: Skip because we're using DFF
- Step 7: Enter & simulate in MaxPlus as exercise

Present State		Input	Next State		Output
PS_1	PS_0	A	NS_1	NS_0	B
0	0	0	0	0	0
0	0	1	0	1	
0	1	0	1	0	0
0	1	1	1	0	
1	0	0	0	0	1
1	0	1	0	0	
1	1	0	X	X	X
1	1	1	X	X	

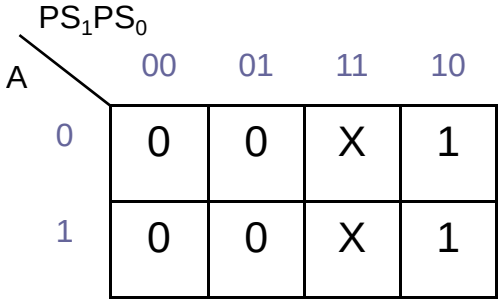


$NS_1 = PS_0$



PS_0 PS_1

$NS_0 = PS_1' \bullet PS_0' \bullet A$



$NS_0 = PS_1$

Example 2: Bit Sequence Detector (BSD)

- The specification:
  - An input is used to detect a sequence or a series of inputs, 110. When the specific sequence is detected, an output high is produced for a cycle. Then, the system will continue detect for the next sequence inputs.
- Motivation
  - The sequence detector circuit has a practical application in code encoding and decoding such as Huffman Codes

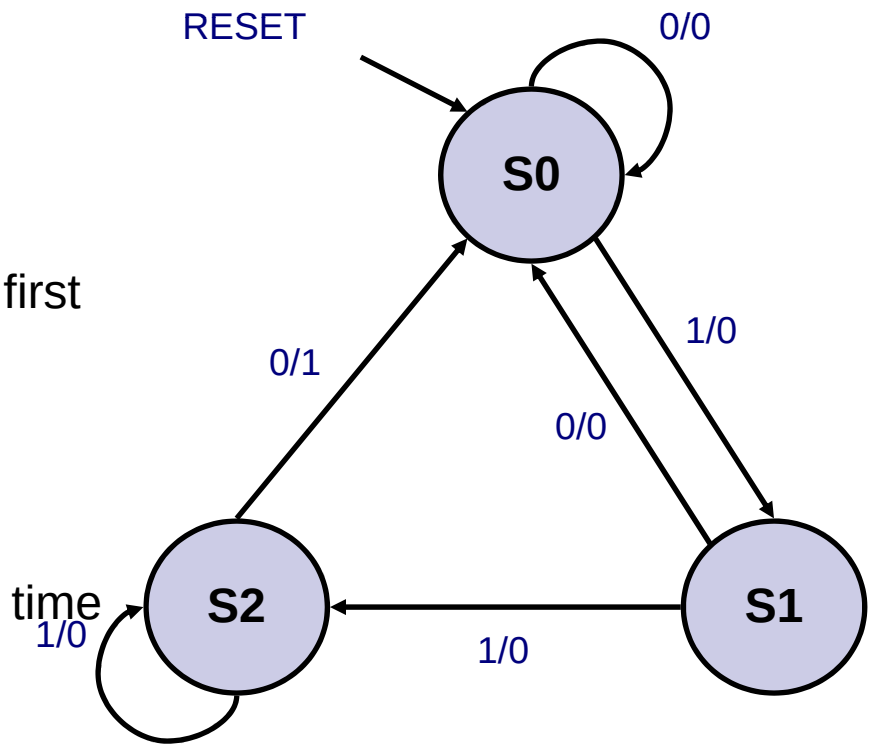
- Step 1: Understand the specs.
  - Get a sample input/output relationship.
    - Sample input/output relationship:

      A : 1100011011110…

      B : 0010000100001…

110 BSD State Diagram

- The specification:
  - An input is used to detect a sequence or a series of inputs, 110. When the specific sequence is detected, an output high is produced for a cycle. Then, the system will continue detect for the next sequence inputs.
- Step 1: Understand the specs.
  - Done. We've seen this circuit before.
- Step 2: Get state diagram
  - Start with the expected sequence first
  - S0 means 0 bit found, S1 = 1 bit found, S2 = 2 bits found
  - If all the third bit is detected (110 sequence completed) while in S2, reset (go to S0) while at the same time outputting a 1



BSD Symbolic State Transition Table

Step 3: Symbolic state table

Present States	Input	Next States	Output	Comments
	A		B	
S0	0	S0	0	Remain in idle state if start sequence is not detected
	1	S1	0	Go to next state if first bit is detected
S1	0	S0	0	Go back to starting state if wrong sequence
	1	S2	0	Go to next state if second bit is detected
S2	0	S0	1	Complete sequence is detected, reset & output 1
	1	S2	0	Sequence is not completed yet, wait until '0' appears

Step 4: State table minimization --> not necessary

BSD Encoded State Table

- Step 5: Perform state assignment:
  - Use “simple” binary encoding:
    - S0 = 00
    - S1 = 01
    - S2 = 10

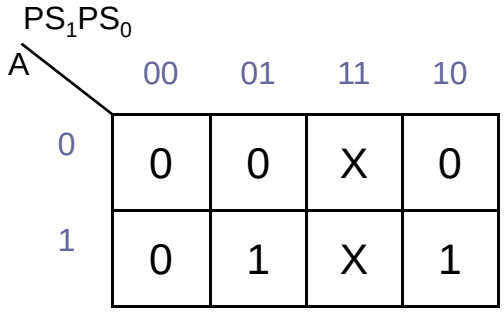
Present State		Input	Next State		Output
PS_1	PS_0	A	NS_1	NS_0	B
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	X	X	X
1	1	1	X	X	X

BSD Next State & Output Equations

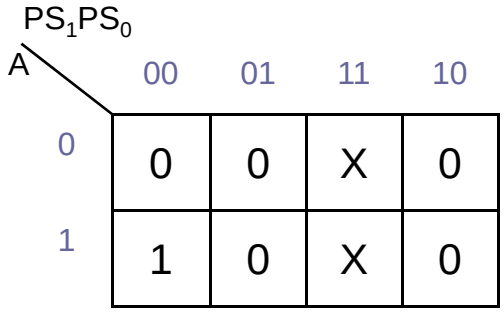
Step 6:

Present State		Input	Next State		Output
PS_1	PS_0	A	NS_1	NS_0	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	1
1	0	1	1	0	0
1	1	0	X	X	X
1	1	1	X	X	X

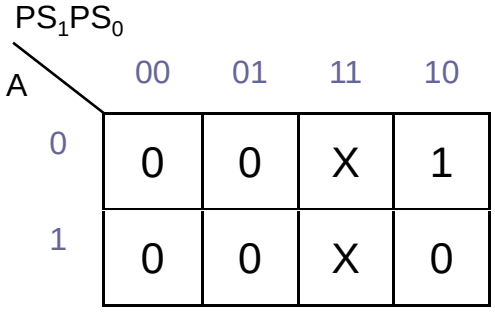
Simpler logic compared to Moore version!
Step7 & 8 : Circuit diagram left as an exercise…



$NS1 = PS_1 \bullet A + PS_0 \bullet A$



$NS0 = PS_1' \bullet PS_0' \bullet A$



$B = PS_1 \bullet A'$