Finite State Machines

Prof. James L. Frankel
Harvard University

Version of 10:32 AM 2-Dec-2021

Introduction

- A Finite State Machine, or *FSM*, is an abstraction used to model a device that can be in any one of a fixed number of *states*
- While in each state, the FSM produces one or more *outputs*
- Some event causes a *transition* from state to state
- The selection of a new state can be affected by *inputs*

- In some sense, this is exactly the same functionality that a program or a computer has

State

- We will represent where we are executing with a *state number*
- The state number will be a binary number

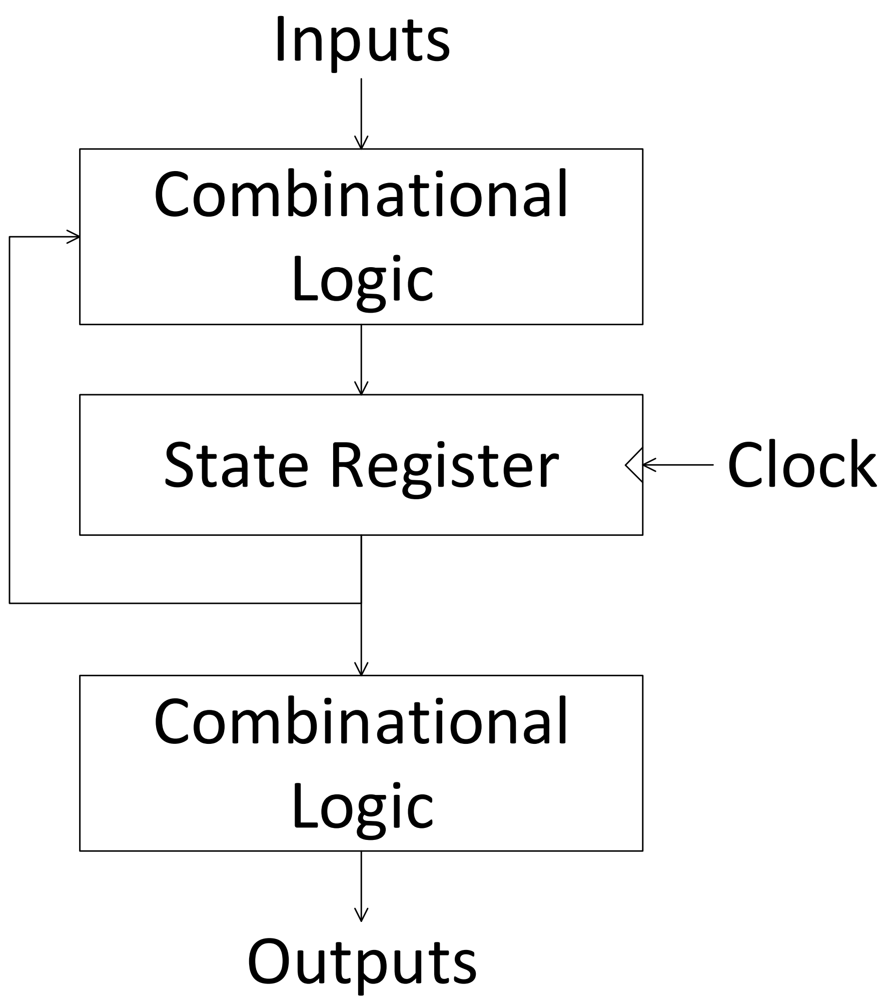
Transition

- Our transitions will be caused by a regularly occurring pulse
  - We refer to this pulse as a *clock*

Moore FSM

- Outputs are determined based on the current state
- Each next state is a function of the current state and the current inputs

Moore FSM Block Diagram



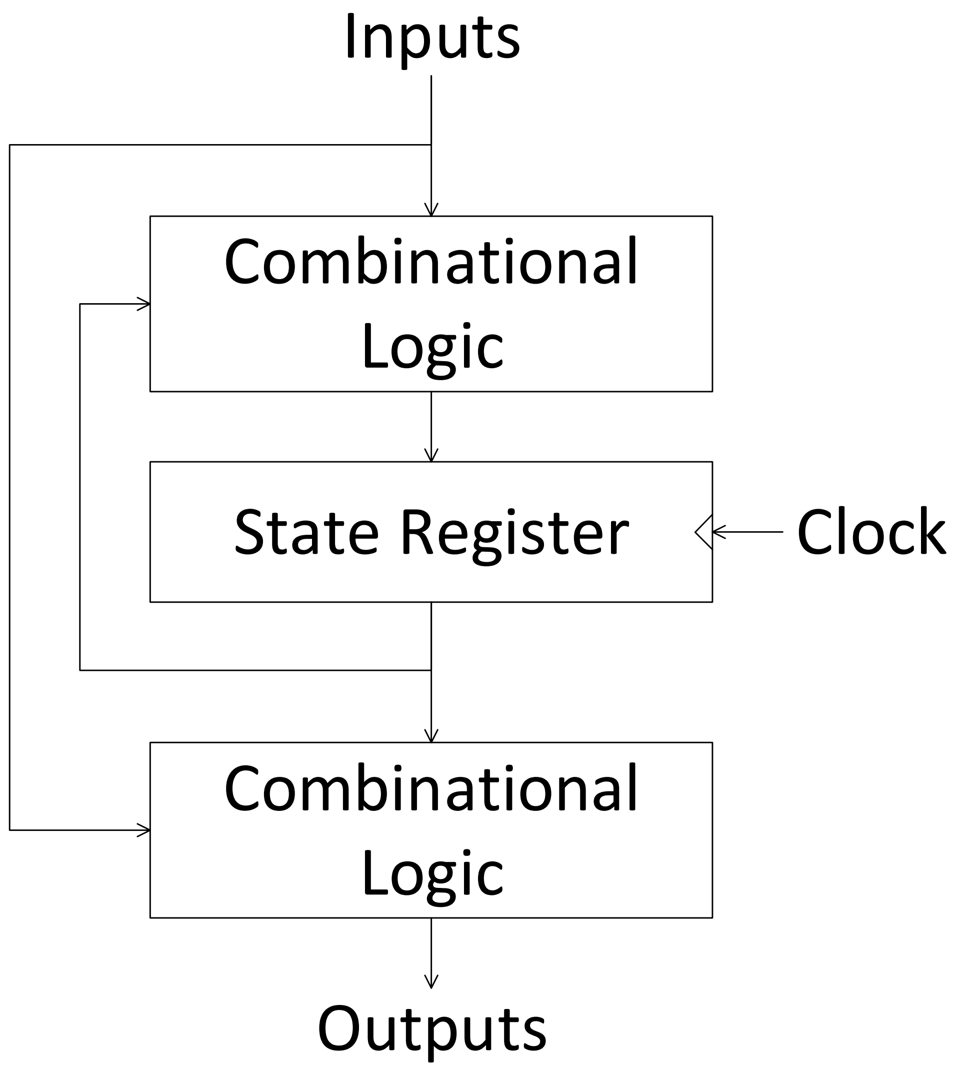
Moore FSM Observations

- Outputs change *synchronously* with the clock edge

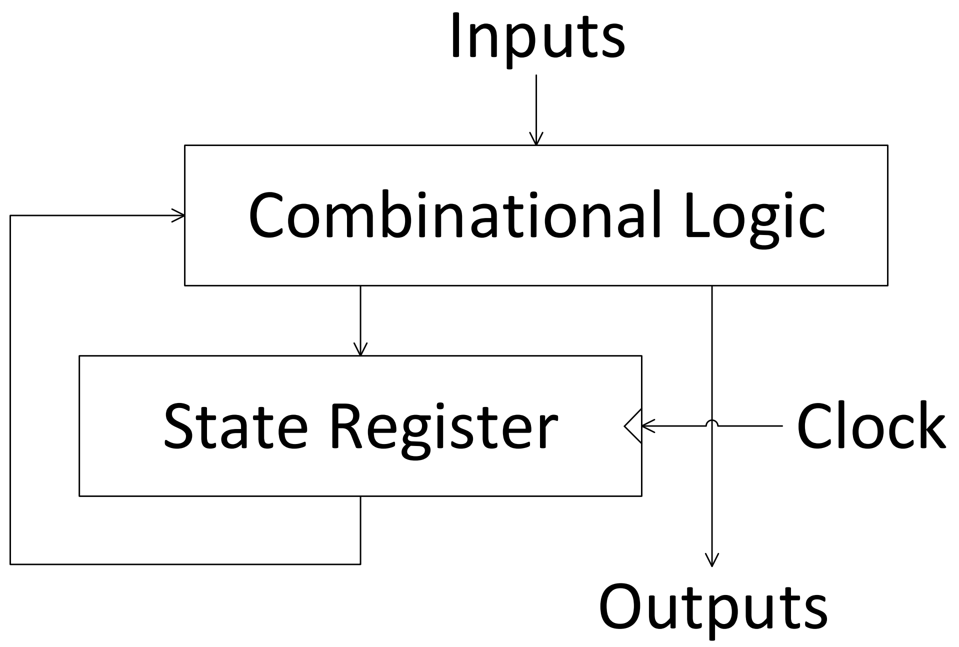
Mealy FSM

- Outputs are determined based on the current state and the current inputs
- Each next state is a function of the current state and the current inputs

Mealy FSM Block Diagram



Simplified Mealy FSM Block Diagram



Mealy FSM Observations

- Outputs change a*synchronously* to the clock edge because they are affected by changes to the inputs (as well as by changes to the state)

Comparing Moore and Mealy FSMs

- Often a Moore FSM has a simplified design at the cost of requiring more states
- The Moore FSM has outputs that change synchronously with the clock
- Often a Mealy FSM requires fewer states
  - That can reduce the number of bits required to store the state number
- It is possible to design either a Moore or a Mealy FSM to exhibit similar functionality

State Transition Diagrams

- Single designated start state
- Our FSMs will run forever
- Our FSMs will be deterministic
  - Only one possible transition from each state with given input conditions
- Edges (or arcs) show all possible transitions
  - Edges are labelled with the input conditions that cause the transition
  - Some inputs can be labelled as don’t-care on the arc
- Moore FSM
  - Outputs are associated with each state
  - Each state label/number is followed by a slash, followed by the outputs
- Mealy FSM
  - Outputs are associated with each edge
  - Each edge is labelled with the input conditions, followed by a slash, followed by the outputs

State Transition Diagrams

- Remember to include self-loops when staying in the same state
- Cover all possible transitions
  - That is, from each state, be sure to cover all input combinations
  - Using don’t-cares can reduce the number of transitions
    - Reducing the number of transitions can simplify the logic required to implement the FSM

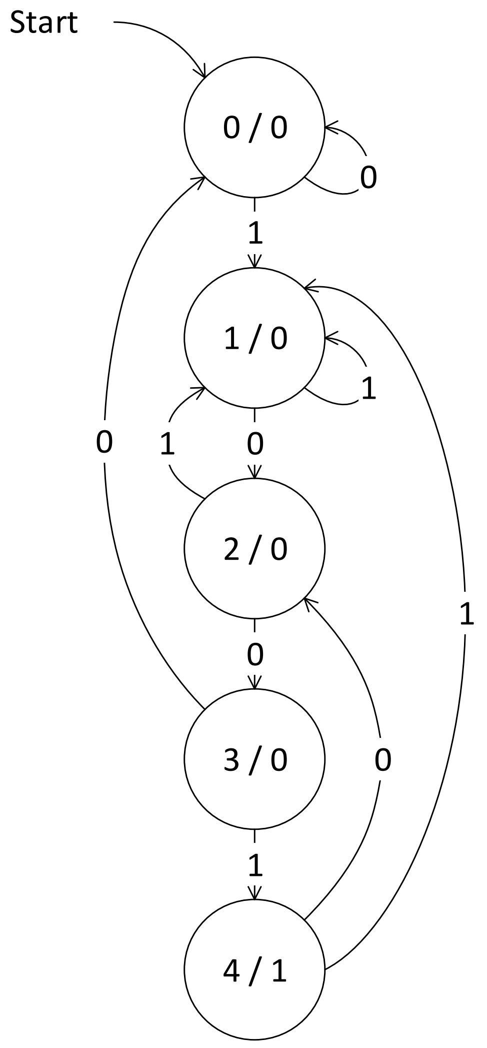
Designing an FSM to Recognize a String of Bits

- Assumptions
  - There is only one binary input
  - The input is stable before and after the rising edge of the clock
    - That is, the input meets the set-up and hold requirements
  - Sequential inputs arrive on sequential clock pulses
    - That is, the inputs are synchronous to the clock

Moore FSM to Recognize 1001

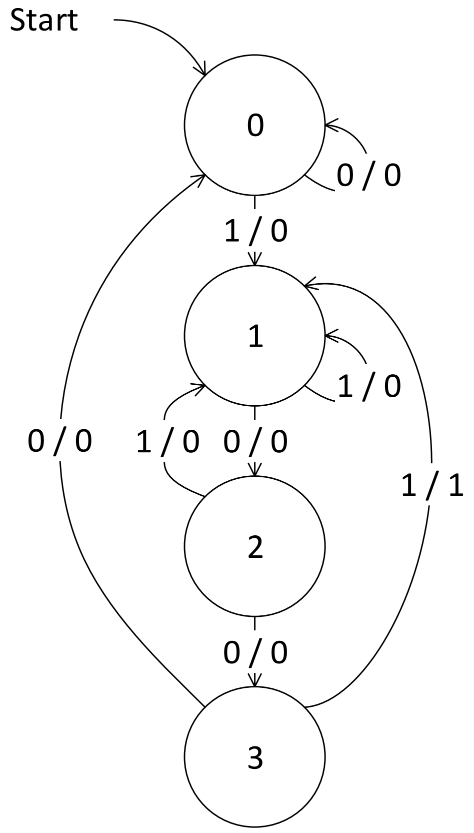
- State transition diagram

- Output high indicates that 1001 has been seen
- Overlapping strings are acceptable



Mealy FSM to Recognize 1001

- State transition diagram



- Output high indicates that 1001 has been seen
- Overlapping strings are acceptable

Moore FSM NextState Truth Table

$State_2$	$State_1$	$State_0$	Input	$NextState_2$	$NextState_1$	$NextState_0$
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	0	1
0	1	0	0	0	1	1
0	1	0	1	0	0	1
0	1	1	0	0	0	0
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	0	1

Moore FSM NextState Truth Table with Decimal States

State	Input	NextState
0	0	0
0	1	1
1	0	2
1	1	1
2	0	3
2	1	1
3	0	0
3	1	4
4	0	2
4	1	1

Moore FSM Output Truth Table

$State_2$	$State_1$	$State_0$	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1

Moore FSM Output Truth Table with Decimal States

State	Output
0	0
1	0
2	0
3	0
4	1

Mealy FSM Truth Table

$State_1$	$State_0$	Input	$NextState_1$	$NextState_0$	Output
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	1	0	0	0	0
1	1	1	0	1	1

Mealy FSM Truth Table with Decimal States

State	Input	NextState	Output
0	0	0	0
0	1	1	0
1	0	2	0
1	1	1	0
2	0	3	0
2	1	1	0
3	0	0	0
3	1	1	1

FSM as a Computer

- An FSM accepts inputs, produces outputs, follows a predetermined “program” (the state transition diagram)
- This is the definition of a computer!
- Only difference: an FSM is not reprogrammable

Using an FSM as our Processor’s Sequencer

- We can use an FSM to transition on clock pulses
  - Each FSM transition corresponds to a cycle in the execution of an instruction
- Each state transition will be based on inputs which can be
  - Opcode bits of the IR register
  - Modifier bits of the instruction in the IR register
  - Various status bits (ALU status (zero, sign, carry$_{out}$, overflow)
- Outputs will control the data path
  - Enable lines to control loading registers
  - Control lines for functional units (memory, register array)
  - Function lines to control the ALU

Using an FSM as a Control Unit

- Similarly to a sequencer in a computer, an FSM can be used as a control unit in any device
- Examples,
  - A dishwasher
  - An elevator
  - A digital radio
  - A disk drive controller
- An FSM can be more responsive (*i.e.,* an FSM can run its program more quickly) than can a programmable microcontroller
  - An FSM may be a better choice when high speed processing is important