Finite State Machine Continued

Combinational and Sequential Circuit

- Digital logic systems can be classified as combinational or sequential.
  - Combinational circuits can be completely described by the truth table.
  - Sequential systems contain state stored in memory elements internal to the system. Their behavior depends both on the set of inputs supplied and on the contents of the internal memory, or state of the system. Thus, a sequential system cannot be described with a truth table. Instead, a sequential system is described as a **finite-state machine (or often just *state machine).***

Clock cycle

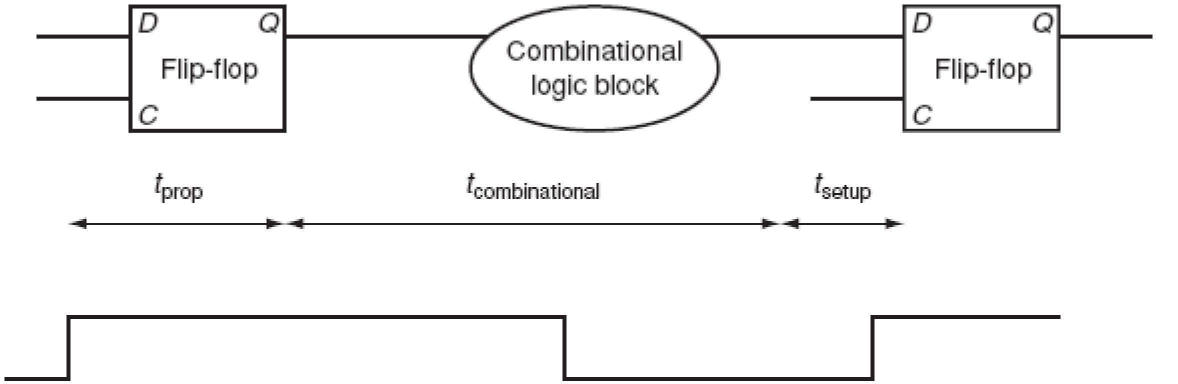


FIGURE B.11.1 In an edge-triggered design, the clock must be long enough to allow signals to be valid for the required setup time before the next clock edge. The time for a flip-flop input to propagate to the flip-flip outputs is t_{prop}; the signal then takes $t_{combinational}$ to travel through the combinational logic and must be valid t_{setup} before the next clock edge.

- t_{prop} is the time for a signal to propagate through a flip-flop; it is also sometimes called clock-to-*Q*.
- $t_{combinational}$ is the longest delay for any combinational logic (which by definition is surrounded by two flip-flops).
- t_{setup} is the time before the rising clock edge that the input to a flip-flop must be valid.

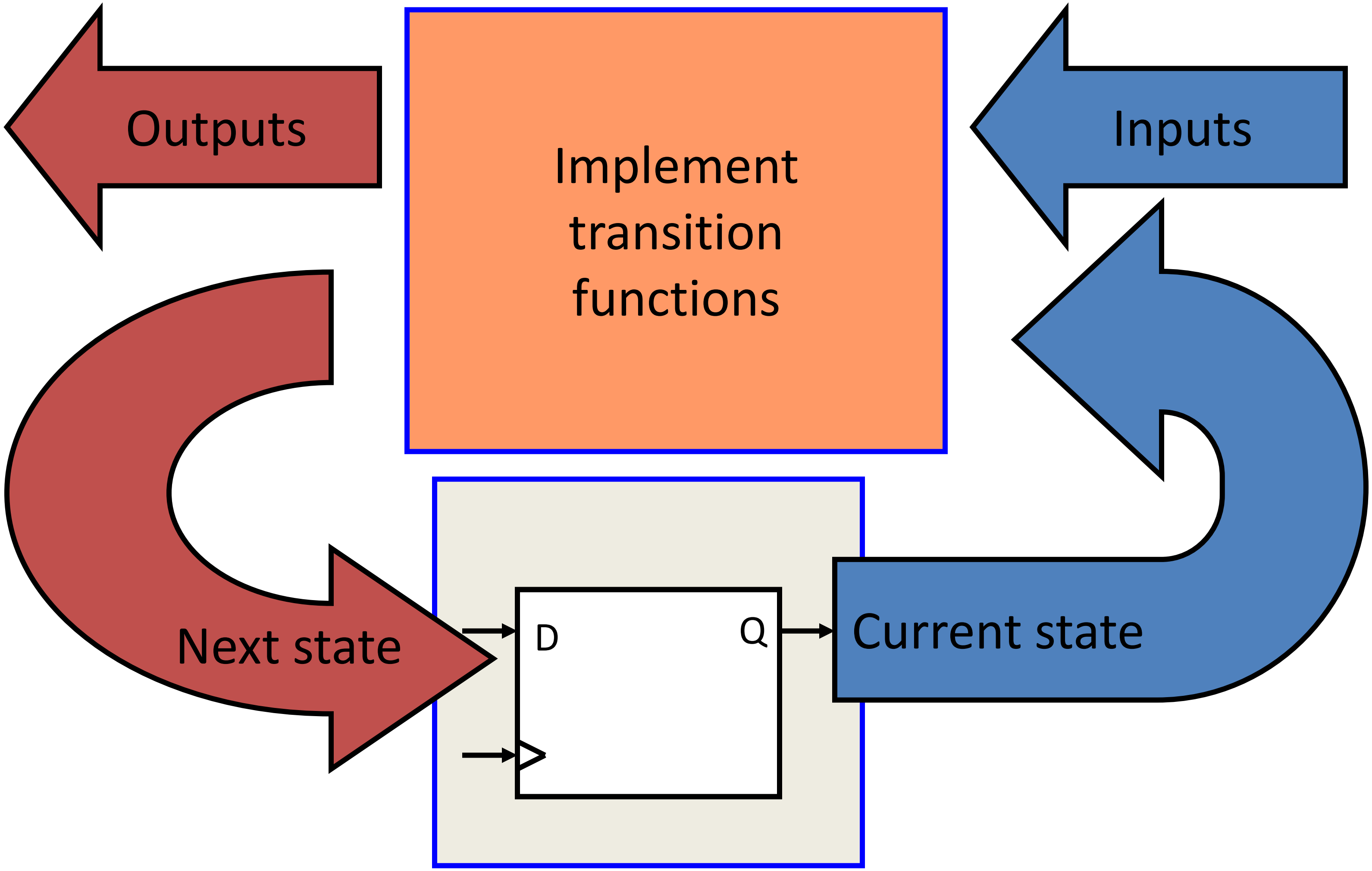
Finite State Machines

- A finite state machine has a set of states and two functions called the next-state function and the output function
  - The set of states correspond to all the possible combinations of the internal storage
    - If there are n bits of storage, there are 2^n possible states
  - The next state function is a combinational logic function that given the inputs and the current state, determines the next state of the system

Finite State Machines

- The output function produces a set of outputs from the current state and the inputs
  - There are two types of finite state machines
  - In a Moore machine, the output only depends on the current state
  - While in a Mealy machine, the output depends both the current state and the current input
  - We are only going to deal with the Moore machine.
  - These two types are equivalent in capabilities

Implementing an FSM



Outputs

Implement transition functions

Inputs

Next state

Current state

Intelligent Traffic Controller

- We want to use a finite state machine to control the traffic lights at an intersection of a north-south route and an east-west route
  - We consider only the green and red lights
  - We want the lights to change no faster than 30 seconds in each direction
    - So we use a 0.033 Hz clock

Intelligent Traffic Controller

- There are two output signals
  - NSlite: When the signal is asserted, the light on the north-south route is green; otherwise, it should be red
  - EWlite: When the signal is asserted, the light on the east-west route is green; otherwise, it should be red

Intelligent Traffic Controller

- There are two inputs
  - NScar: Indicates that there is at least one car that is over the detectors placed in the roadbed in the north-south road
  - EWcar: Indicates that there is at least one car that is over the detectors placed in the roadbed in the east-west road

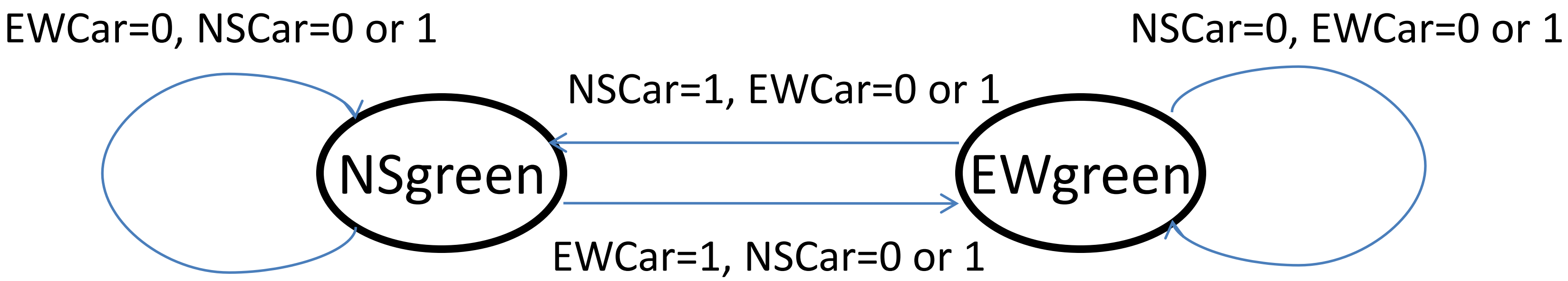
Intelligent Traffic Controller

- The traffic lights should only change from one direction to the other only if there is a car waiting in the other direction
  - Otherwise, the light should continue to show green in the same direction

Intelligent Traffic Controller

- Here we need two states
  - *NSgreen:* The traffic light is green in the north-south direction
  - *EWgreen:* The traffic light is green in the east-west direction

Graphical Representation



EWCar=0, NSCar=0 or 1

NSCar=0, EWCar=0 or 1

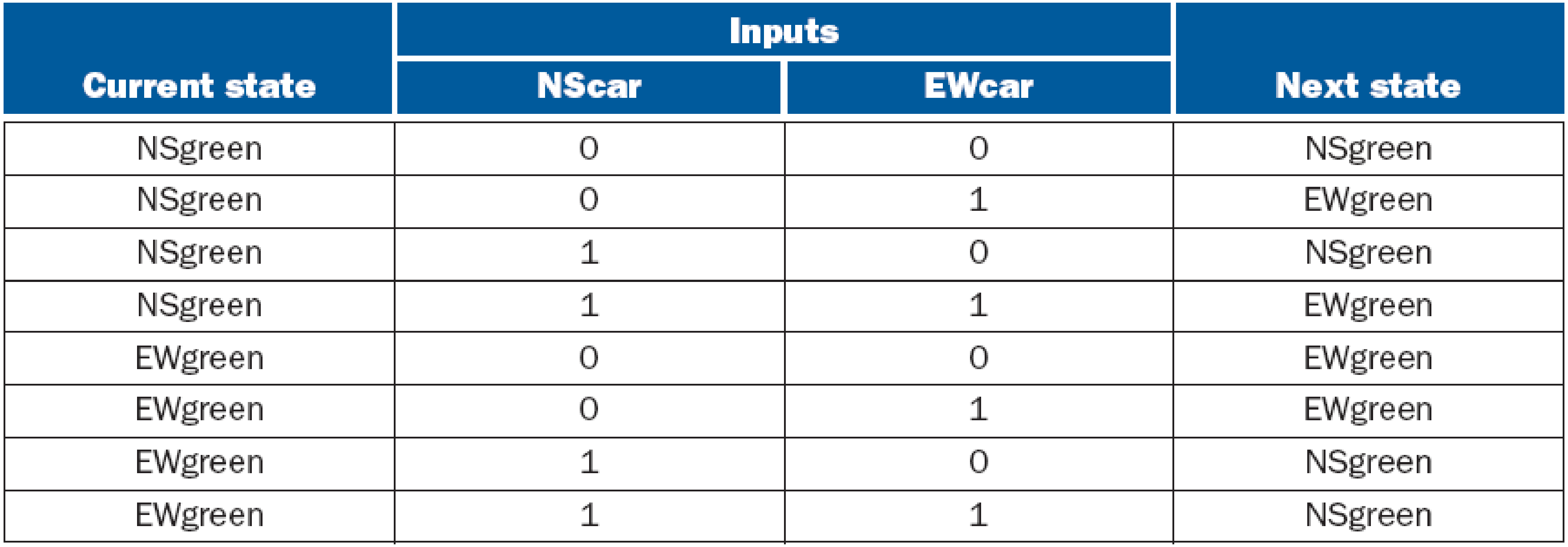
NSCar=1, EWCar=0 or 1

NSgreen

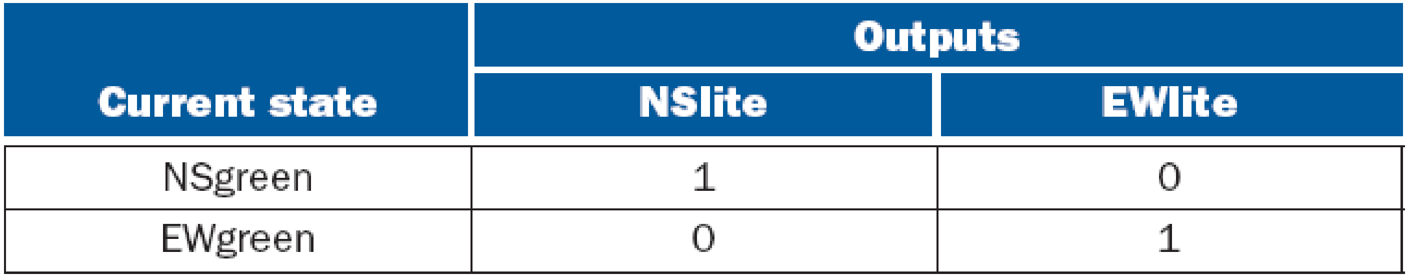
EWgreen

EWCar=1, NSCar=0 or 1

Next State Function and Output Function



Current state	Inputs		Next state
	NScar	EWcar	
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

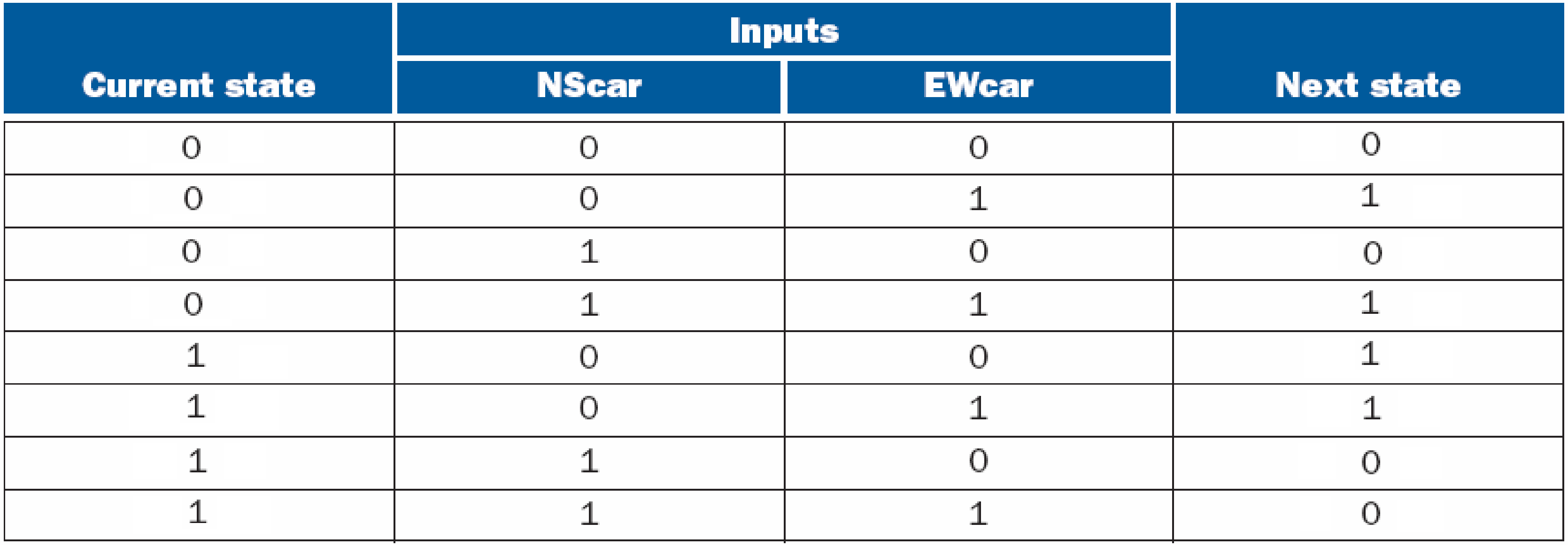


Current state	Outputs	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1

State Assignment

- We need to assign state numbers to the states
  - In this case, we can assign NSgreen to state 0 and EWgreen to state 1
  - Therefore we only need 1 bit in the state register

Combinational Logic for Next State Function



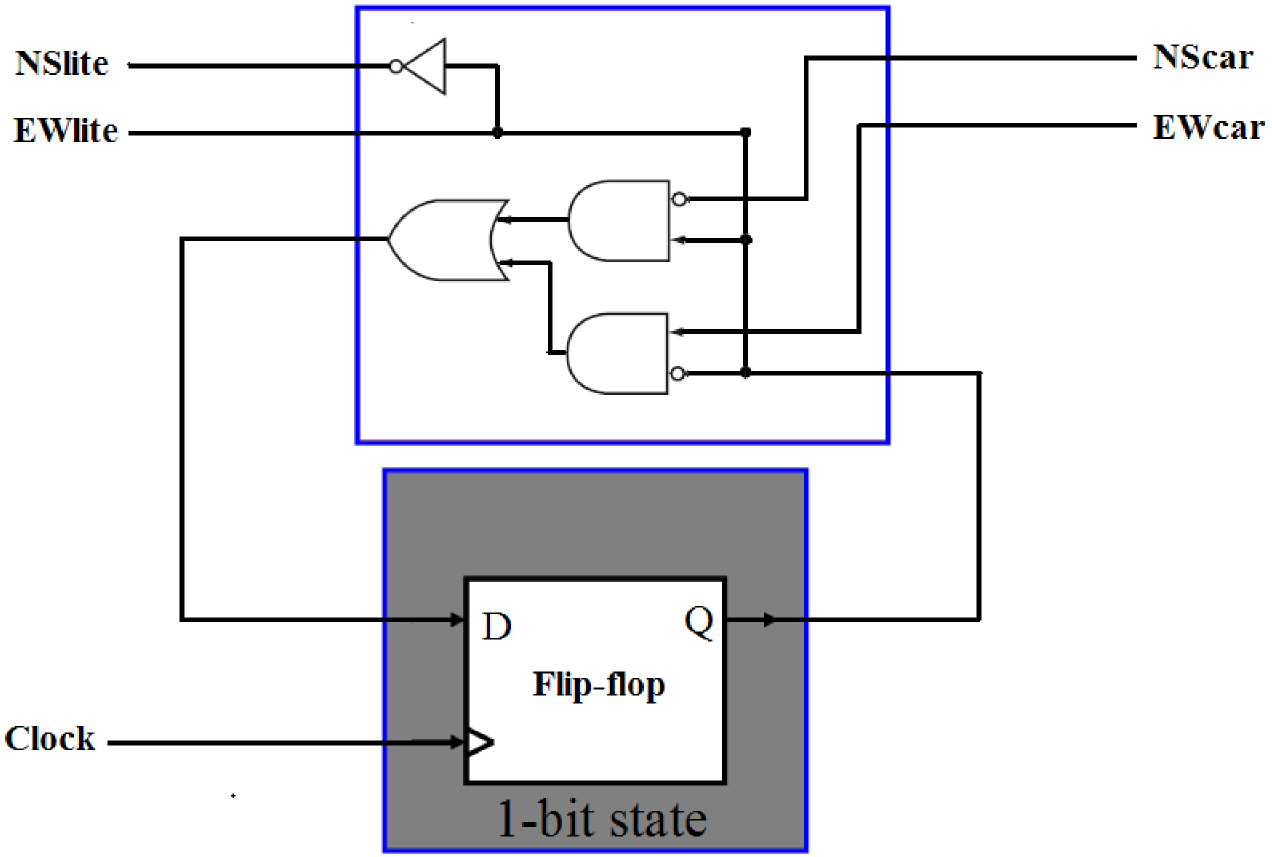
Current state	Inputs		Next state
	NScar	EWcar	
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$$\text{NextState} = (\overline{\text{CurrentState}} \cdot \text{EWcar}) + (\text{CurrentState} \cdot \overline{\text{NScar}})$$

$$\text{NSlite} = \overline{\text{CurrentState}}$$

$$\text{EWlite} = \text{CurrentState}$$

Implementing Intelligent Traffic Controller



Four Steps to Build a Finite State Machine

- Step 1 – State diagram and state table
  - There are no set procedures and diagrams. Application dependent
  - Choose a state to be the starting state when power is turned on the first time
  - A state diagram can be represented by a graph or by a table and is easy to convert between the two

Four Steps to Build a Finite State Machine

- Step 2 – State assignment
  - Assign a unique binary number to each state
  - Rewrite the state table using the assigned number for each state
- Step 3 – Combinational logic for next state function and output function
- Step 4 - Implementation

FSM

- The state is updated at the edge of the clock cycle
- The next state is computed once every clock.

Finite State Machine for a Vending Machine

Build a custom controller for a vending machine.

We could use a general purpose processor, but we might save money with a custom controller.

Take coins, give drinks

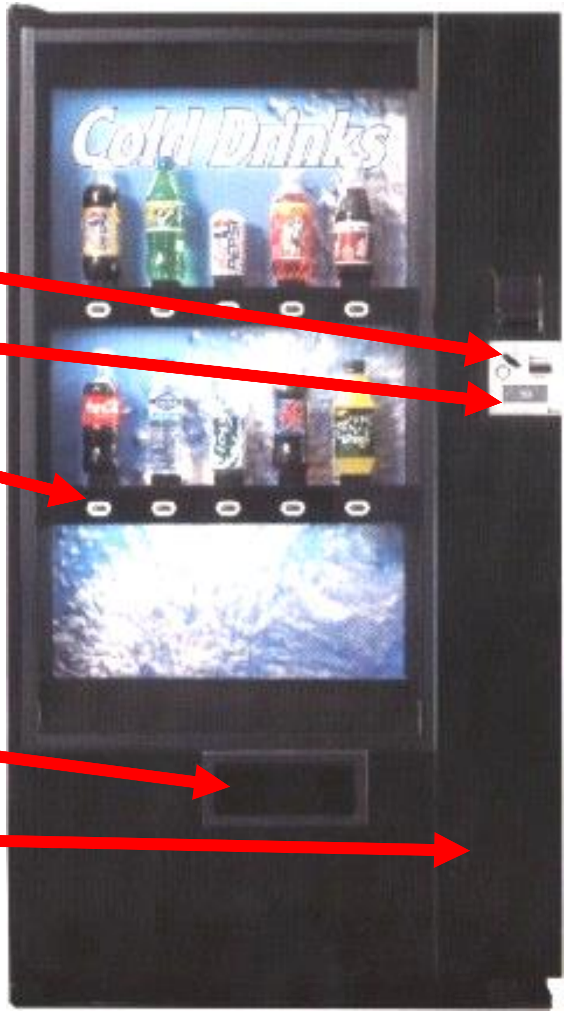
Inputs and Outputs

Inputs:

- coin trigger
- refund button
- drink selectors

Outputs:

- drink release latches
- Coin refund latch



Specifications

- Sells only two kinds of drinks, A,B. (For now, assume all drinks are available in the machine.)
- All drinks are $0.50.
- Accepts quarters only. If you put in more than $0.50, consider it as $0.50.
- Will respond to refund button. If pressed, release all quarters.
- If the current amount of money is less than $0.50
  - Will not respond to select buttons.
- If the current amount of money is $0.50
  - Will respond to select buttons. If SA is pressed, release drink A, if SB is pressed, release drink B. Then, take in all the money.

Controller Outputs

- Suppose the latch to be used to build the vending machine is controlled by one bit. It will be closed if the control signal is 0. If the control signal is 1 for a duration of one clock cycle, it will open for a period of time sufficient to allow things stored to fall through; after that, if the control signal is 0, it will be closed. If it is 1, it will stay open until the control signal returns to 0.
- Controller outputs are: L_A, L_B, L_RF, L_TK. These are signals to control the latches.
  - L_A=1, the latch for drink A opens, and drink A will fall out.
  - L_B=1, the latch for drink B opens, and drink B will fall out.
  - L_RF=1, the latch for coin refund opens, and coins will fall out.
  - L_TK = 1, the latch for coin take opens, and coins will fall from the temporarily storing place to the inside of the machine.
  - Based on the specification of the latches, we need to set the control signals to be 1 for one clock.

Inputs

- Inputs include some buttons: SA, SB, RF.
  - SA = 1 when the user is pressing the select A button, else it is 0
  - SB = 1 when the user is pressing the select B button, else it is 0
  - RF = 1 when the user is pressing the refund button, else it is 0
- Inputs also include CIS (coin insert). When a coin is falling in, CIS is 1 for one clock cycle (from one falling edge to the next falling edge). It is 0 all other time.

Design

- How to design this controller, given the specifications and the inputs and the outputs?
- Is this a stateless controller, or a controller with states?

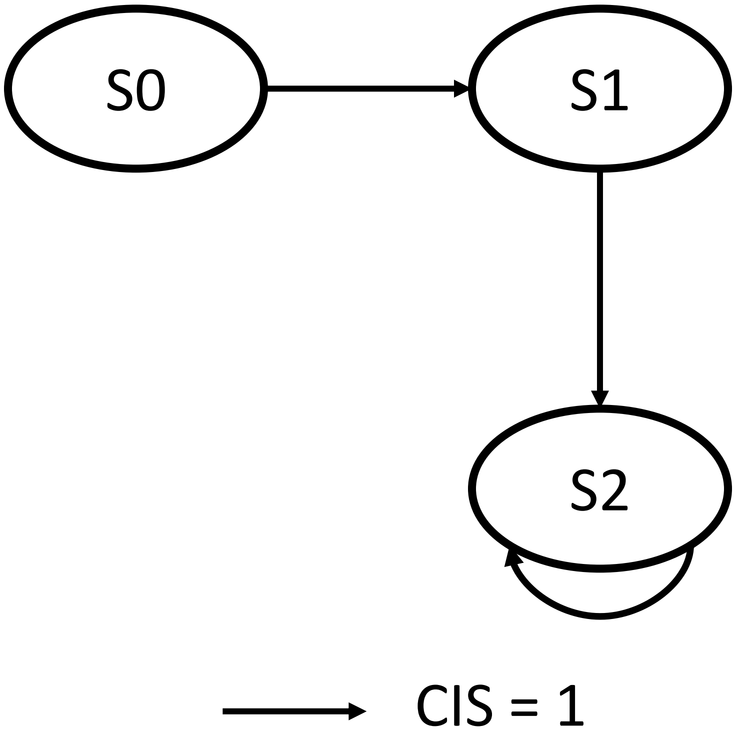
State

- To tell if a controller has states or not, the simplest way is to check if the controller’s output is relevant to what happened in the past. If it is relevant, it has state; otherwise it does not.
- The vending machine controller has state, because the controller’s response to the same input (e.g., SA) is different depending on the number of quarters inserted.

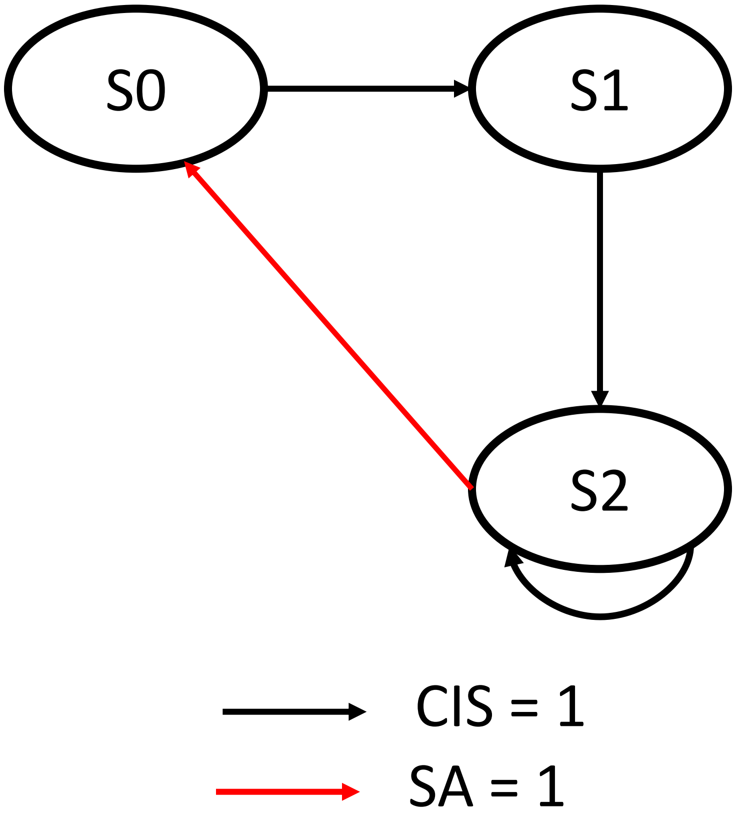
Identifying the States

- We need at least three states to remember how many quarters we have got.
  - S0: The initial state. Got 0 quarters.
  - S1: Got 1 quarter.
  - S2: Got 2 quarters.

State Diagram

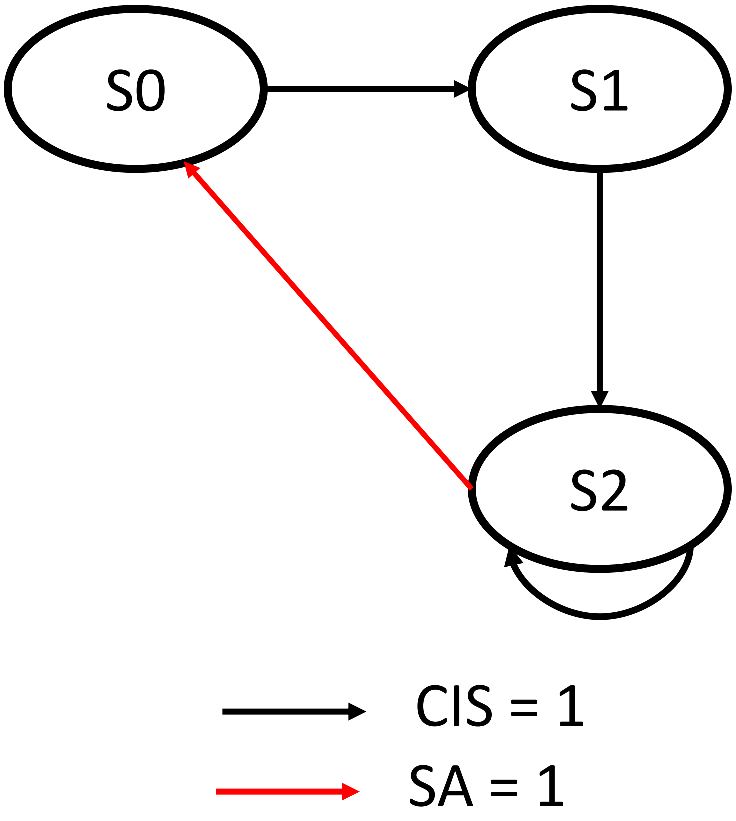


State Diagram



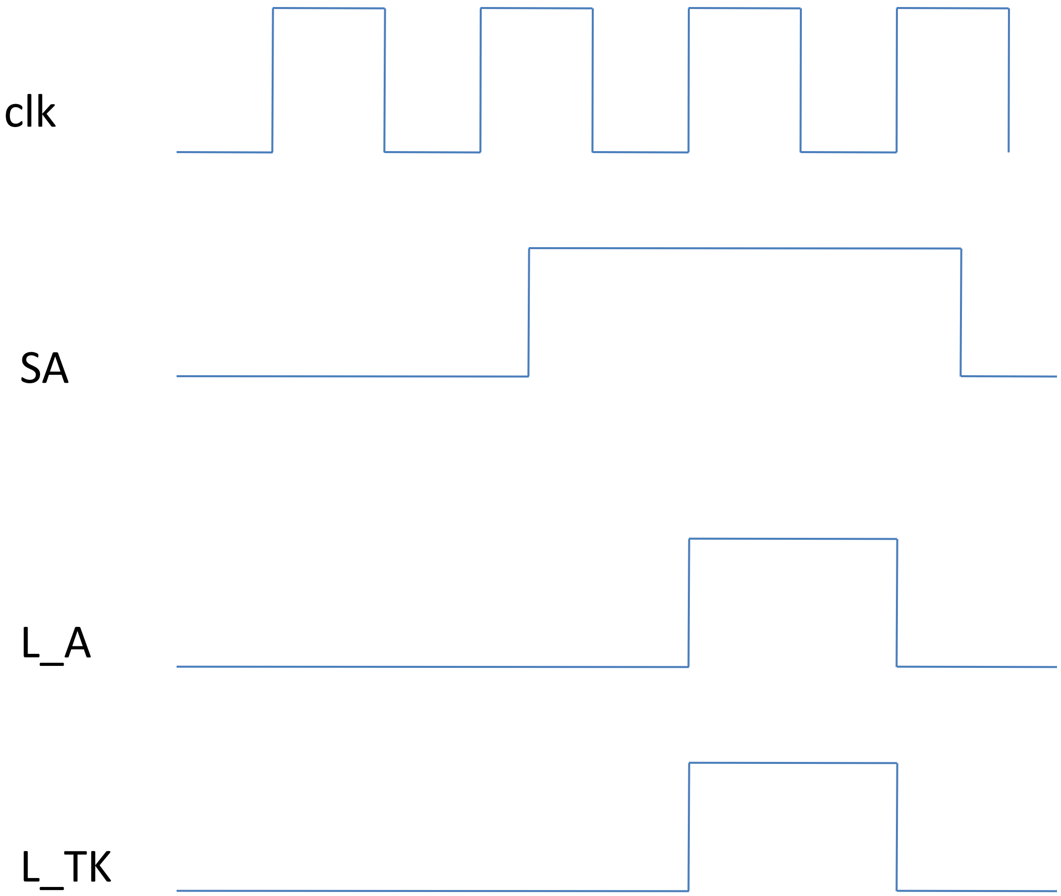
- When got $0.50, if the user presses select button, should release drink, take money, and go back to state S0.
- But is this diagram correct?

State Diagram



- Not complete – we haven’t take action yet.
- When the SA is pressed, the controller should change some output signal – not shown in the diagram

Change the output

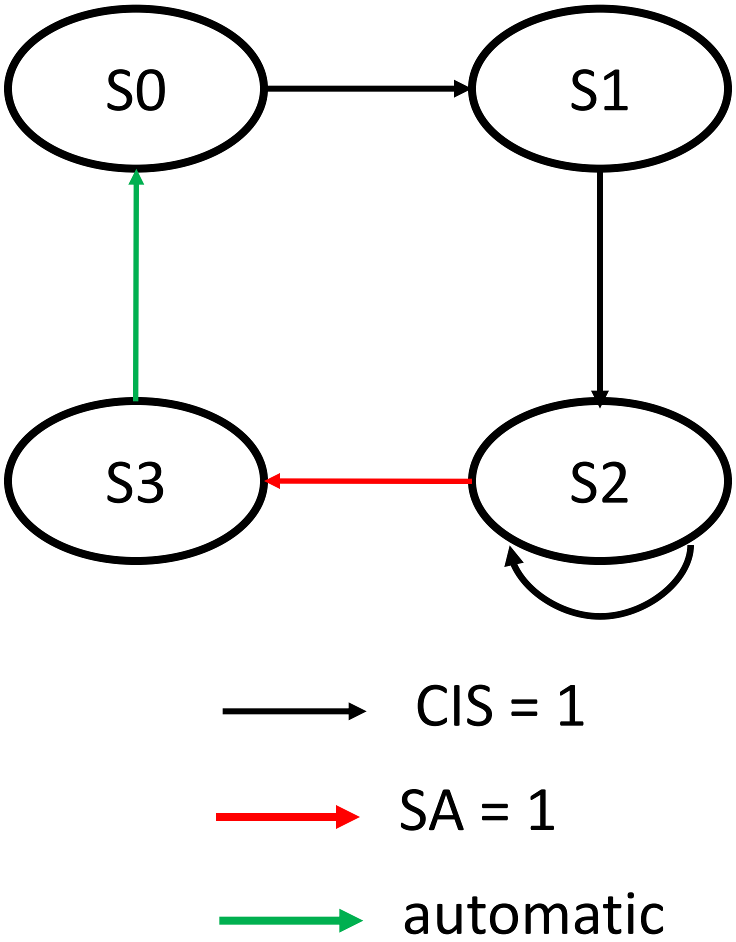


- The output to be changed, clearly, is the L_A and L_TK.
- By the specification, we should let them be 1 for one clock cycle.

Other options

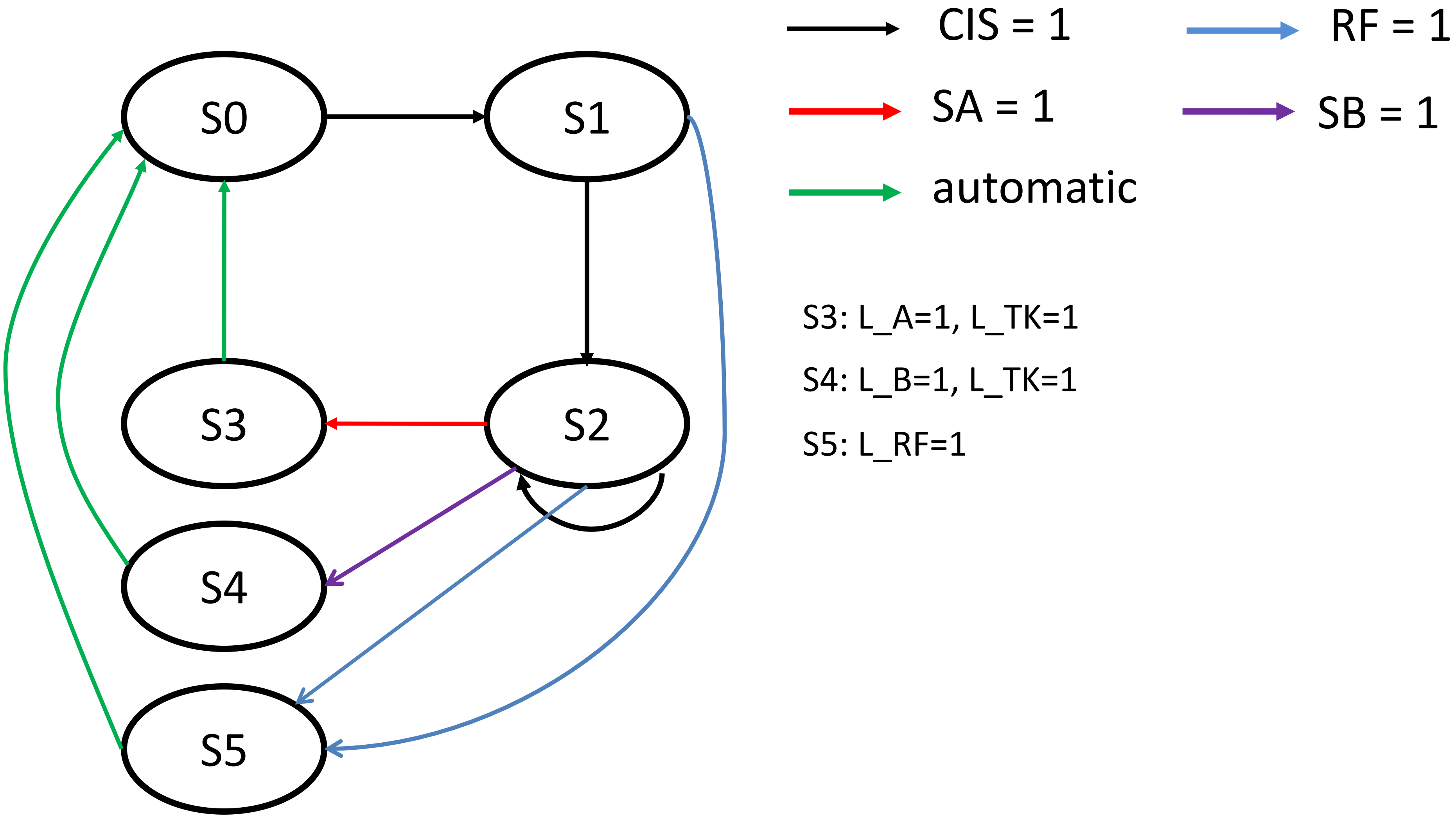
- Can we just let the latch be controlled by the SA button, meaning that the latch is open when SA is pressed?
- If we do this, I will just get free drinks.
- So the latch has to be determined by the states somehow.
- Can we just say that the latch is open if in state S2 and when SA is pressed?
- When in state S2, if SA is pressed, the next state is not S2 – the overlapping time may not be enough because SA can become 1 at arbitrary time.

The Action State For SA



- To ensure that the control signal stays high for one clock cycle, we need another state.
- In S3,
  - L_A = 1
  - L_TK = 1

The complete diagram



S3: L_A=1, L_TK=1

S4: L_B=1, L_TK=1

S5: L_RF=1

In class exercise