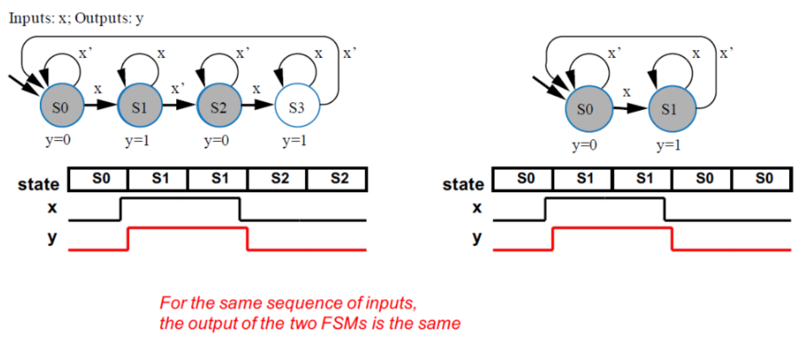
VE270 Lecture 11 FSM Optimizations

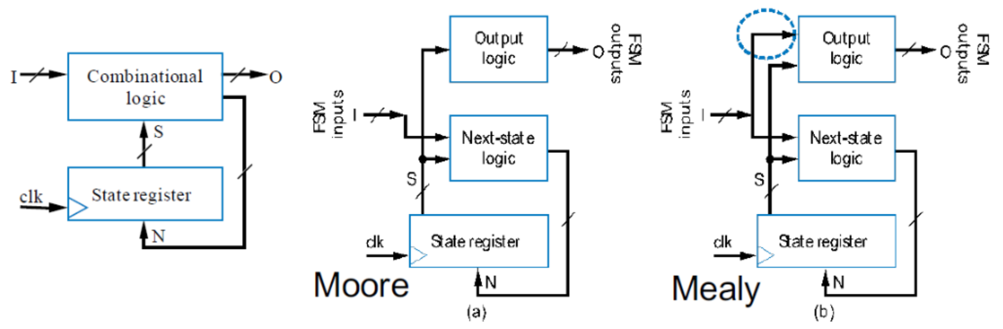
Optimization by State Reduction

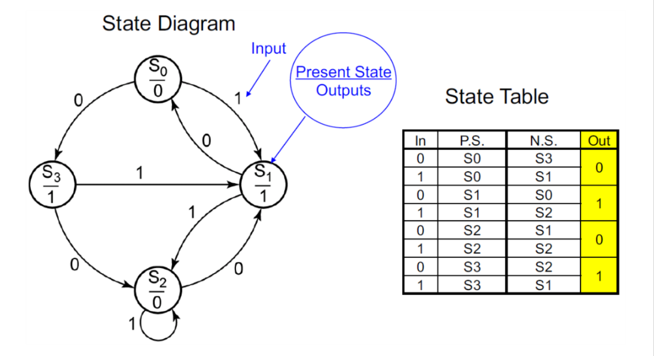
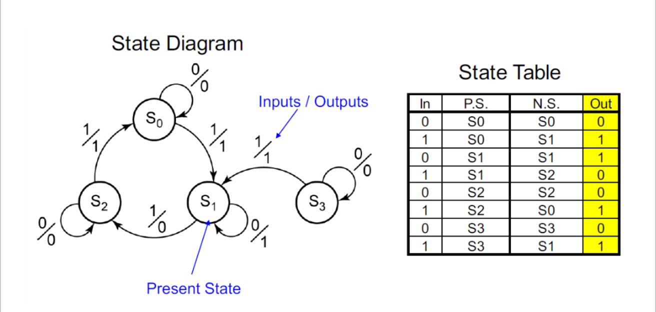


Moore and Mealy FSMs

Check FSM circuit:

- **Next state logic** is defined by the **current state logic** and **FSM inputs**.
- Output logic:
  - **Moore logic FSM**: it depends on **present state** only.
  - **Mealy logic FSM**: it depends on **present state** and **FSM inputs**.



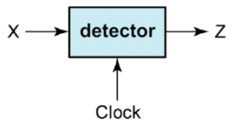
Moore FSM	Mealy FSM
	

In	P.S.	N.S.	Out
0	S0	S3	0
1	S0	S1	
0	S1	S0	1
1	S1	S2	
0	S2	S1	0
1	S2	S2	
0	S3	S2	1
1	S3	S1	

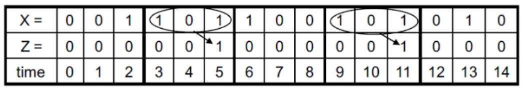
In	P.S.	N.S.	Out
0	S0	S0	0
1	S0	S1	1
0	S1	S1	1
1	S1	S2	0
0	S2	S2	0
1	S2	S0	1
0	S3	S3	0
1	S3	S1	1

Problem Example

- Example: design a non-overlapping sequence detector as Mealy FSM



- Z is determined every three bits, Z = 1, as soon as an input sequence 101 is detected

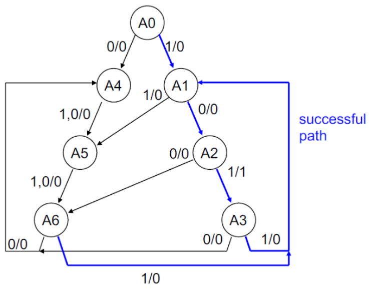


X =	0	0	1	1	0	1	1	0	0	1	0	1	0	1	0
Z =	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

one input: X

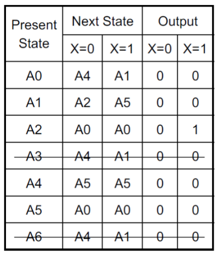
one output: Z

Mealy FSM Design



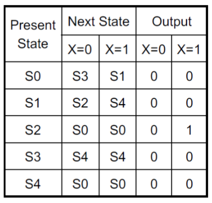
it is the initial diagram, and we can use a state reduction table.

Two states are equivalent iff both next states and outputs are identical



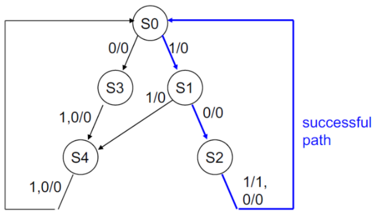
Present State	Next State		Output	
	X=0	X=1	X=0	X=1
A0	A4	A1	0	0
A1	A2	A5	0	0
A2	A0	A0	0	1
~~A3~~	~~A4~~	~~A1~~	~~0~~	~~0~~
A4	A5	A5	0	0
A5	A0	A0	0	0
~~A6~~	~~A4~~	~~A1~~	~~0~~	~~0~~

A0 → S0
A1 → S1
A2 → S2
A4 → S3
A5 → S4



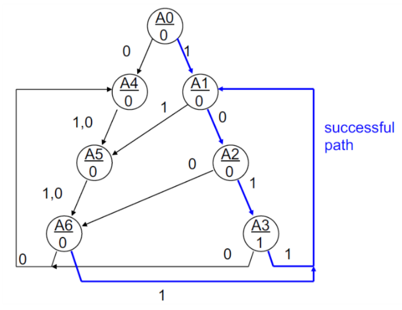
Present State	Next State		Output	
	X=0	X=1	X=0	X=1
S0	S3	S1	0	0
S1	S2	S4	0	0
S2	S0	S0	0	1
S3	S4	S4	0	0
S4	S0	S0	0	0

and we get new one:

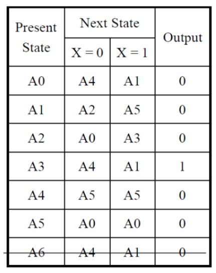


After applying state register, we implement it into a circuit.

Moore FSM Design

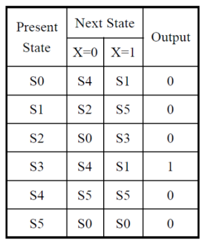


is the initial diagram



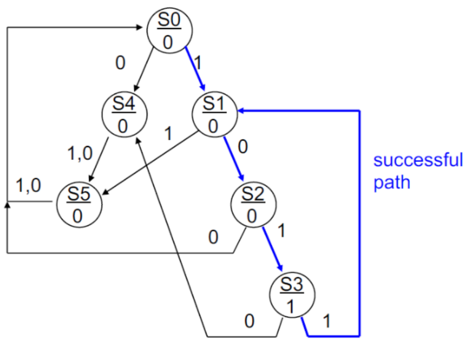
Present State	Next State		Output
	X = 0	X = 1	
A0	A4	A1	0
A1	A2	A5	0
A2	A0	A3	0
A3	A4	A1	1
A4	A5	A5	0
A5	A0	A0	0
~~A6~~	~~A4~~	~~A1~~	~~0~~

A0 → S0
A1 → S1
A2 → S2
A3 → S3
A4 → S4
A5 → S5



Present State	Next State		Output
	X=0	X=1	
S0	S4	S1	0
S1	S2	S5	0
S2	S0	S3	0
S3	S4	S1	1
S4	S5	S5	0
S5	S0	S0	0

reduces the states

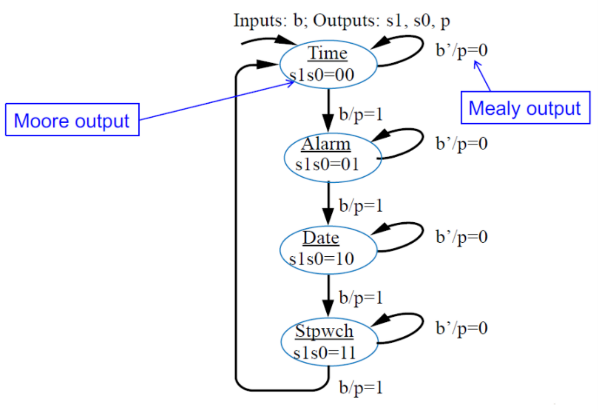


and here is the new diagram

Moore vs Mealy

- Output
  - Moore: Depends on current state
  - Mealy: Depends on current state and inputs
- State Diagram
  - Moore: More states $\longrightarrow$ possibly bigger circuit
  - Mealy: Less states $\longrightarrow$ possibly less number of flip-flops
- Speed of output response to the inputs
  - Moore: as long as one clock cycle delay
  - Mealy: quick, as soon as input changes
- Timing issue
  - Moore: synchronous, more stable
  - Mealy: asynchronous (also based on inputs), may cause serious problem

Combined Mealy and Moore

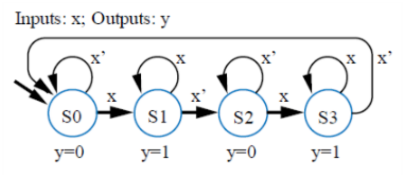


FSM Reverse Engineering

Given a circuit of FSM, we should find the behavior

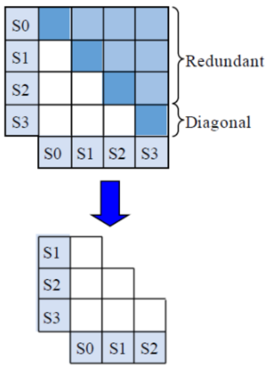
- Mealy or Moore
- State number
- Logic for next state
- State table
- State diagram

Implication Table State Reduction

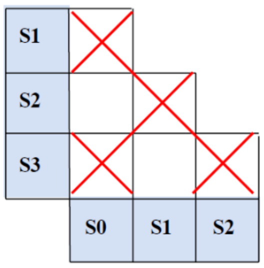


Use implication tables to simplify the FSM.

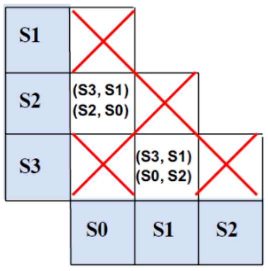
After a change, we get the implication table:



We cross out the state pairs with different outputs since they are non-equivalent:



We list next state pairs for the unmarked state pairs corresponding to every combination of inputs:



We cross out the state pairs if one of its next state pairs is marked.

Then we merge the remaining state pairs.

Optimization by State Encoding

Different encoding may optimize size or tradeoff between size and speed.

State Encoding: One-Hot Encoding

A bit being 1 corresponding to a particular state.

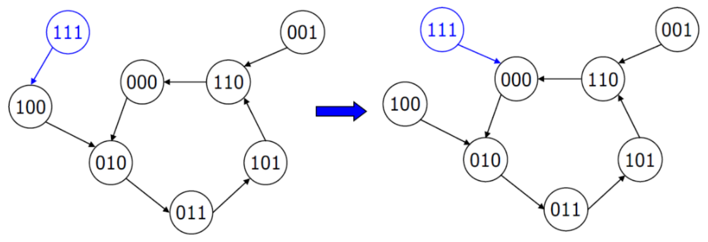
Fewer gates and only one level of logic, less delay than two levels, faster clock frequency.

Self-Starting FSM

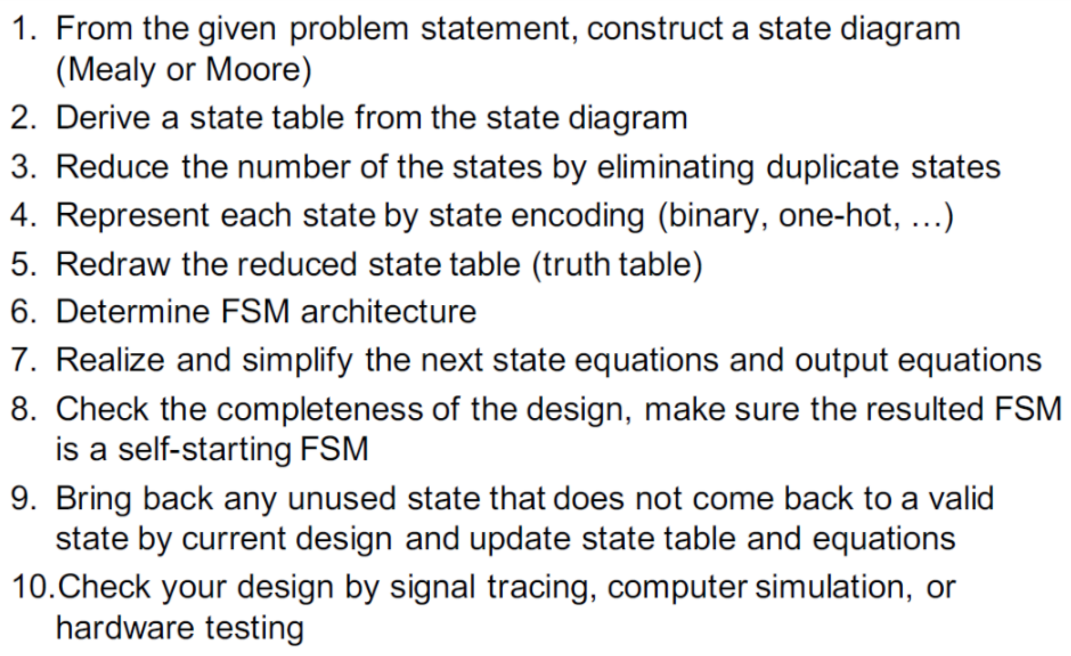
At power-up, FSM may be in an unused or invalid state.

So FSM for invalid states eventually go to valid state, it is self-starting.

We can also make the unused states transfer faster.



FSM Design Procedure



1. From the given problem statement, construct a state diagram (Mealy or Moore)
2. Derive a state table from the state diagram
3. Reduce the number of the states by eliminating duplicate states
4. Represent each state by state encoding (binary, one-hot, …)
5. Redraw the reduced state table (truth table)
6. Determine FSM architecture
7. Realize and simplify the next state equations and output equations
8. Check the completeness of the design, make sure the resulted FSM is a self-starting FSM
9. Bring back any unused state that does not come back to a valid state by current design and update state table and equations
10. Check your design by signal tracing, computer simulation, or hardware testing