
CSE 20221: Logic Design

FSM Design Example In Class Work

Automatic Garage Door Opener

In Class Work

- The format of the lecture will be as follows
 - Students work together on one of the FSM design steps
 - An example of the design step results is presented and discussed
 - The process is repeated for each subsequent design step

Problem Statement

- Design a controller for an automatic garage door opener.
- Write a description of the functional behavior for the automatic garage door controller

Version I, General Description

- When a button is pushed the garage door goes up unless the door is all the way up, then it goes down.
- The door will move up until it is all the way up or the button is pushed; in that case the door stops and the next time the button is pushed the door moves down.
- The door will move down until it is all the way down or the button is pushed; in that case the door immediately begins to move up.
- When power is first turned on, the door will move up the first time the button is pressed unless the door is all the way up.

Step 1

- Identify the inputs and outputs for the automatic garage door opener.
- Provide a description of the function of each I / O

Inputs / Outputs

NAME	TYPE	FUNCTION
activate (A)	input	starts the door to go up or down or stops the motion
Up_limit (UPL)	input	indicates maximum upward travel
Dn_limit (DNL)	input	indicates maximum downward travel
Motor_up (MU)	output	Causes motor to run in direction to raise the door
Motor_dn (MD)	output	Causes motor to run in direction to lower door

Step 2

- Modify the functional description to include the names of the inputs and outputs. This should clearly describe the behavior in regard to the I / O signals.

Description with I/O

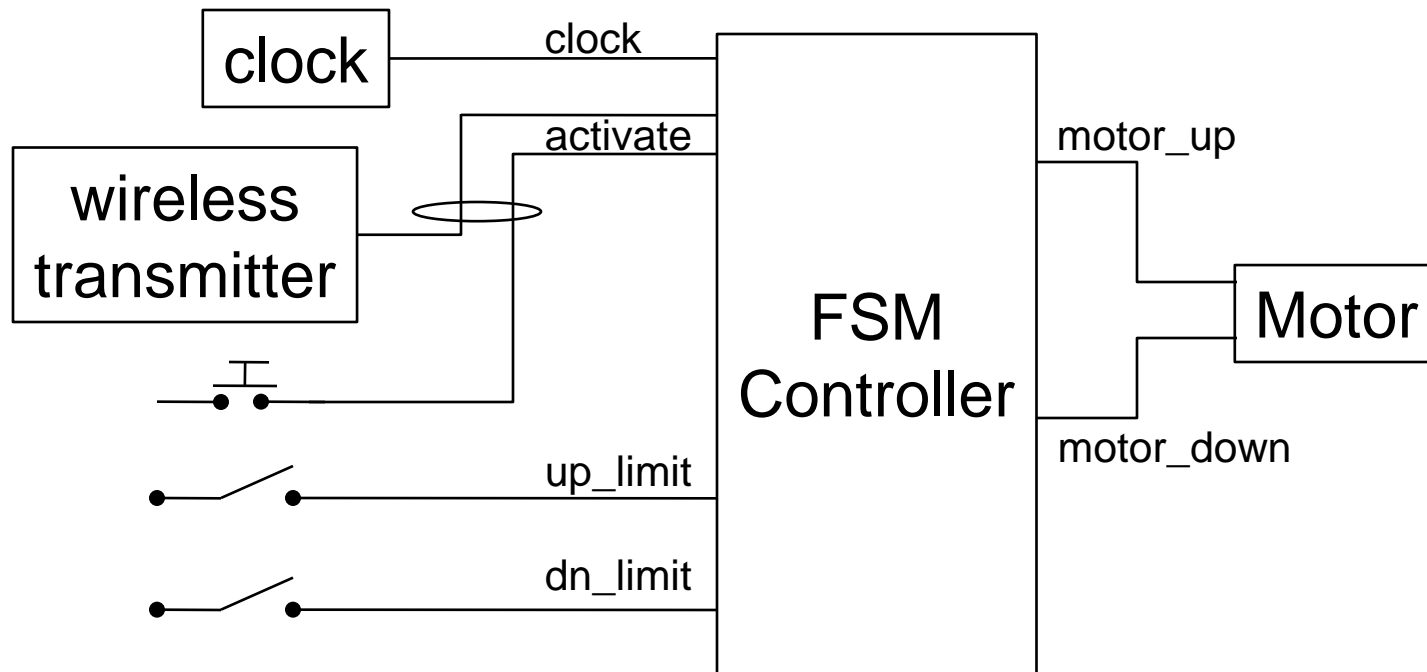
- When the activate signal is asserted the garage door goes up unless the up_limit switch is asserted, then it goes down.
- The motor_up signal remains asserted until the up_limit switch is asserted or the activate signal is asserted again.
- The motor_down signal remains asserted until the down_limit switch is asserted or the activate signal is asserted again.
- Asserting the activate signal while the garage door is moving up causes the door to stop. When asserted again the door continues to move up.
- Asserting the activate signal while the garage door is moving down causes the door to change direction and move up.
- Upon the initial state, if no limit switch is asserted then the door must move up when the activate signal is asserted.

Step 3

- Sketch a block diagram of the system. This should include the FSM controller and all interacting signals, modules and hardware devices.

Block Diagram

Automatic Garage Door Controller



Step 4

- Identify all the unique states. Refer to the functional description to help identify the different states

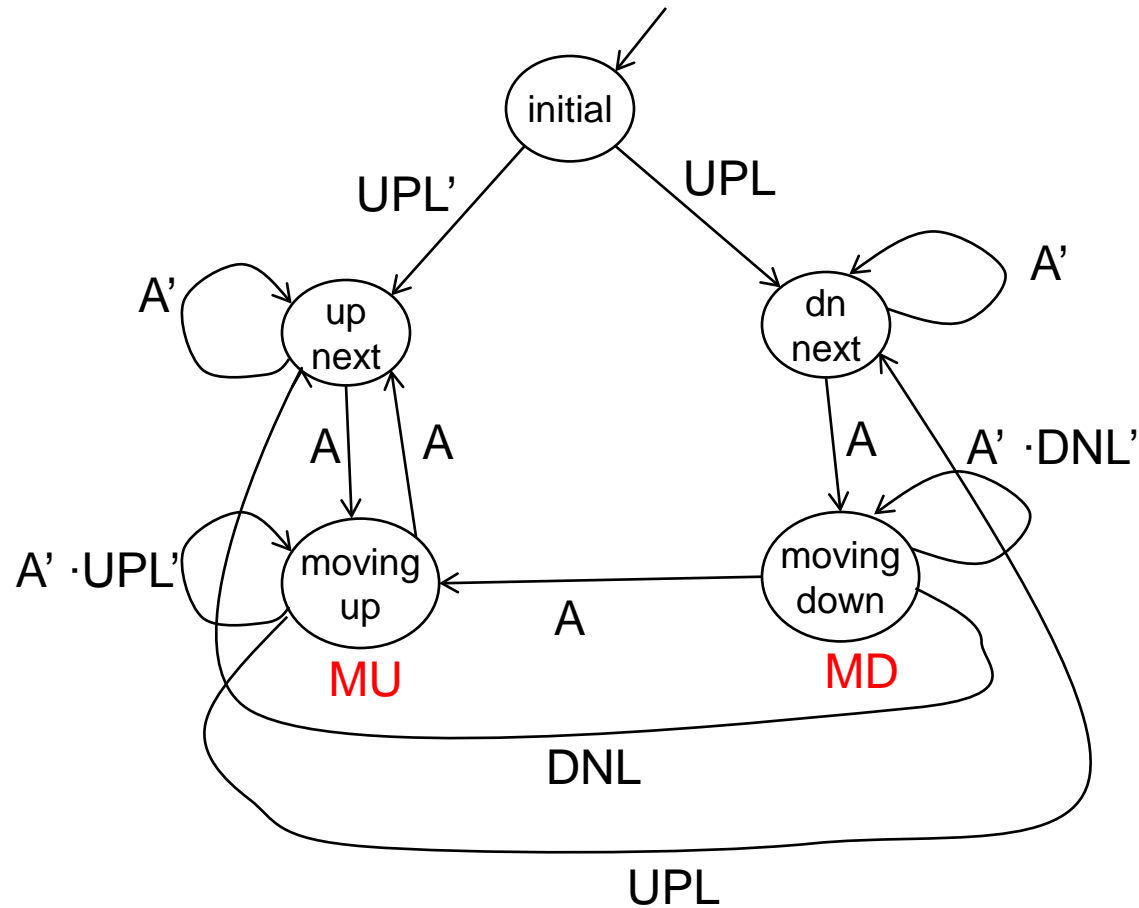
Possible States

- When the activate signal is asserted the garage door goes up (up next) unless the up_limit switch is asserted, then it goes down (down next).
- The motor_up signal remains asserted until the up_limit switch is asserted or the activate signal is asserted again (moving up).
- The motor_down signal remains asserted until the down_limit switch is asserted or the activate signal is asserted again (moving down).
- Asserting the activate signal while the garage door is moving up causes the door to stop (moving up). When activated again the door continues to move up (up next).
- Asserting the activate signal while the garage door is moving down causes the door to change direction and move up (moving down).
- Upon the initial state, if no limit switch is asserted then the door must move up when the activate signal is asserted.

Step 5

- Draw the state diagram

State Diagram

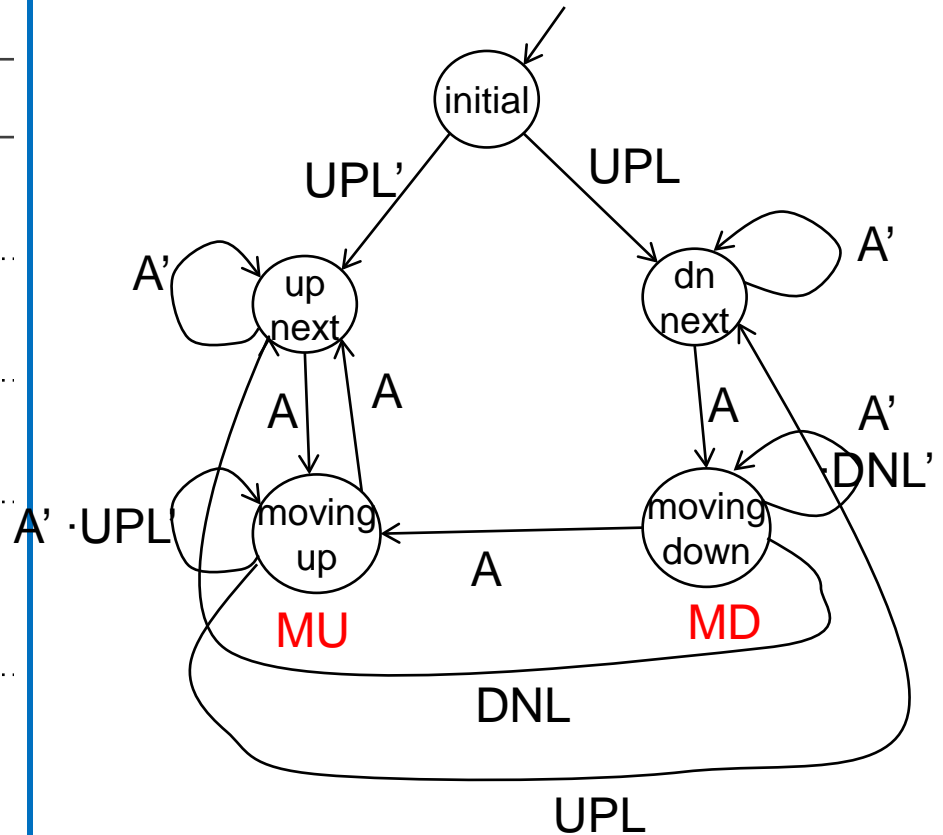


Step 6

- Draw the present / next state table, complete with all inputs and outputs.

State Table

present state	next state	outputs	
		MD	MU
initial	UPL : dn next UPL' : up next	0	0
dn next	A' : dn next A : moving dn	0	0
up next	A' : up next A : moving up	0	0
moving down	DNL : up next A : moving up DNL' · A' : mv dn*	1	0
moving up	UPL : dn next A : dn next UPL' · A' : mv up*	0	1



* may be omitted since it is assumed the controller remains in the present state unless an exit condition is satisfied.

Step 7

- Make the state assignments and update the state table

State Assignments

present state	next state	outputs	
		MD	MU
initial	UPL : dn next UPL' : up next	0	0
dn next	A' : dn next A : moving dn	0	0
up next	A' : up next A : moving up	0	0
moving down	DNL : up next A : moving up	1	0
moving up	UPL : dn next A : up next	0	1

present state S2 S1 S0	next state	D2 D1 D0	outputs	
			MD	MU
0 0 0	UPL UPL'	0 0 1 0 1 0	0	0
0 0 1	A' A	0 0 1 0 1 1	0	0
0 1 0	A' A	0 1 0 1 0 0	0	0
0 1 1	DNL A A'·DNL'	0 1 0 1 0 0 0 1 1	1	0
1 0 0	UPL A A'·UPL'	0 0 1 0 1 0 1 0 0	0	1
x x x		x x x	0	0

Step 8

- Determine the logic equations for the next state decoder.

K Maps

s1 s0 s2 \	00	01	11	10
0	initial	dn next	move down	up next
1	move up	x	x	x

s1 s0 s2 \	00	01	11	10
0	UPL'	A	A'·DNL'+ DNL	A'
1	A	x	x	x

$$D1 = UPL' \cdot S2' \cdot S1' \cdot S0' + A \cdot S1' \cdot S0 + (A' \cdot DNL' + DNL) \cdot S1 \cdot S0 + A' \cdot S1 \cdot S0' + AS2$$

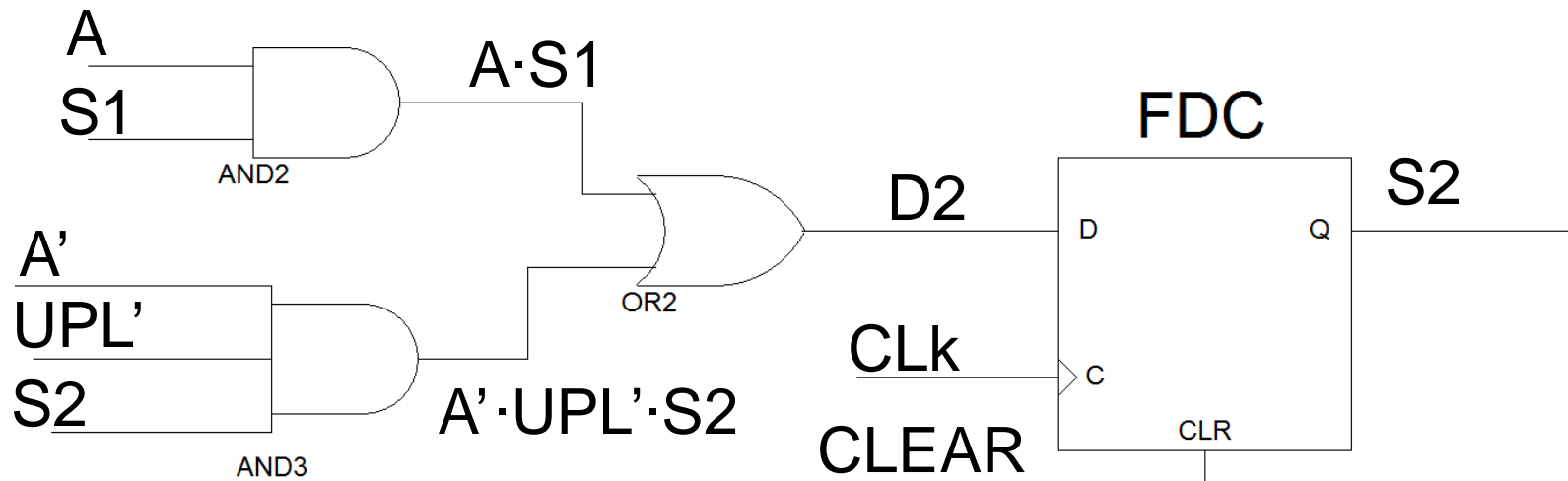
s1 s0 s2 \	00	01	11	10
0	UPL	1	A'·DNL'	0
1	UPL	x	x	x

$$D0 = UPL \cdot S2' \cdot S1' \cdot S0' + S1' \cdot S0 + A' \cdot DNL' \cdot S1 \cdot S0 + UPL \cdot S2$$

s1 s0 s2 \	00	01	11	10
0	0	0	A	A
1	A'·UPL'	x	x	x

$$D2 = A \cdot S1 + A' \cdot UPL' \cdot S2$$

Logic Circuit for D2 and S2



Step 9

- Determine the logic equations for the output decoder

Output Equations

present state	next state			outputs	
S2 S1 S0		D2 D1 D0	MD	MU	
0 0 0	UPL UPL'	0 0 1 0 1 0	0	0	
0 0 1	A' A	0 0 1 0 1 1	0	0	
0 1 0	A' A	0 1 0 1 0 0	0	0	
0 1 1	DNL A	0 1 0 1 0 0	1	0	
1 0 0	UPL A	0 0 1 0 0 1	0	1	
x x x		x x x	0	0	

$$MD = S2' \cdot S1 \cdot S0$$

$$MU = S2 \cdot S1' \cdot S0'$$

“don’t care” conditions are not used because of possible dangerous condition resulting from motor running at wrong time

Step 10 - 12

- 10. Implement the logic equations
- 11. Simulate the design and verify correct functionality
- 12. Correct the design if necessary

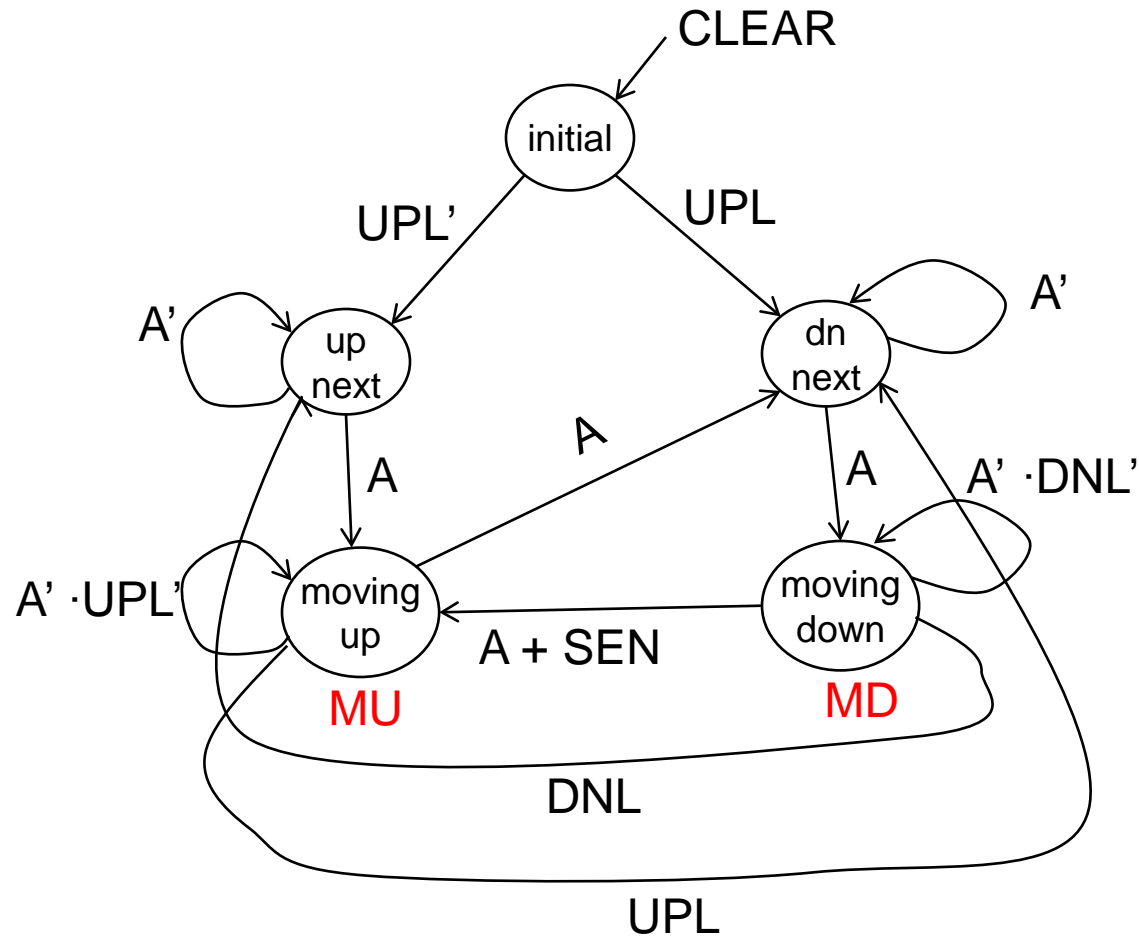
Simulation Results



Design Revision Lab Assignment

- Modify the garage door opener design by including a sensor (sen) that detects if there is an obstruction in the path of the door.
 - If an object is in the path of the door when it is closing, the controller will cause the door motion to change from down to up.
- Simulate the results of your design in lab during the week of March 1. ~~Turn in an annotated simulation timing diagram that clearly indicates all the possible branching.~~ Using the simulation results, show the lab TA that your circuit follows all possible branching correctly.
- Include an asynchronous reset that will force the controller into the initial state.

Revised State Diagram



Revised State Assignments

present state	next state	outputs	
		MD	MU
initial	UPL : dn next UPL' : up next	0	0
dn next	A' : dn next A : moving dn	0	0
up next	A' : up next A : moving up	0	0
moving down	DNL : up next A+SEN : moving up	1	0
moving up	UPL : dn next A : up next	0	1

present state S2 S1 S0	next state	D2 D1 D0	outputs	
			MD	MU
0 0 0	UPL UPL'	0 0 1 0 1 0	0	0
0 0 1	A' A	0 0 1 0 1 1	0	0
0 1 0	A' A	0 1 0 1 0 0	0	0
0 1 1	DNL A+SEN A'·DNL' ·SEN'	0 1 0 1 0 0 0 1 1	1	0
1 0 0	UPL A A'·UPL'	0 0 1 0 1 0 1 0 0	0	1
x x x		x x x	0	0

Revised K Maps

s1 s0		00	01	11	10
s2					
0		initial	dn next	move down	up next
1		move up	x	x	x

s1 s0		00	01	11	10
s2					
0		UPL'	A	A' · DNL' · SEN' + DNL	A'
1		A	x	x	x

$$D1 = UPL' \cdot S2' \cdot S1' \cdot S0' + A \cdot S1' \cdot S0 + (A' \cdot DNL' \cdot SEN' + DNL) \cdot S1 \cdot S0 + A' \cdot S1 \cdot S0' + AS2$$

s1 s0		00	01	11	10
s2					
0		UPL	1	A' · DN' · SEN'	0
1		UPL	x	x	x

$$D0 = UPL \cdot S2' \cdot S1' \cdot S0' + S1' \cdot S0 + A' \cdot DNL' \cdot SEN' \cdot S1 \cdot S0 + UPL \cdot S2$$

s1 s0		00	01	11	10
s2					
0	0	0	0	A + SEN	A
1	A' · UPL'	x	x	x	x

$$D2 = A \cdot S1 + A' \cdot UPL' \cdot S2 + SEN \cdot S1 \cdot S0$$

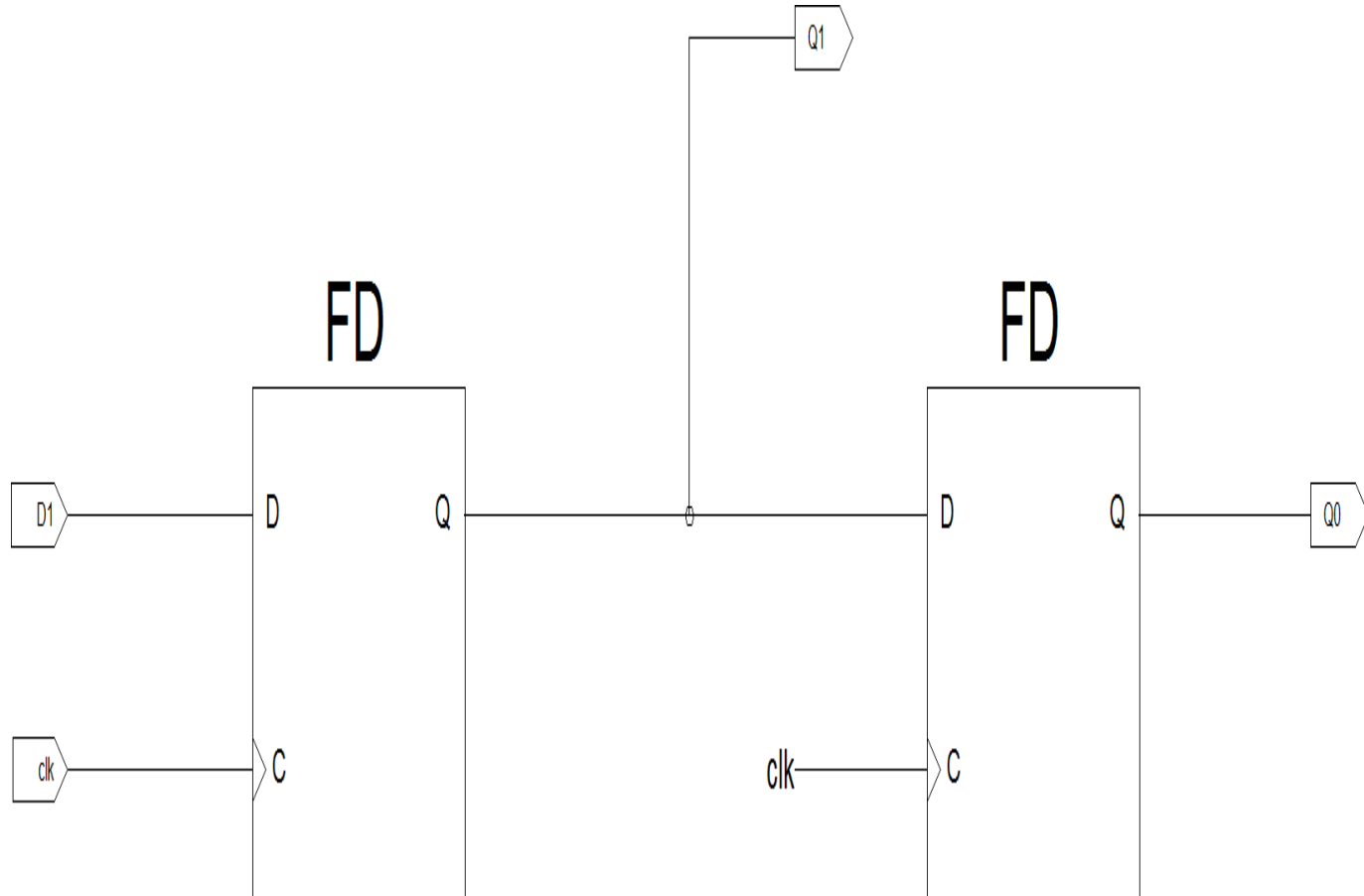
Testbench for Sequential Circuits

- Sequential circuits require a clock
 - The following statements will generate a clock

```
// Clock Procedure
always begin
    #100; clk <= 0;
    #100; clk <= 1;
end
```

- The above procedure will generate a continuous running clock signal until the simulator stops

A Sequential Circuit Example

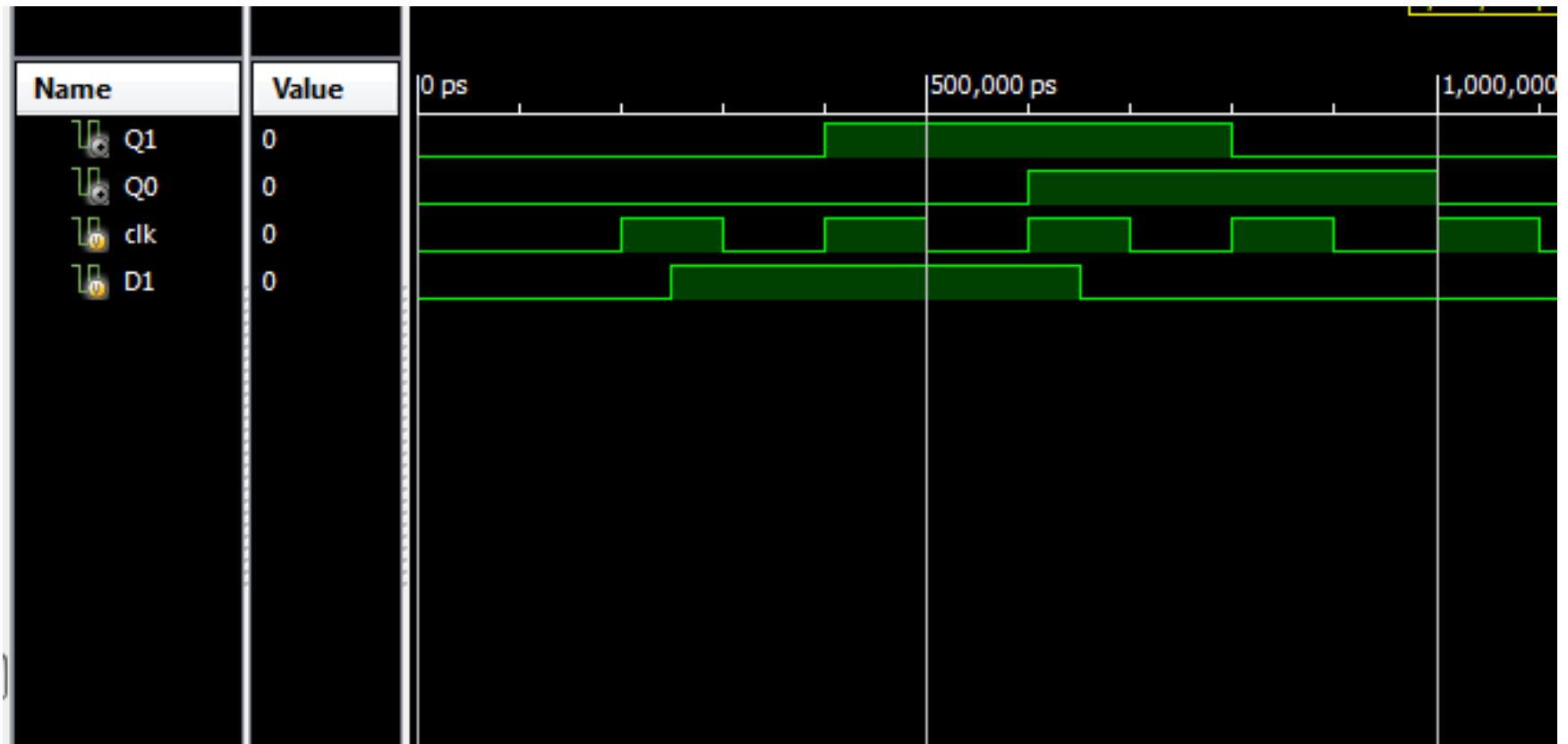


Sequential Testbench Example

```
// Clock Procedure
always begin
    #100; clk <= 0;
    #100; clk <= 1;
end    // Note: Procedure repeats until simulation ends

initial begin
    clk = 0; //initialize clk
    D1 = 0; //initialize D1
    //ALL INPUTS MUST BE INITIALIZED, otherwise the simulator will not assign values
    @(posedge clk); //wait for next clock pulse
    #50; D1 = 1;    // change values when clock is stable
    @(posedge clk); //wait for next clock pulse
    @(posedge clk); //wait for next clock pulse
    #50; D1 = 0;
    @(posedge clk); //wait for next clock pulse
    @(posedge clk); //wait for next clock pulse
end;
```


Simulation Results for Example Circuit



Homework

- Problems: chapter 3.25, 3.27, 3.29, 3.38, 3.46¹
- Homework is due on Thursday, March 4

¹reverse engineering in this case means: draw the state diagram and be sure to identify the states in terms of S1 and S0