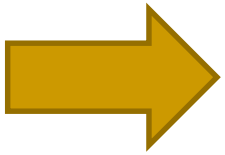
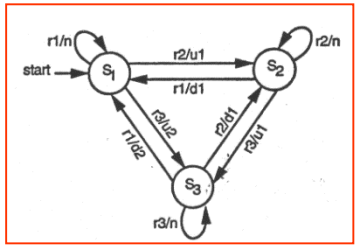
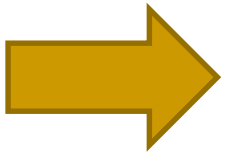
State Machine Models

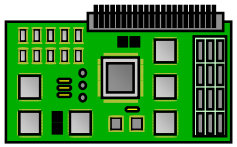
State Machine Modeling

- 1) For Design (optimization)



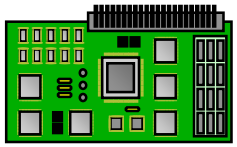


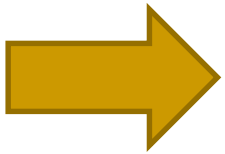


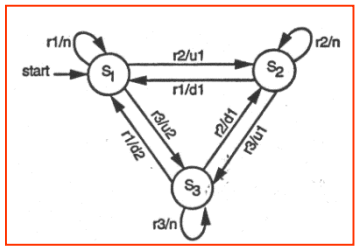


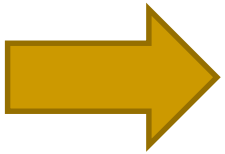
State Machine Modeling

- 2) For analysis and verification



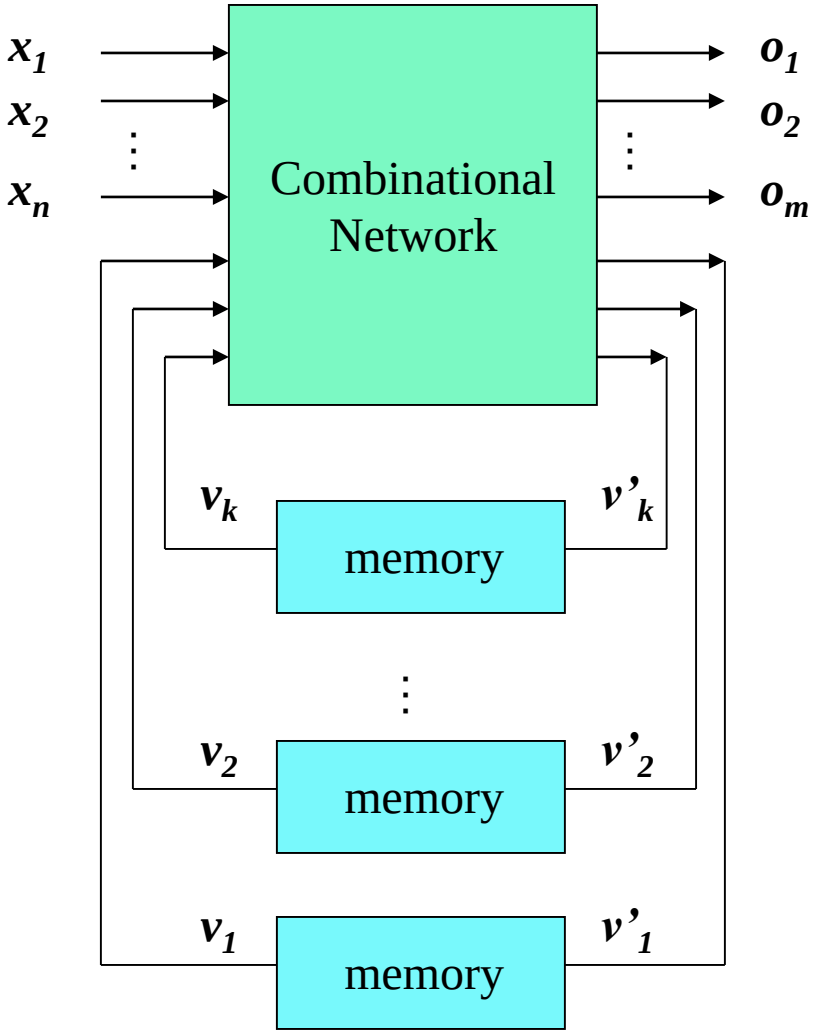






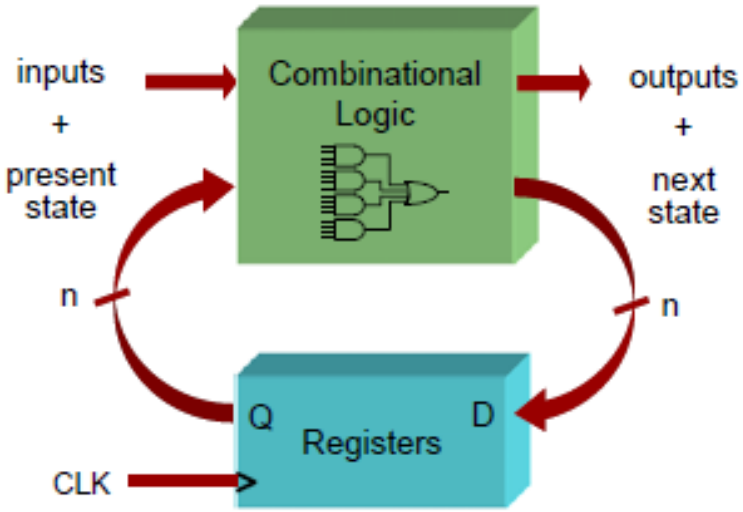
Sequential Systems

- M = (I, S, δ, λ, O, S_0) where
  - I: set of inputs
  - S: set of states
  - O: set of outputs
  - δ: transition function ($S \times I \rightarrow S$)
  - λ: output function
    - Mealy: $\lambda \in S \times I \rightarrow O$
    - Moore: $\lambda \in S \rightarrow O$
  - S_0: set of initial states
  - Deterministic machines: δ and λ are functions.



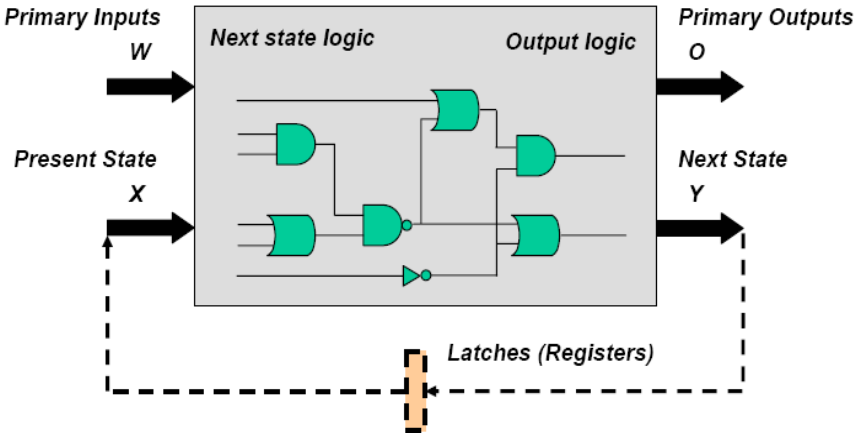
Finite State Machines

- An FSM is an effective means of implementing control functions.
- An FSM works in two phases:
  - The new state is calculated
  - The new state is sampled into a register
- The key point:
  - The maximum time to calculate the next state
  - Determine the highest clock frequency that the FSM can cope with



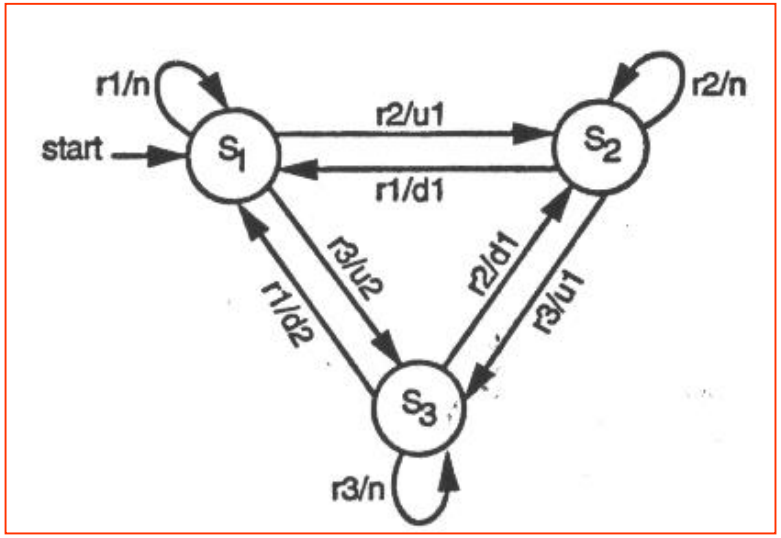
Finite State Machines

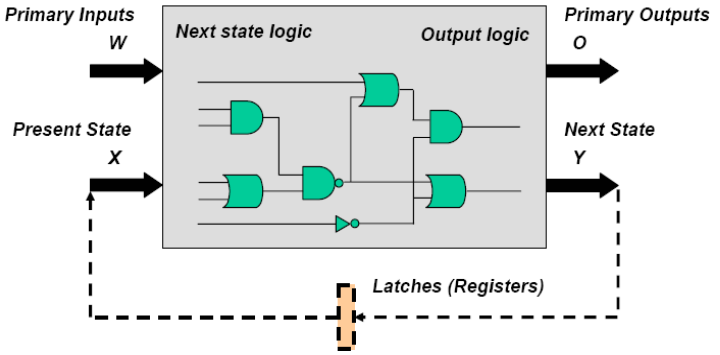
- Two basic types of FSM
  - Difference lies in the outputs
- Mealy machine: $\lambda \in S \times I \rightarrow O$
  - It's outputs are a function of the present state and all the inputs.
- Moore machine: $\lambda \in S \rightarrow O$
  - It's outputs are a function of the present state only.



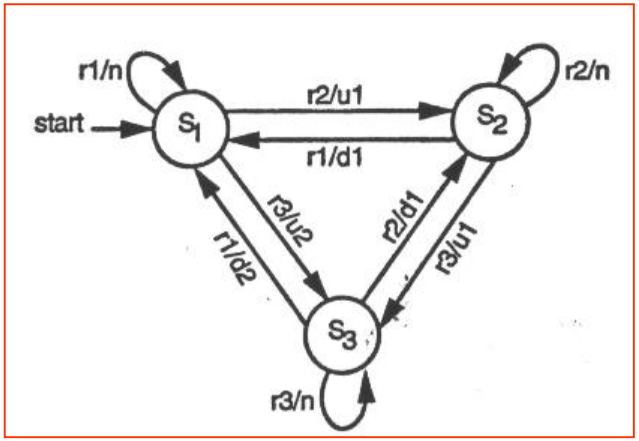
Example: Elevator Controller

- Elevator Controller with three floors
- I={r1, r2, r3}: floor requests
- O={d2, d1, n, u1, u2}: instructions for the elevator:
  - di: go-down i floors
  - ui: go-up i floors

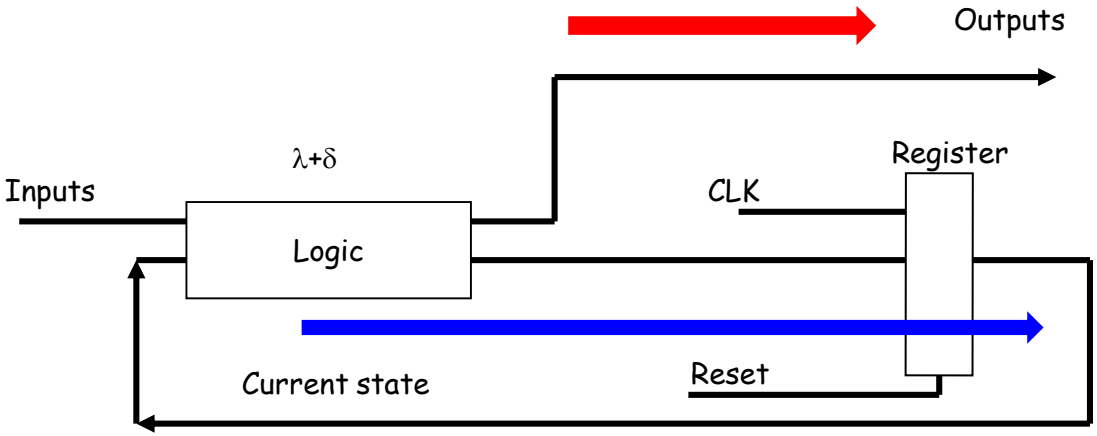




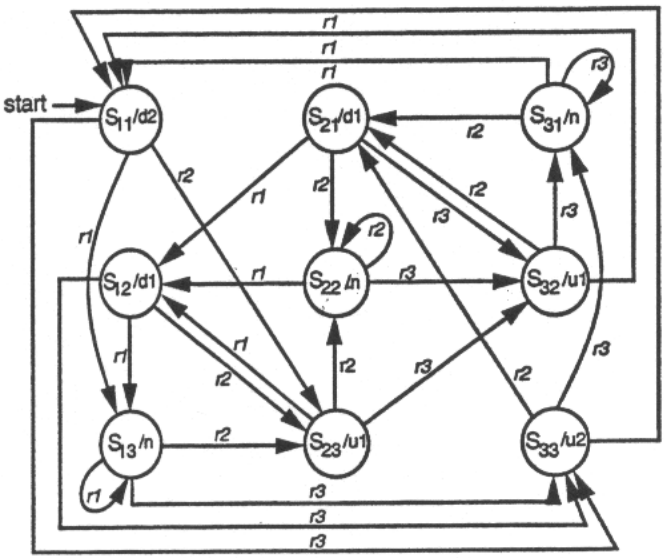
Transition-Based FSM



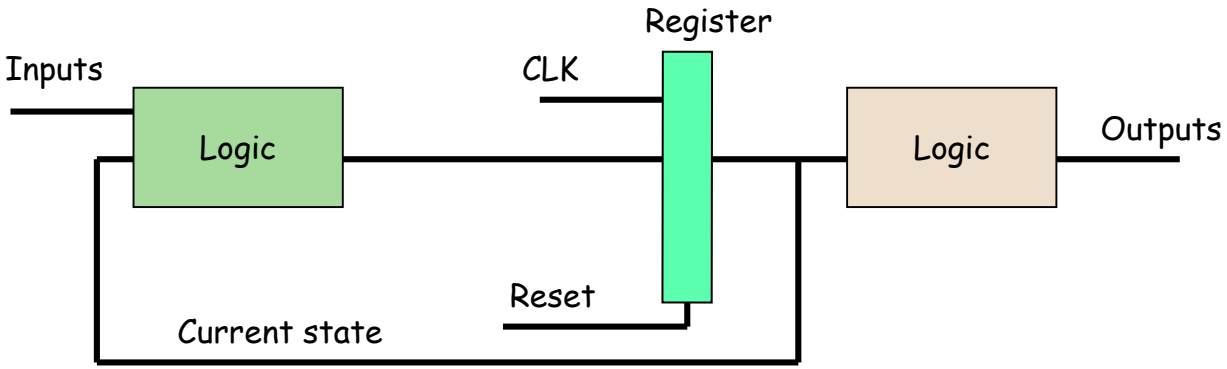
- Reactive
  - Zero response time
- Hard to compose:
  - Problem with combinational cycles
- Problematic for implementation



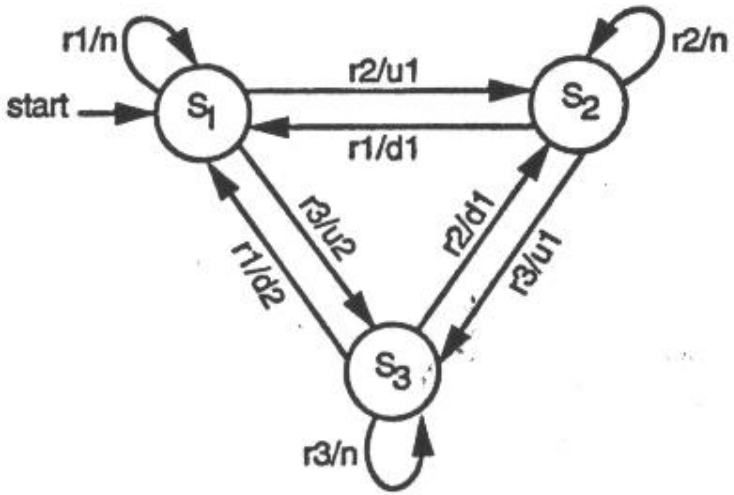
State-Based FSM

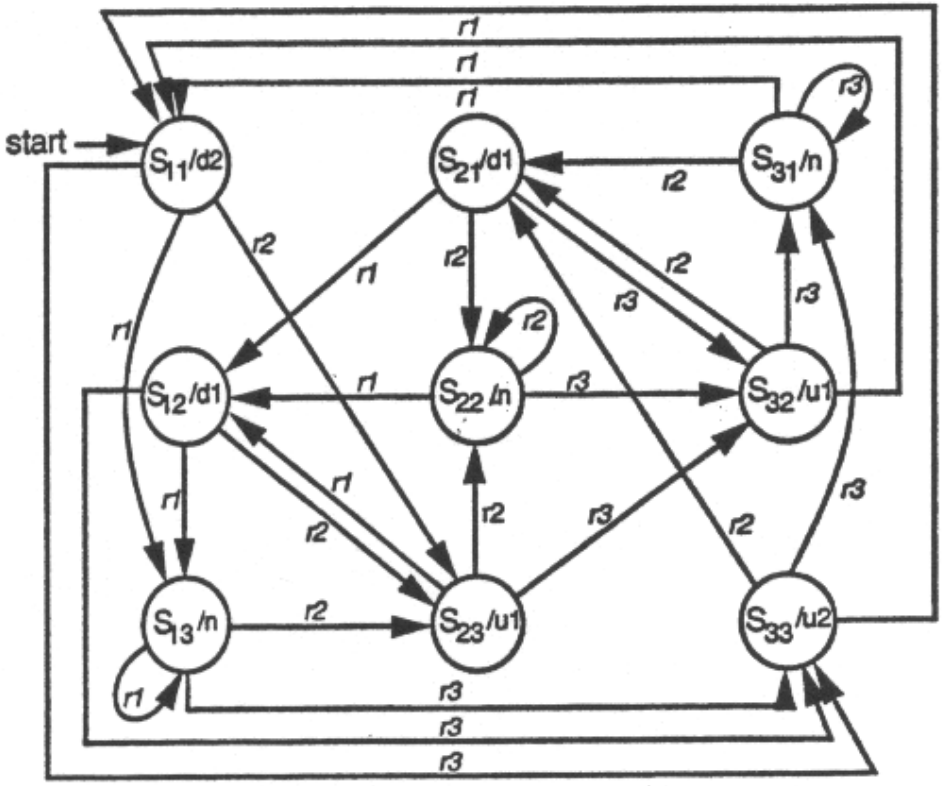


- Non-reactive
  - response delayed by one cycle
- Easy to compose: always well-defined
- Good for implementation
- Ant brain is a Moore Machine
  - Output does not react immediately to input change



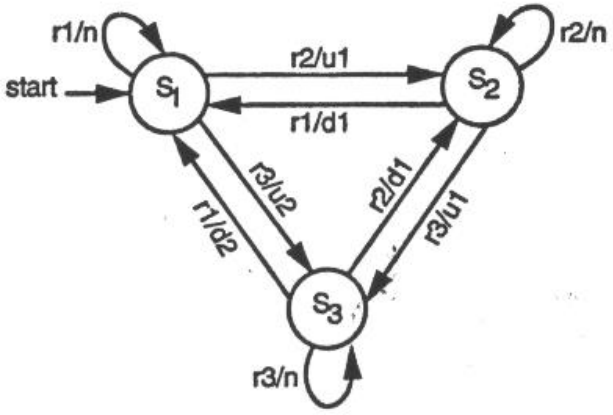
Mealy → Moore Machines

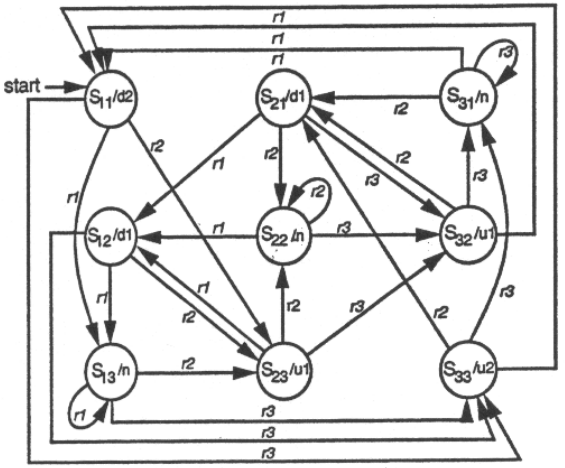




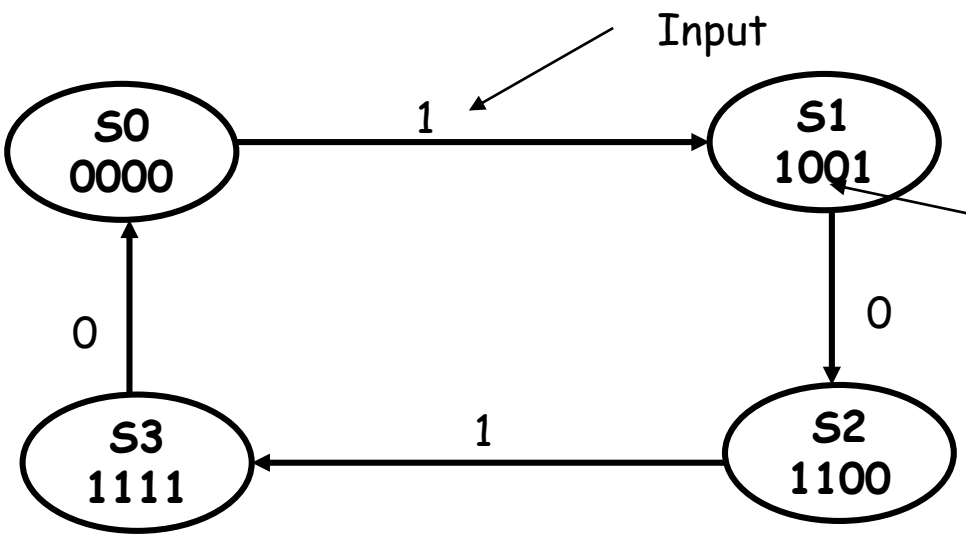
Finite State Machines

- A Mealy machine always works one clock cycle in advance of a Moore machine
  - A Mealy machine changes its outputs immediately when the inputs change.
  - A Moore machine has to change state first:
    - Wait a clock cycle before the output values can be changed.

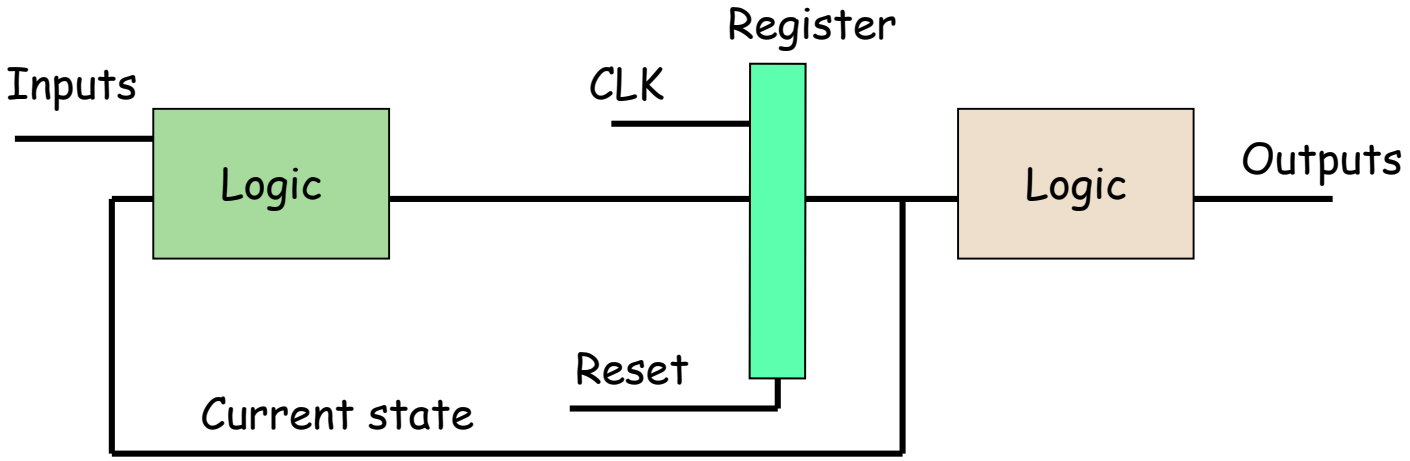




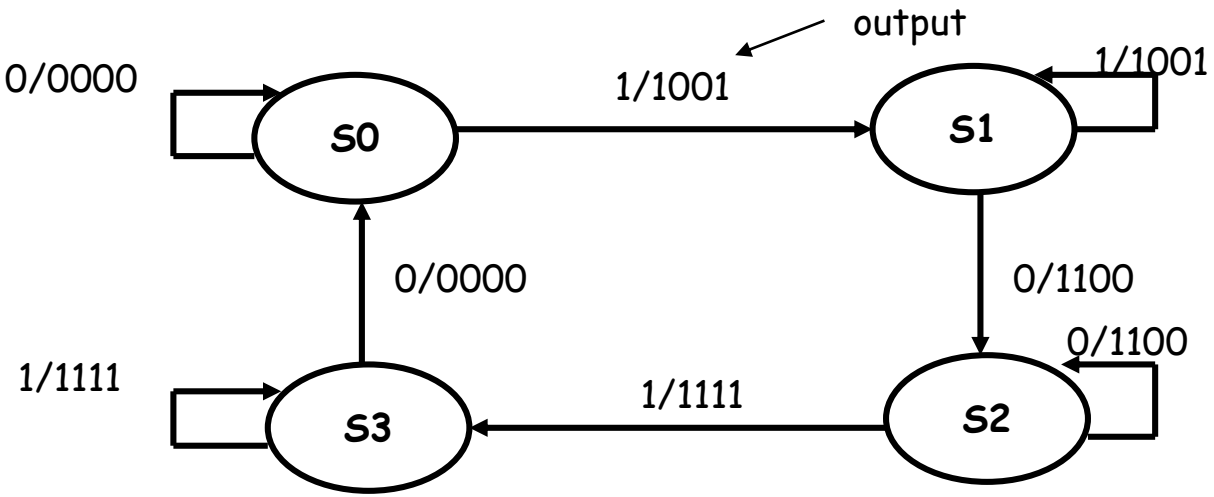
Moore Machine

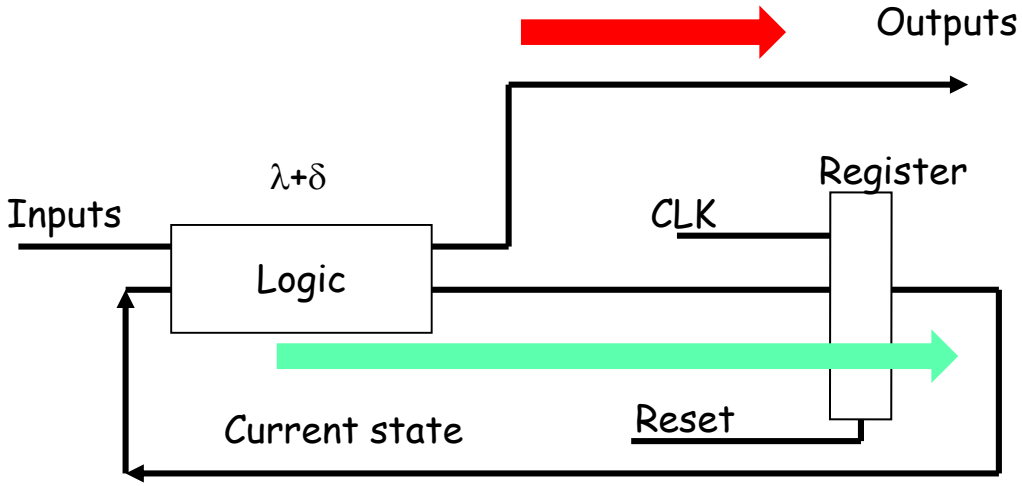


which part of the FSM will be affected by state encoding?

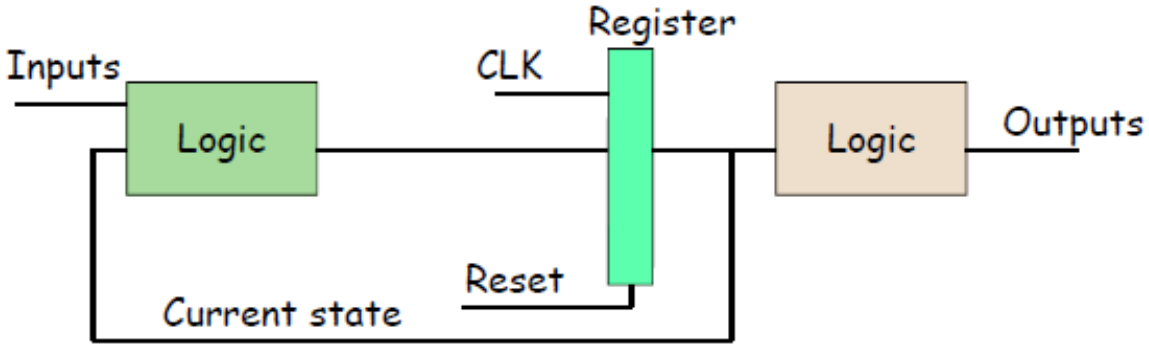


Mealy Machine





Buffering



- In a typical Moore machine, we need combinational output logic to implement the output function.
- Since the Moore output is not a function of input signals,
  - it is shielded from the glitches of the input signals.
- However, the state transition and output logic may still introduce glitches to the output signals.

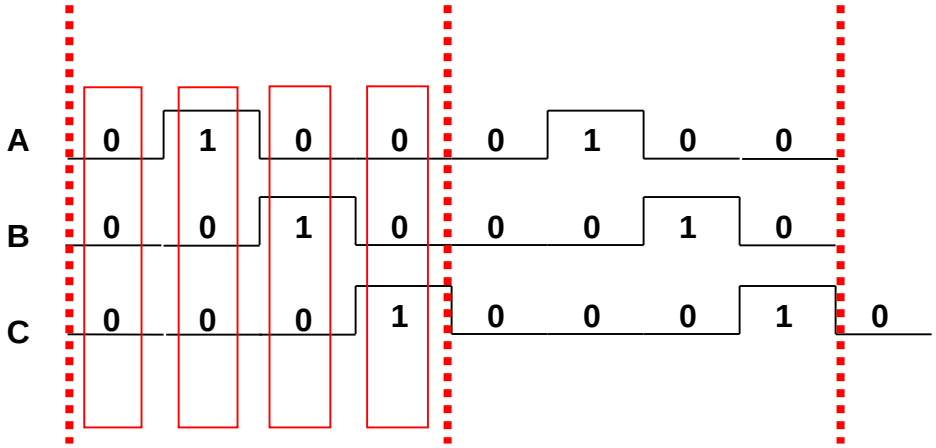
Buffering

- There are two sources of glitches.
  - The first is the possible simultaneous multiple-bit transitions of the state register, as from the "111" state to the "000" state.
    - Even the register bits are controlled by the same clock,
    - the clock-to-Q delay of each D-FF may be slightly different, and thus a glitch may show up in the output signal.
  - The second source is the possible hazards inside the output logic.
- One way to reduce the effect of the output logic is to eliminate it completely.

FSM Variants

- Output = state machine
- Moore machine with clocked outputs
- Mealy machine with clocked outputs

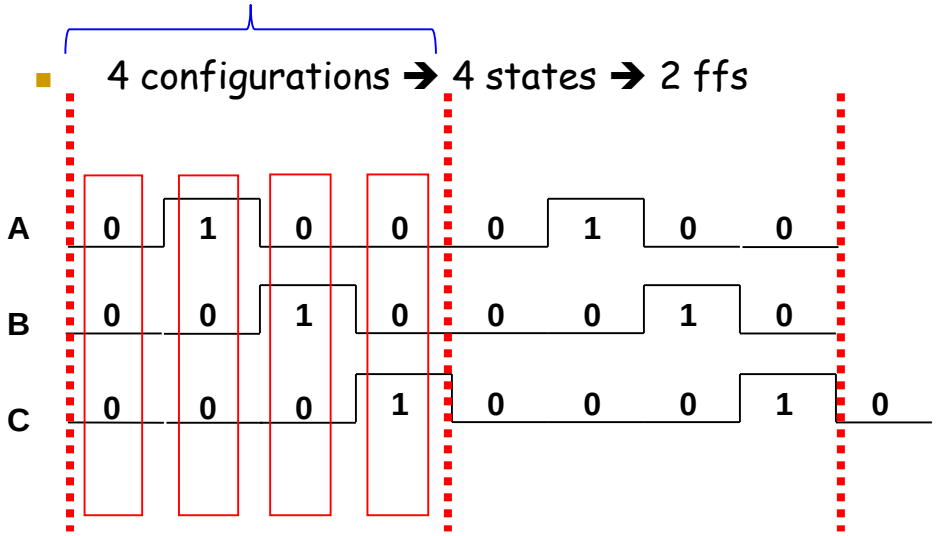
Example: FSM Design

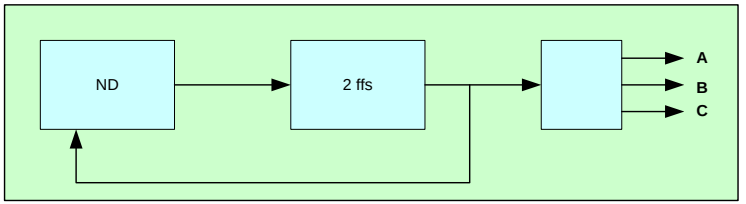


- Design a state machine to produce the waveforms for A, B, C.

Solution 1

- 4 configurations ➔ 4 states ➔ 2 ffs





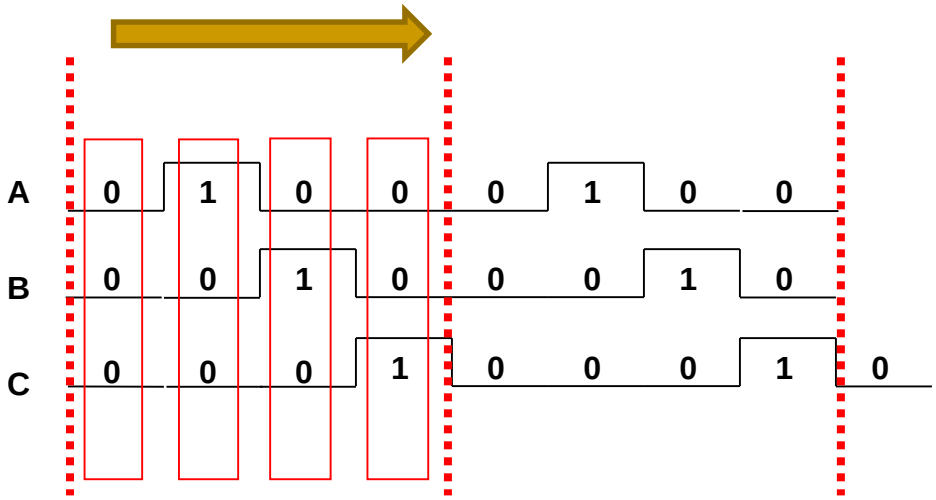
State Vars

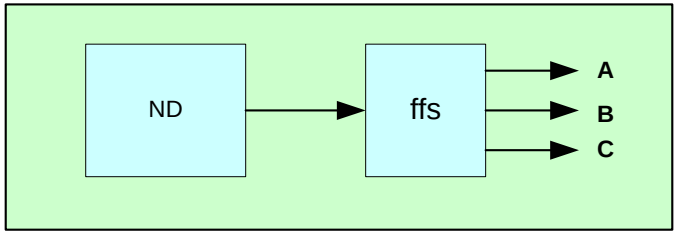
QB	QA	A	B	C
0	0	0	0	0
0	1	1	0	0
1	1	0	1	0
1	0	0	0	1

- Output decoder
  - delay of the output decoder
  - shortcoming !

$A = (QB)'.QA \quad , \quad B = QB.QA \quad , \quad C = QB.(QA)'$

Solution-2





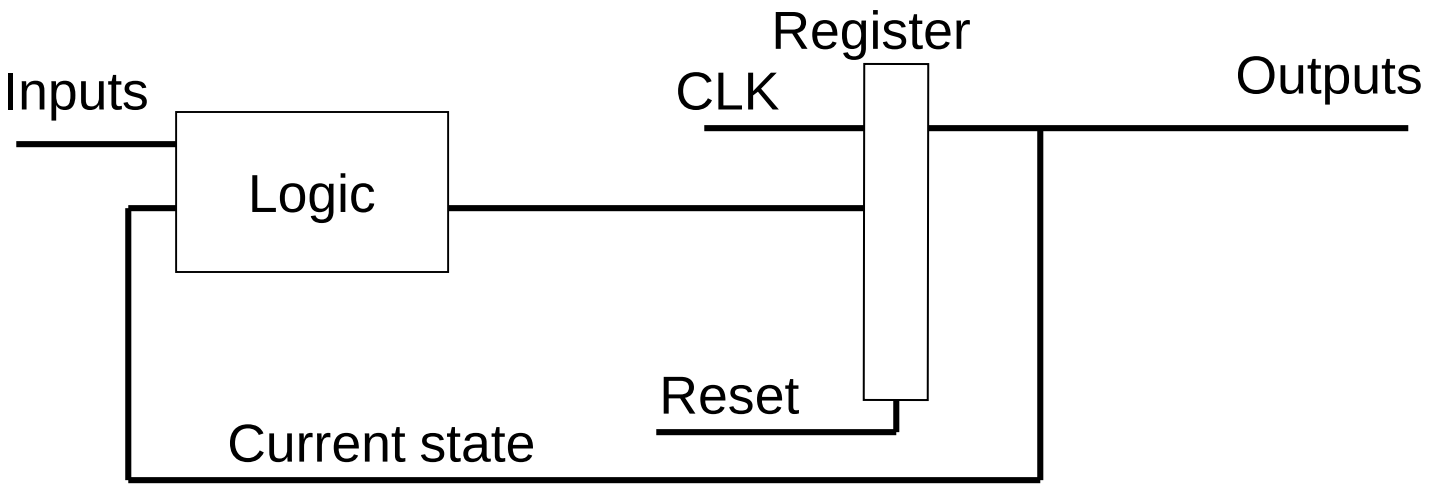
Q_C	Q_B	Q_A
< 0	0	0 >
< 0	0	1 >
< 0	1	0 >
< 1	0	0 >

- Without output decoder
- The outputs directly from ffs.
- A, B, C : State and Output vars

Output = State Machine

- In an output=state machine
  - the output signals are assigned the value of the state vector direct.
- This means that the state coding must be determined in the HDL code, as it affects the function of the output signals.
- An output=state machine is a special instance of a Moore machine
  - in which the combinational process for the outputs has been optimized out.

Output = State Machine



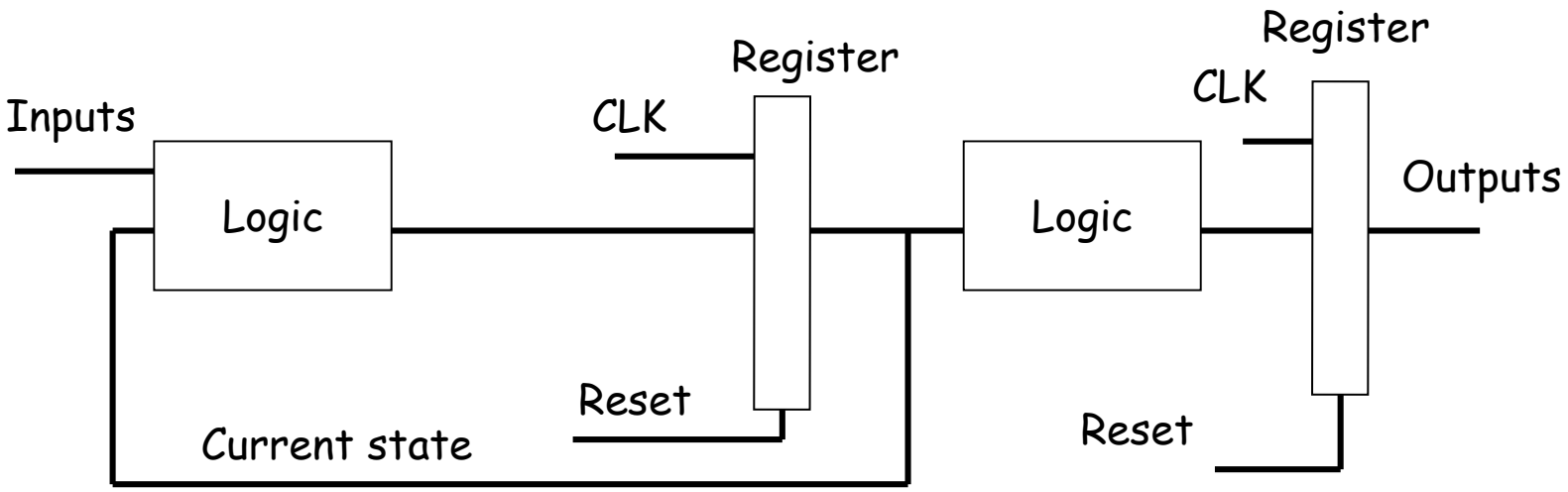
- The output signals come from the state vector direct,
  - the state flip-flops,
  - the outputs are guaranteed spike-free.

Look-ahead Output Circuit For Output

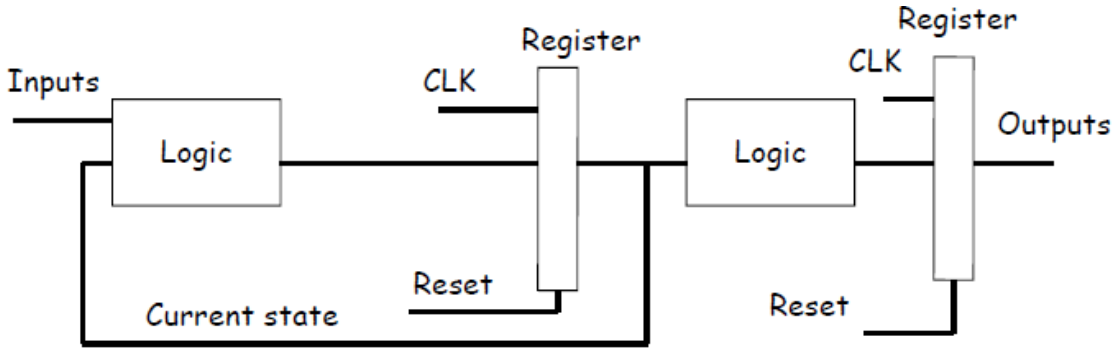
- A more systematic approach to Moore output buffering is to use a look-ahead output circuit.
- The basic idea is to buffer the next output value to cancel the one-clock delay introduced by the output buffer.
- In most systems, we don't know a signal's next or future value.
- However, in an FSM, the next value of the state register is generated by next-state logic and is always available.

- Since the output of an FSM is frequently used for control purposes, we sometimes need a fast, glitch-free signal.
- We can apply the regular output buffering scheme to a Mealy or Moore output signal.
- The buffered signal, of course, is delayed by one clock cycle.
- For a Moore output, it is possible to obtain a buffered signal without the delay penalty.

Moore Machine With Clocked Outputs

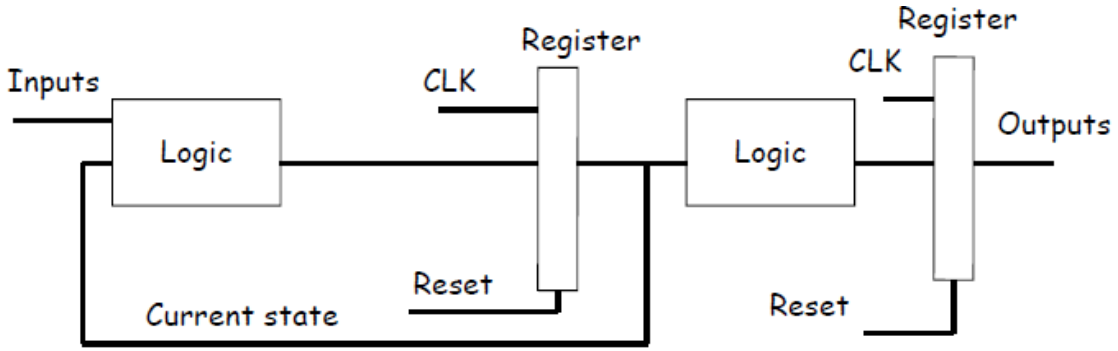


Moore Output Buffering



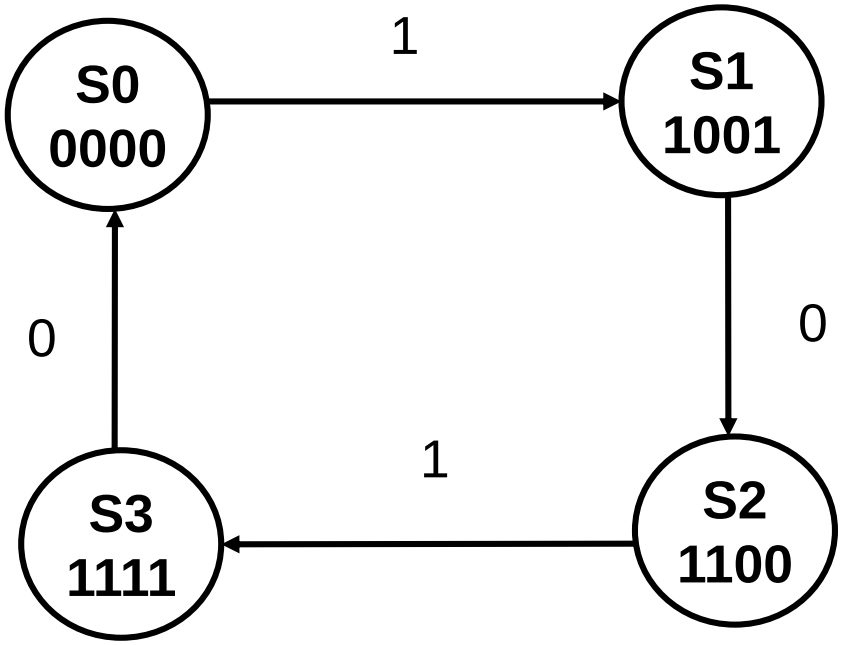
- We can add a buffer by inserting a register or a D-FF to any output signal.
- The purpose of an output buffer is to remove glitches and minimize the clock-to-output delay.
- The disadvantage of this approach is that the output signal is delayed by one clock cycle.

Moore Machine With Clocked Outputs



- All the outputs are synchronized with an extra D-type flip-flop to obtain spike-free outputs.
  - The outputs are a clock cycle behind the ordinary Moore machine's outputs.
- In the VHDL code, this means that the assignment of the output signals has to be done inside the clocked process.
  - This will cause all the output signals to be given the extra D-type flip-flop,
  - as all signals assigned in a clocked process result in a flip-flop

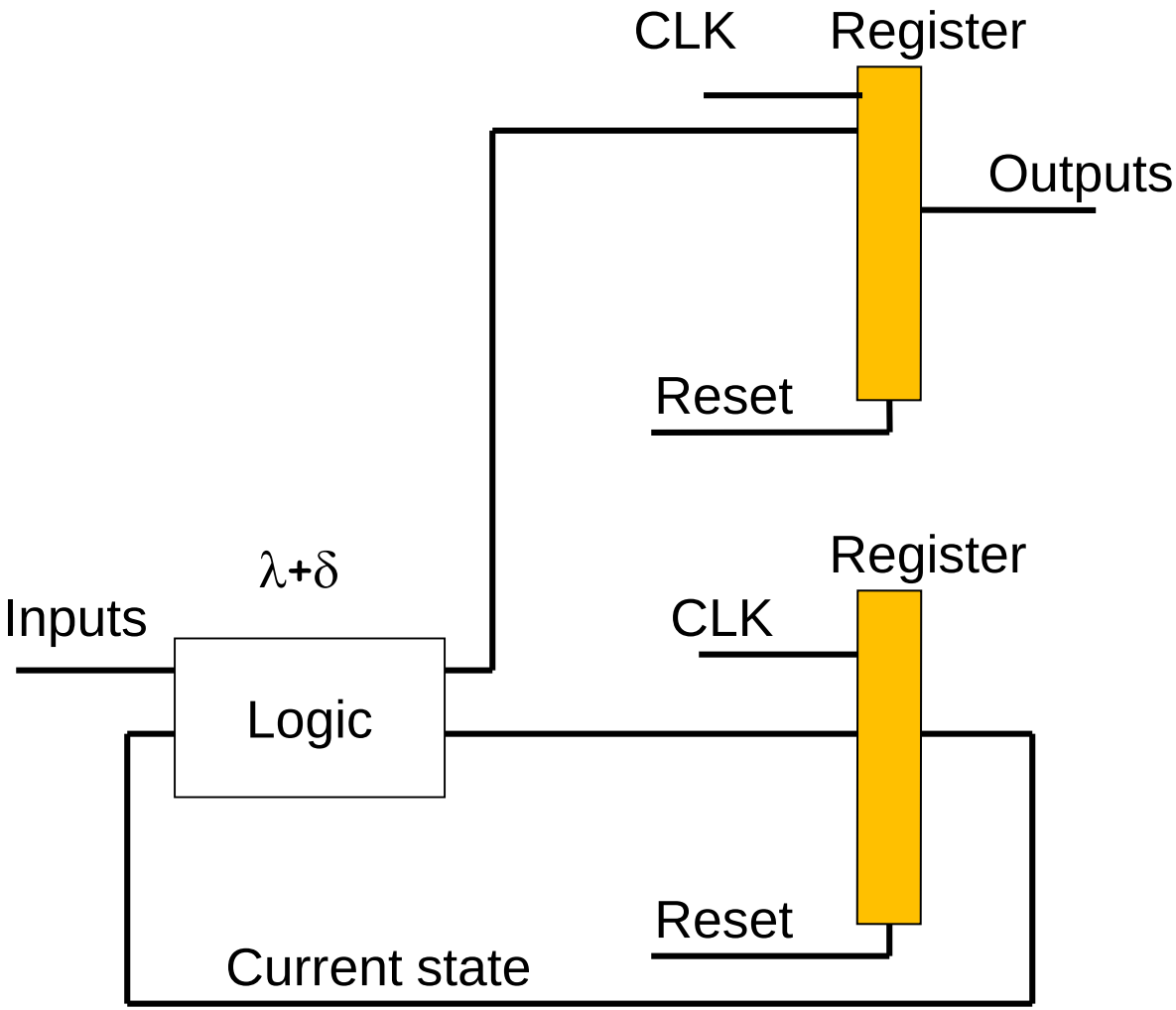
Moore Machine With Clocked Outputs



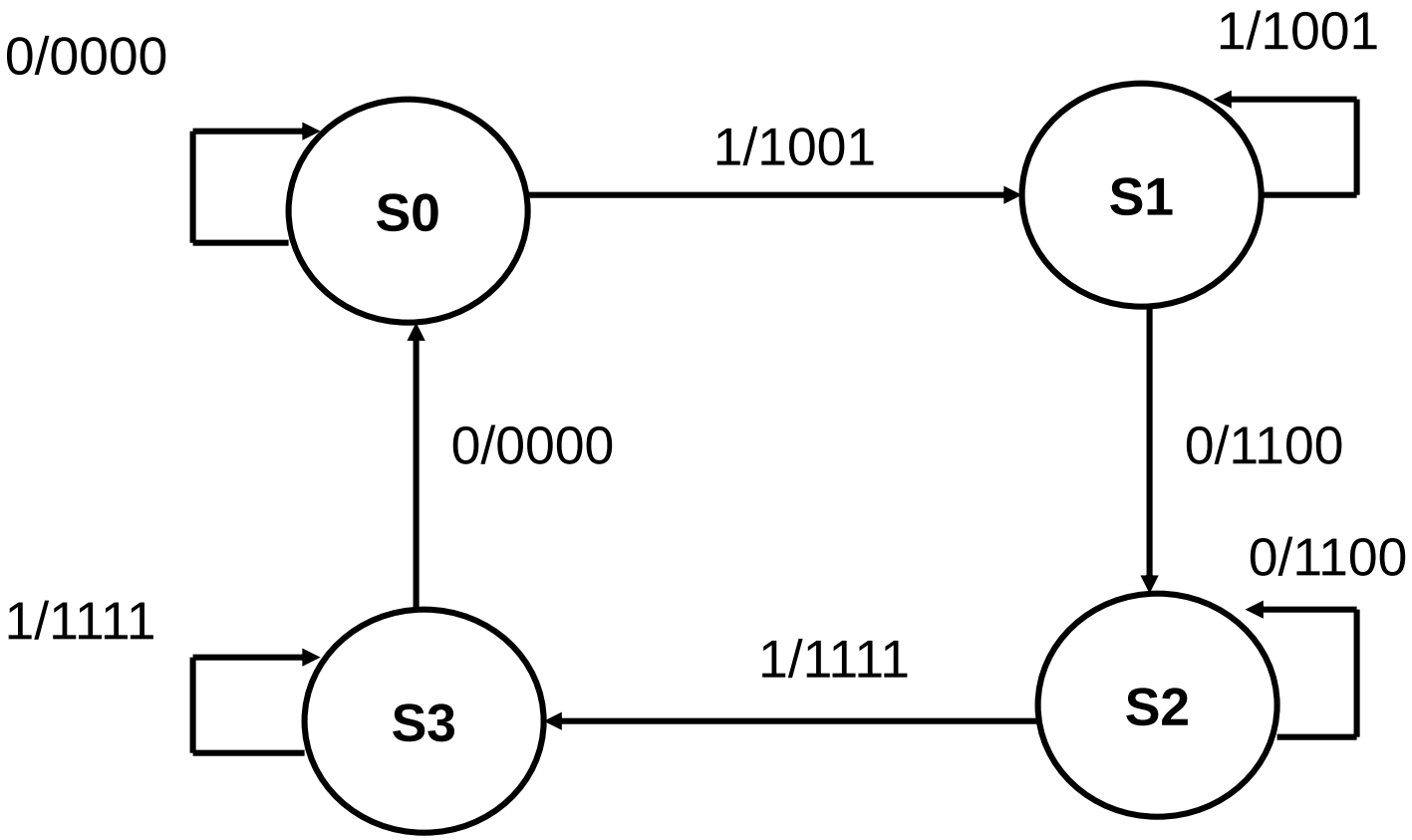
Mealy Machine With Clocked Outputs

- The difference between a Mealy machine with clocked outputs and an ordinary Mealy machine is that
  - all the outputs have been synchronized with an extra D-type flip-flop to obtain spike-free outputs.
- This means that the outputs on a clocked Mealy machine are a clock cycle behind the ordinary Mealy machine's outputs.

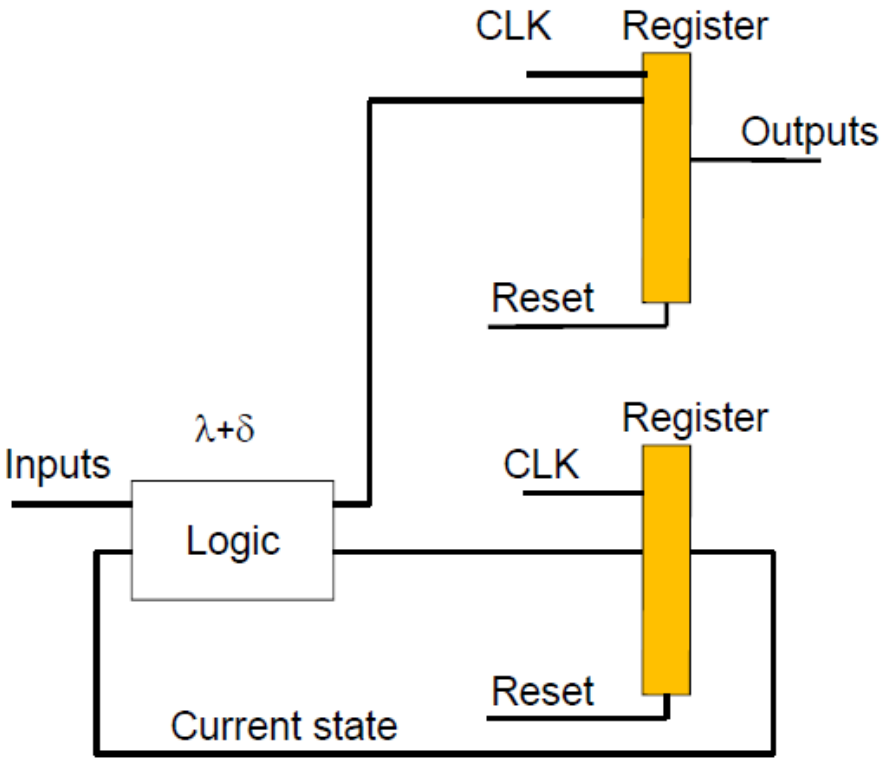
Mealy Machine With Clocked Outputs

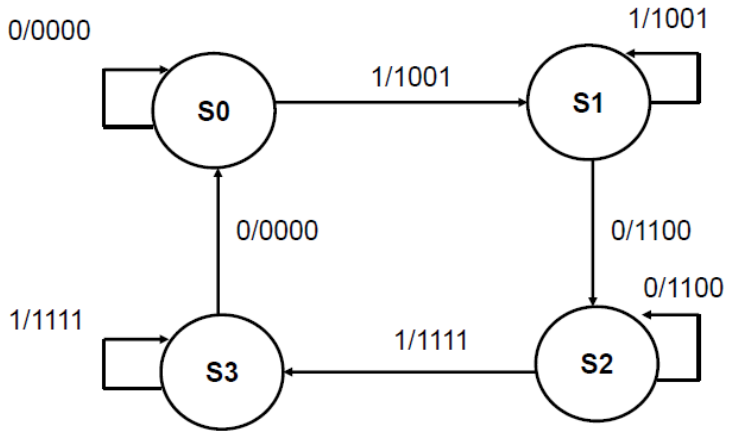


Mealy Machine With Clocked Outputs



Mealy Machine With Clocked Outputs





State Encoding

- Our discussion of FSM so far utilizes only symbolic states.
- During synthesis, each symbolic state has to be mapped to a unique binary representation so that the FSM can be realized by physical hardware.
- *State assignment* is the process of mapping symbolic values to binary representations.
- An FSM with n symbolic states requires a state register of at least $[\log_2 n]$ bits to encode all possible symbolic values.

State Encoding

- A state machine with s states can be implemented using m state variables, where $2^{m-1} < s \leq 2^m$
- There are $(2^m)! / (2^m - s)!$ possible state assignments.
- There is no method for determining which of these assignments will result in minimal combinational next state logic (in polynomial time).
- In addition, other non-minimal state encoding schemes, such as one-hot, exist.

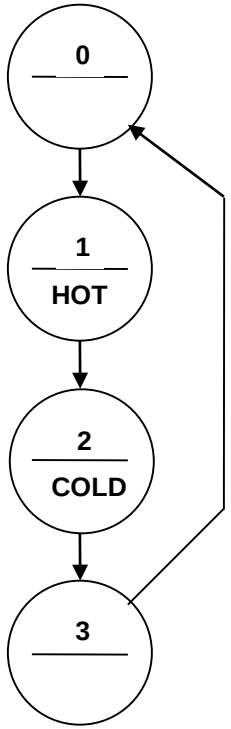
State Encoding

- For a synchronous FSM, the circuit is not delay sensitive and is immune to hazards.
- As long as the clock period is large enough, the synthesized circuit will function properly for any state assignment.
- However, physical implementation of next-state logic and output logic is different for each assignment.
- A good assignment can reduce the circuit size and decrease the propagation delays, which in turn, increases the clock rate of the FSM.

Example 1

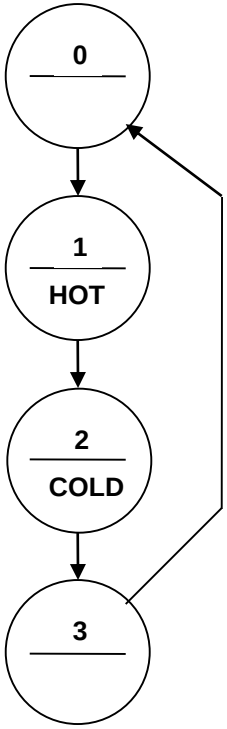
- Design a 2-bit Binary Synchronous up-only counter

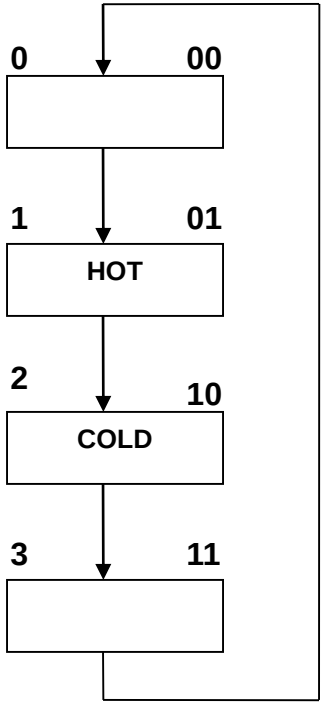
- Step 1: Specification
  - (0, 0) -> (0, 1) -> (1, 0) -> (1, 1) ->(0, 0)
  - 0 -> 1 -> 2 -> 3 -> 0
  - Hot=1 Cold=1 regardless of inputs



Example 1

- Step 2: Determine the encoding for states





Example 1

- Step 3: Determine the devices
  - Number of devices needed
    - N = # of FFs: encode 2^N states
    - N=2FFs to encode 4 states
  - Types of devices
    - D-FFs or JK-FFs
    - Here we use D-FFs
  - Remember the state transition table for selected devices

State transition table for D-ffs

Q → Q+	D
0 → 0	0
1 → 1	1
0 → 1	1
1 → 0	0

Example 1

- Step 4:
  - Determine the state transition table for the global system

Present State		Next State	
QB	QA	QB+	QA+
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Example 1

- Step 5:
  - Determine the present inputs to reach the next states
  - For each individual device:
    - Use its state transition table to determine the possible inputs.

Q → Q+	D
0 → 0	0
1 → 1	1
0 → 1	1
1 → 0	0

Present State		Next State		Flip-Flop Inputs	
QB	QA	QB+	QA+	DB	DA
0	0	0	1	0	1
0	1	1	0	1	0
1	0	1	1	1	1
1	1	0	0	0	0

Example 1

- Step 6: Design the next state decoder

Present State		Next State		Flip-Flop Inputs	
QB	QA	QB+	QA+	DB	DA
0	0	0	1	0	1
0	1	1	0	1	0
1	0	1	1	1	1
1	1	0	0	0	0

Present State		Flip-Flop Inputs	
QB	QA	DB	DA
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Example 1

- Step 6: Design the next state decoder

DB = f (QB, QA)

DA = g (QB, QA)

Present State		Flip-Flop Inputs	
QB	QA	DB	DA
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

DB

QB \ QA	0	1
0	0	1
1	1	0

DB = QB⊕QA

DA

QB \ QA	0	1
0	1	0
1	1	0

DA =QA'

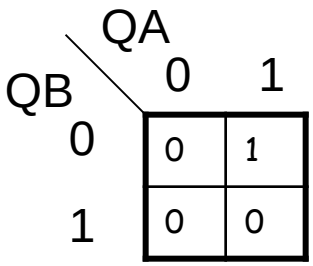
Example 1

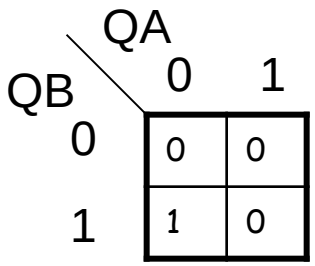
- Step 6: Design the output decoder

QB	QA	HOT	COLD
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

HOT = λ (QA , QB)

COLD = λ (QA , QB)





HOT = (QB)'∧QA

COLD = QB∧(QA)'

Binary (Sequential) Code

- *Binary* (or *sequential) assignment:*
  - assigns states according to a binary sequence.
- This scheme
  - uses a minimal number of bits and
  - needs only a $\lceil \log_2 n \rceil$-bit register.

Example 1

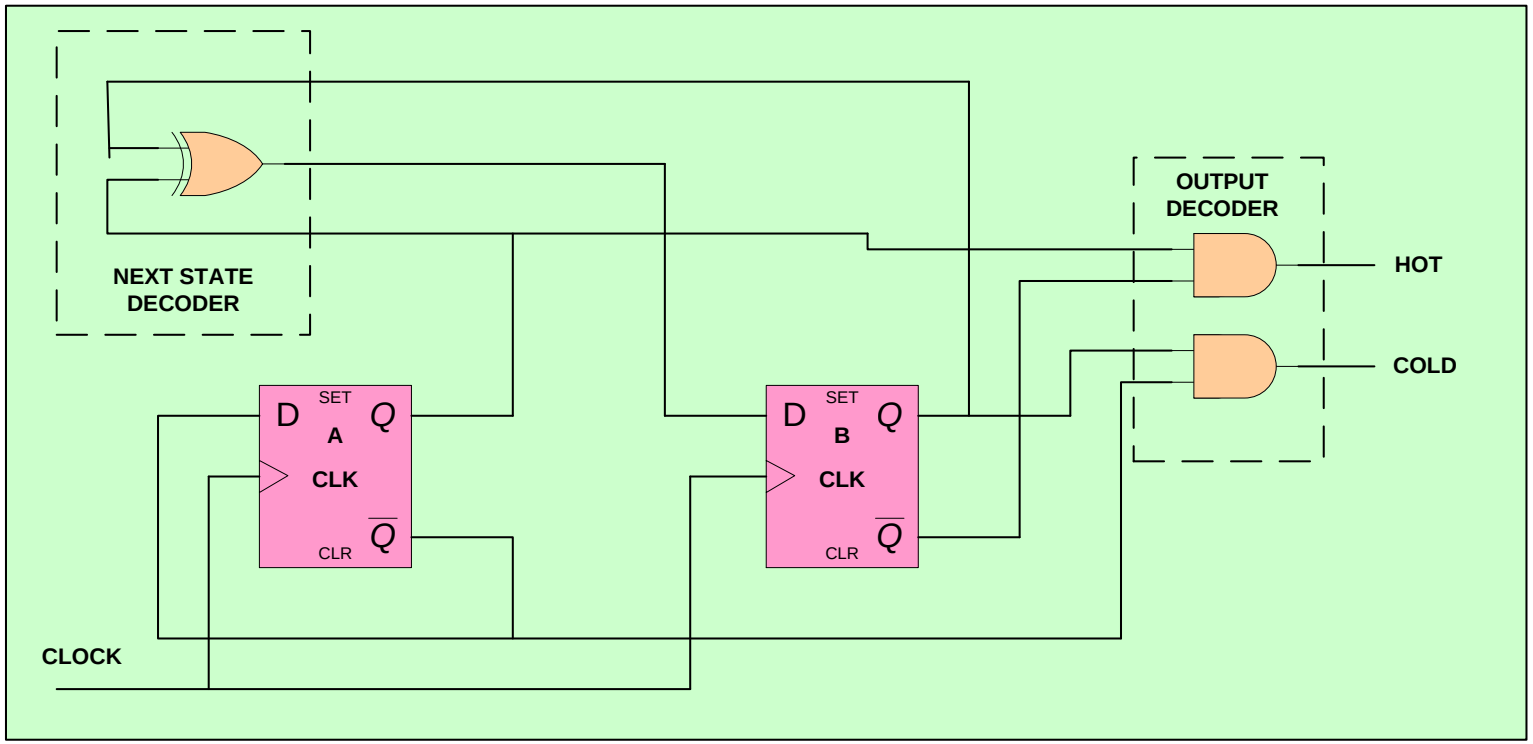
- Step 7: Draw the implementation.

DB = QB⊕QA

DA =QA'

HOT = (QB)'∧QA

COLD = QB∧(QA)'

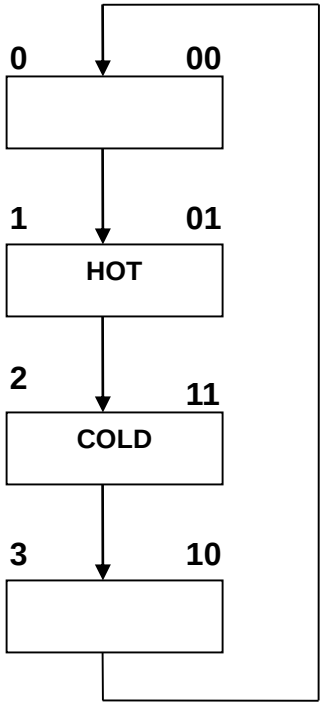


Gray Code after Frank Gray (1947)

- Gray code definition:
  - two successive values differ in only one bit.
- Gray code assignment:
  - assigns states according to a Gray code sequence.
  - uses a minimal number of bits.
- Because only one bit changes between the successive code words in the sequence,
  - we **may** reduce the complexity of next state logic if assigning successive code words to neighboring states.

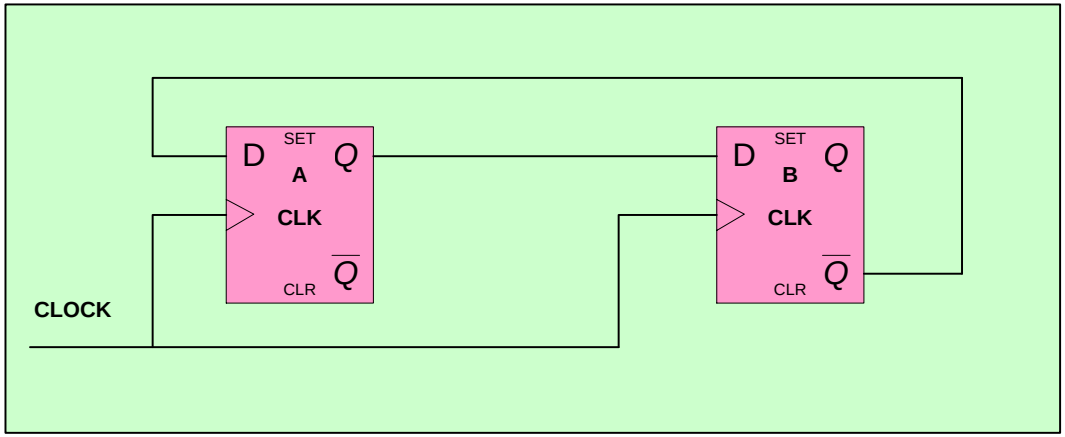
Example 2

- Use Gray code to design it.



DB = QA
DA = QB'

QB	QA	QB'	QA'	DB	DA
0	0	0	1	0	1
0	1	1	1	1	1
1	1	1	0	1	0
1	0	0	0	0	0



State Coding

- For example: type state_type is (s0, s1, s2, s3);
- *Sequential* *Gray* *One-hot*

s0="00";	s0="00"	s0="0001"
s1="01"	s1="01"	s1="0010"
s2="10"	s2="11"	s2="0100"
s3="11"	s3="10"	s3="1000"

One-hot Code

- One-hot refers to how each of the states is encoded in the state vector.
- In a one-hot state machine, the state vector has as many bits as number of states.
- Each bit represents a single state, and only one bit can be set at a time—*one-hot*.
- A one-hot state machine is generally faster than a state machine with encoded states because of the lack of state decoding logic.

One-hot Code

- One-hot assignment:
  - assigns one bit for each state, and thus only a single bit is '1' (or "hot") at a time.
- For an FSM with n states, this scheme needs an n-bit register.
- One-hot has as many flip-flops as it has states.
  - The number of gates becomes larger, but there are a lot of flip-flops in some FPGA architectures (type Xilinx).
    - one-hot decoding can be effective when synthesizing to FPGA.

One-hot Code

- When designing with FPGAs, it may not be important to minimize the number of flip-flops used in the design.
- Instead, we should try to reduce the total number of logic cells used and try to reduce the interconnections between cells.

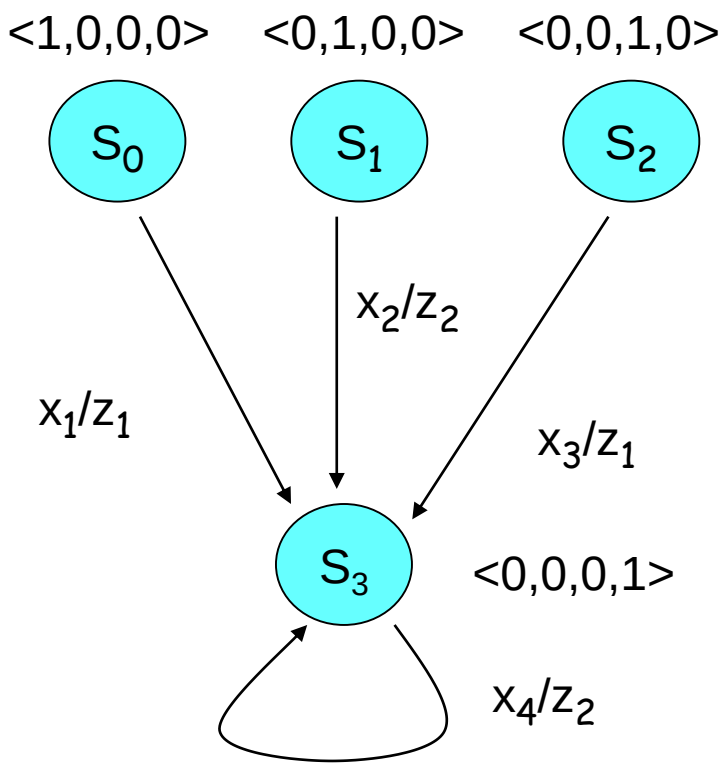
- To design faster logic, we should try to reduce the number of cells required to realize each equation.
- Using a *one-hot state assignment* will often help to accomplish this.

One-hot Code

- One-hot assignment takes more flip-flops than encoded assignment;
  - however, the next state equations for flip-flops are often simpler in the one-hot method than the equations in the encoded method.
- Although one-hot assignment needs more register bits, empirical data from various studies show that
  - these assignments may reduce the circuit size of next-state logic and output logic.

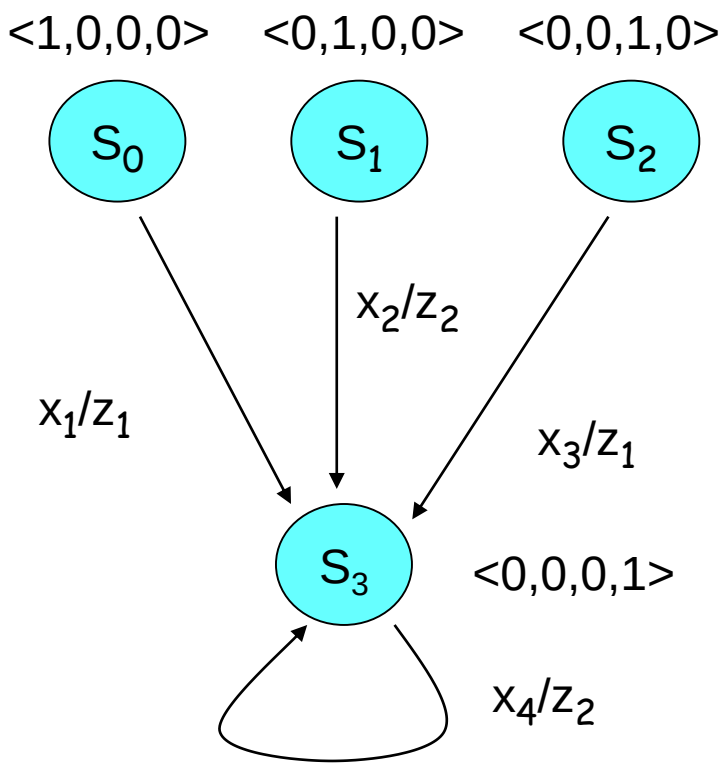
One-hot Code

- Four states: S_0, S_1, S_2, S_3
- Four FFs: Q_0, Q_1, Q_2, Q_3
- State=< Q_0, Q_1, Q_2, Q_3 >
  - S_0= <1,0,0,0>
  - S_1= <0,1,0,0>
  - S_2= <0,0,1,0>
  - S_3= <0,0,0,1>
  - Other 12 states unused.



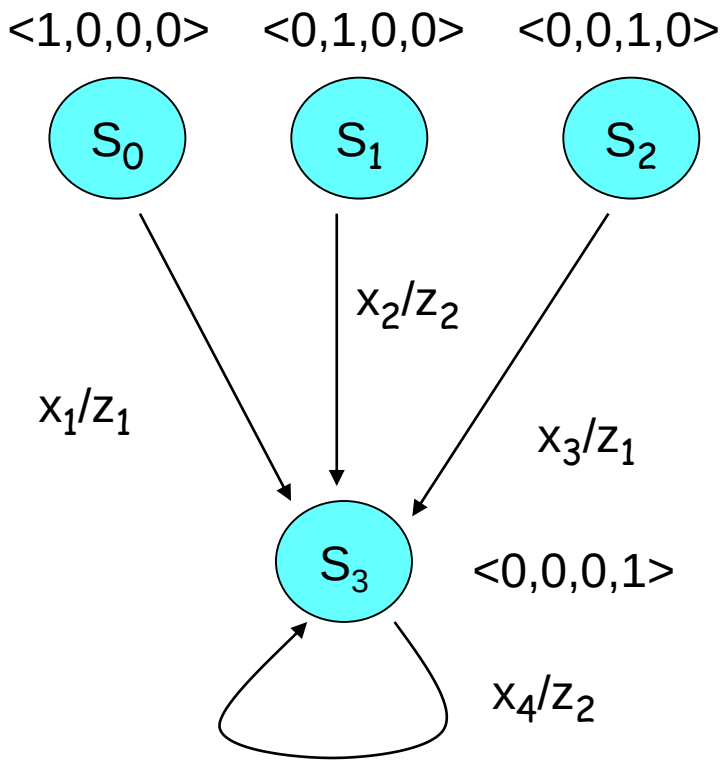
One-hot Code

- $Q_3^+ = x_1(Q_0Q_1'Q_2'Q_3') +$
  $x_2(Q_0'Q_1Q_2'Q_3') +$
  $x_3(Q_0'Q_1'Q_2Q_3') +$
  $x_4(Q_0'Q_1'Q_2'Q_3)$
  $= x_1Q_0 + x_2Q_1 + x_3Q_2 + x_4Q_3$
- Since $Q_0=1$,
  - $Q_1=Q_2=Q_3=0$
  - $Q_1'Q_2'Q_3'$ is redundant.
  - Each term contains exactly one state variable.



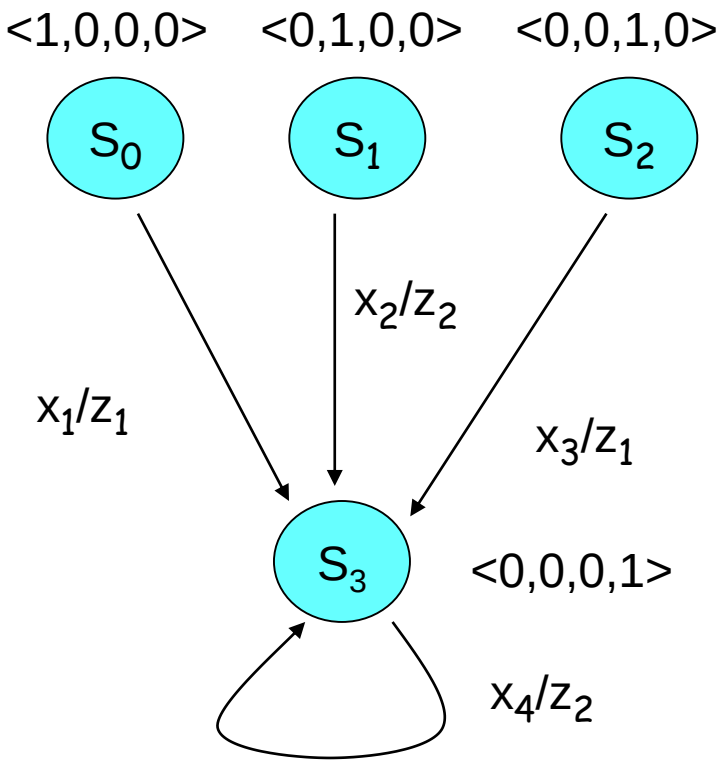
One-hot Code

- In general, when a one-hot state assignment is used, each term in the next-state equation for each flip-flop contains exactly one state variable.



One-hot Code

- Similarly, each term in each output equation contains exactly one state variable.
  - $Z_1 = x_1Q_0 + x_3Q_2$
  - $Z_2 = x_2Q_1 + x_4Q_3$



State Encoding

- The commonest types of state coding are:
  - Sequential
  - Gray
  - One-hot
  - Random
  - ……etc

	Binary	Gray	Johnson	One-Hot
0	0000	0000	00000000	0000000000000001
1	0001	0001	00000001	0000000000000010
2	0010	0011	00000011	0000000000000100
3	0011	0010	00000111	0000000000001000
4	0100	0110	00001111	0000000000010000
5	0101	0111	00011111	0000000000100000
6	0110	0101	00111111	0000000001000000
7	0111	0100	01111111	0000000010000000
8	1000	1100	11111111	0000000100000000
9	1001	1101	11111110	0000001000000000
10	1010	1111	11111100	0000010000000000
11	1011	1110	11111000	0000100000000000
12	1100	1010	11110000	0001000000000000
13	1101	1011	11100000	0010000000000000
14	1110	1001	11000000	0100000000000000
15	1111	1000	10000000	1000000000000000

State Coding

- Obtaining the optimal assignment is very difficult.
- For example, if we choose the one-hot scheme for an FSM with n states, there are n! (which is worse than 2^n) possible assignments.
- It is not practical to obtain the optimal assignment by examining all possible combinations,
- However, there exists special software that utilizes heuristic algorithms that can obtain a good, suboptimal assignment.
- The state coding does not affect the function of the state machine and can be selected in the synthesis tool.
- A state optimizer:
  - determine the state coding when synthesizing the HDL code.

Handling the Unused States

- When we map the symbolic states of an FSM to binary representations,
  - there frequently exist unused binary representations (or states).
- For example, there are 6 states in the memory controller FSM.
- If the binary assignment is used, a 3-bit register is needed.
- Since there are 8 possible combinations from 3 bits, two binary states are not used in the mapping.
- If one-hot state assignment is used, there are 58 (i.e., 2^6 - 6) unused states.

Handling the Unused States

- During the normal operation, the FSM will not reach these states;
  - however, it may accidentally enter an unused state due to noise or an external disturbance.
- One question is what we should do if the FSM reaches an unused state.
- In certain applications, we can simply ignore the situation.
  - It is because we assume that the error will never happen, or, if it happens, the system can never recover.
  - In the latter case, there is nothing we can do with the error.
- On the other hand, some applications can resume from a short period of anomaly and continue to run.
  - In this case we have to design an FSM that can recover from the unused states.
  - It is known as a fault-tolerant or safe FSM.

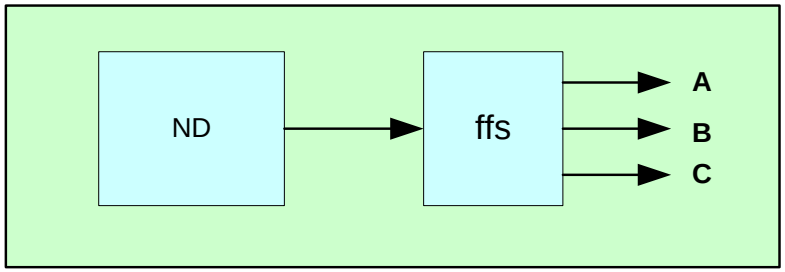
Handling the Unused States

- If not all the states are used, there are three options:
  - Let "chance" decide what is to happen if you land in an undefined valid state.
  - Define what is to happen if you land in an undefined state by ending the case statement with when others.
  - Define all possible states in the HDL code.

- The first option normally requires the least logic,
  - as no decoding is required for the undefined states.

Solution-2

- Without output decoder ➔ The outputs directly from ffs.



❖ A, B, C : Output vars

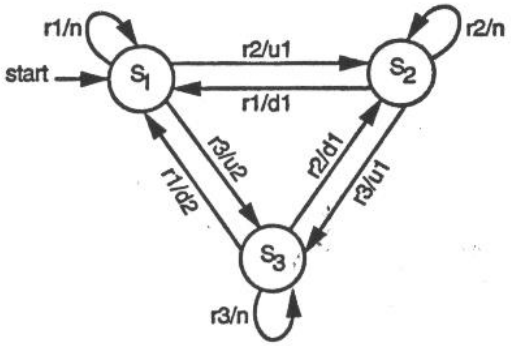
≡ State vars.

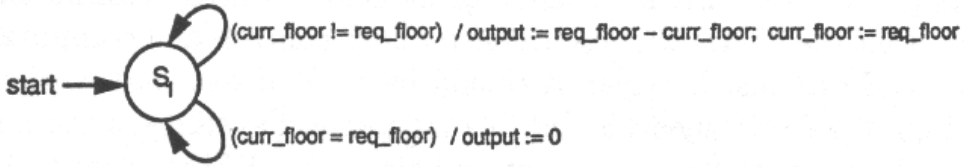
Q_C	Q_B	Q_A
< 0	0	0 >
< 0	0	1 >
< 0	1	0 >
< 1	0	0 >

Hang States		
< 0	1	1 >
< 1	0	1 >
< 1	1	0 >
< 1	1	1 >

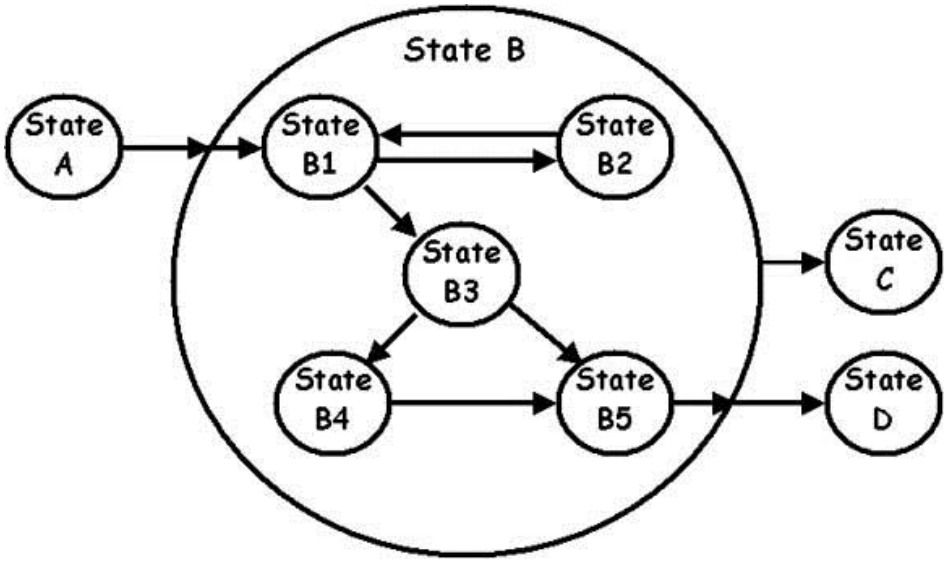
More State-based Systems

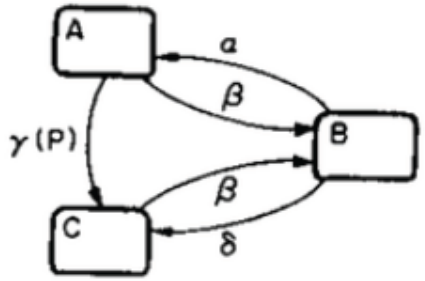
- Extended FSM

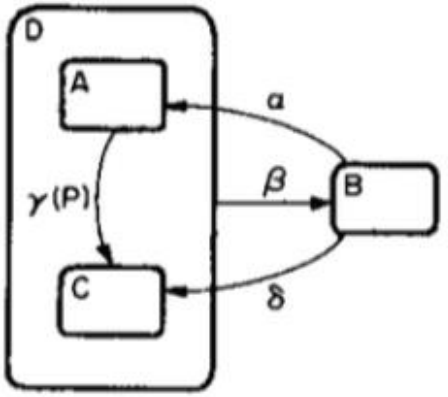




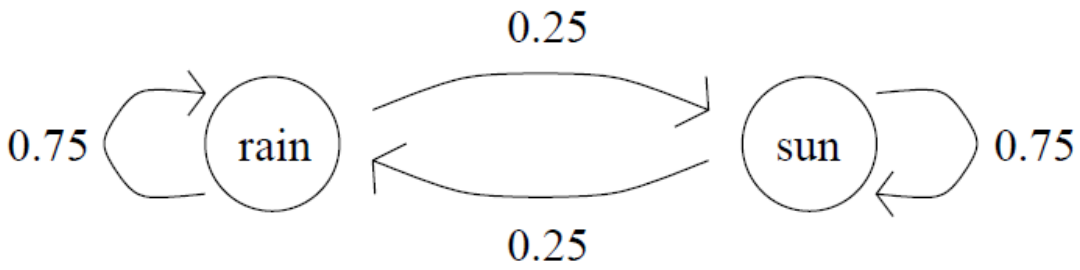
Hierarchical FSM

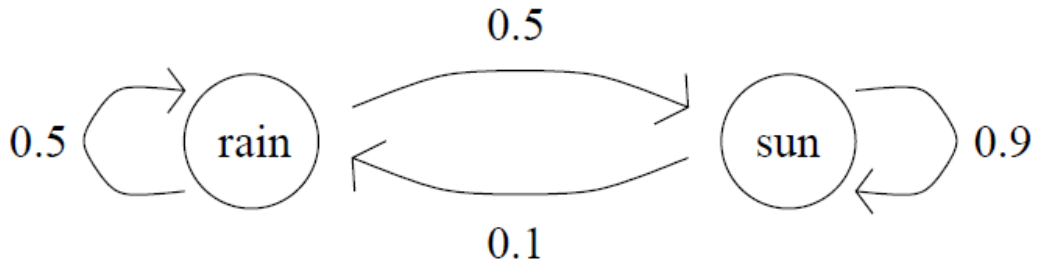




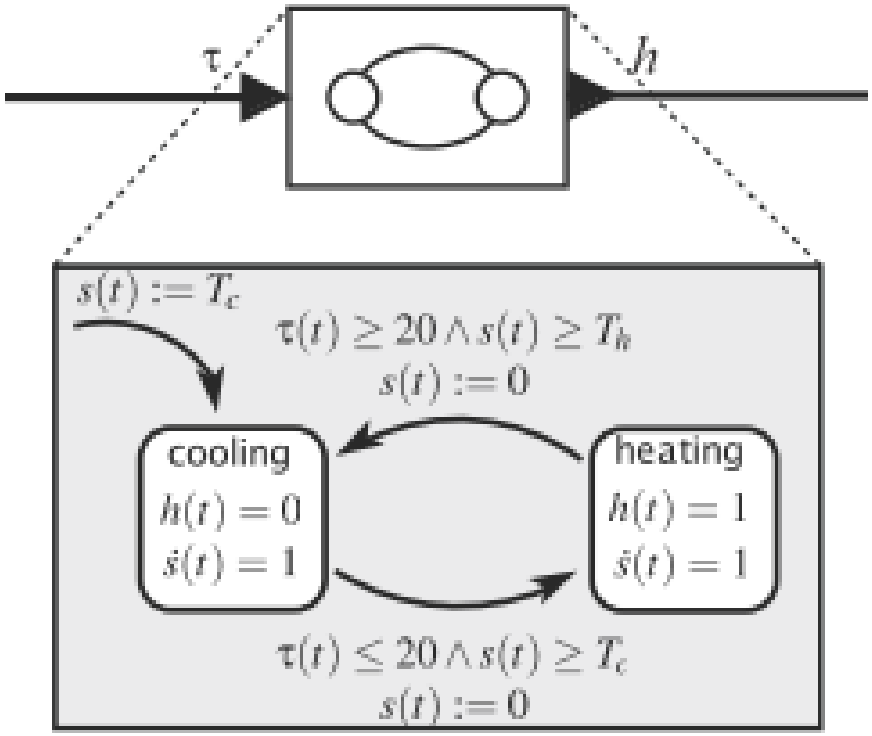


Probabilistic (Stochastic) FSM

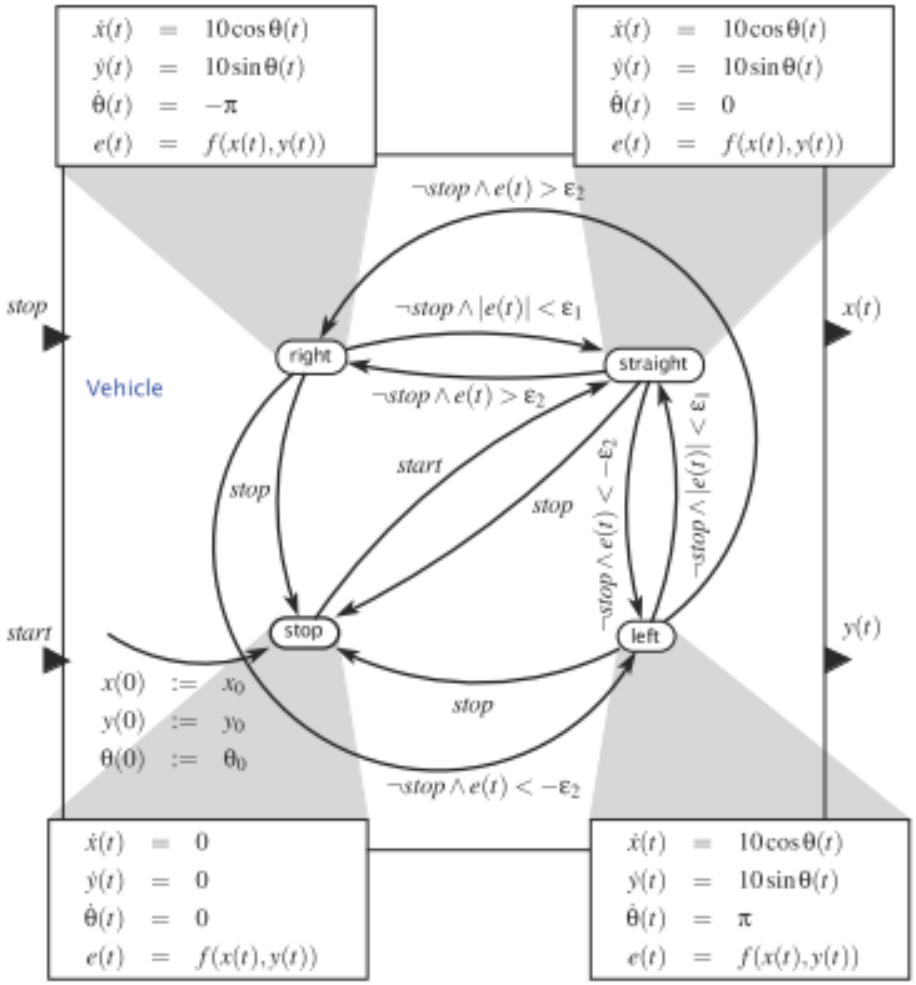




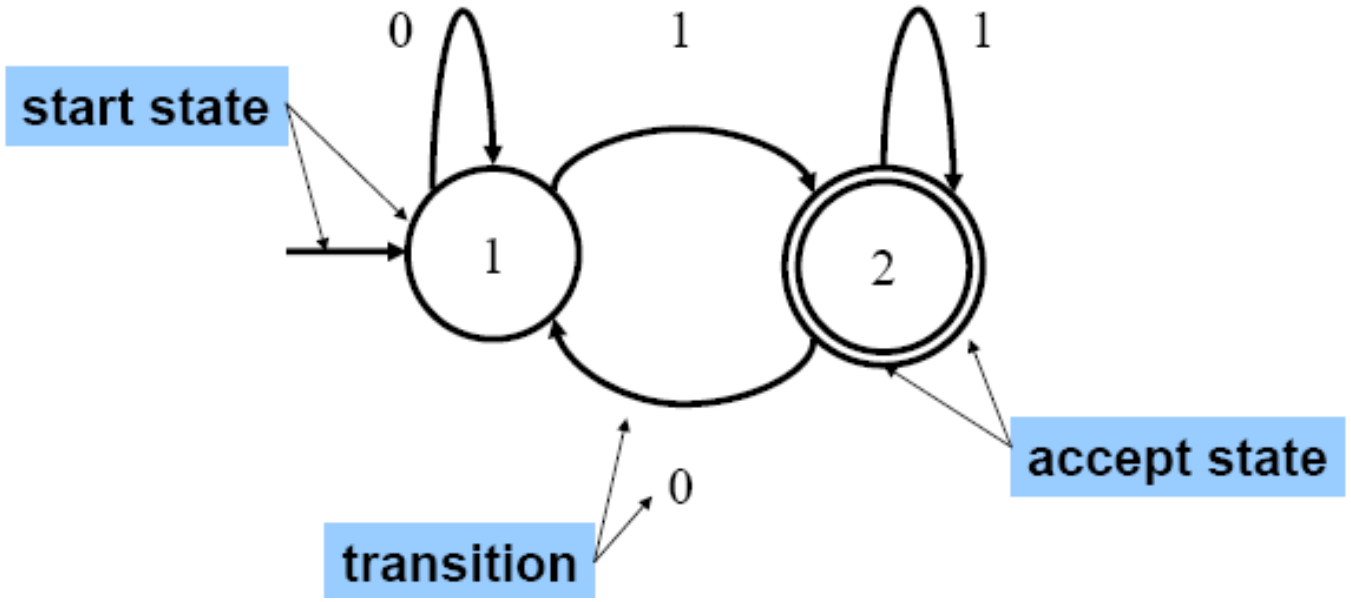
Timed State Machine



Hybrid State Machine



More ? Never Stop .....



References

- Digital System Design with SystemVerilog by Mark Zwolinski. Prentice Hall, 2009.
- Language Reference Manual (LRM) – IEEE 1800-2005