Synchronous Sequential Circuits - Part II

VOLKAN KURSUN

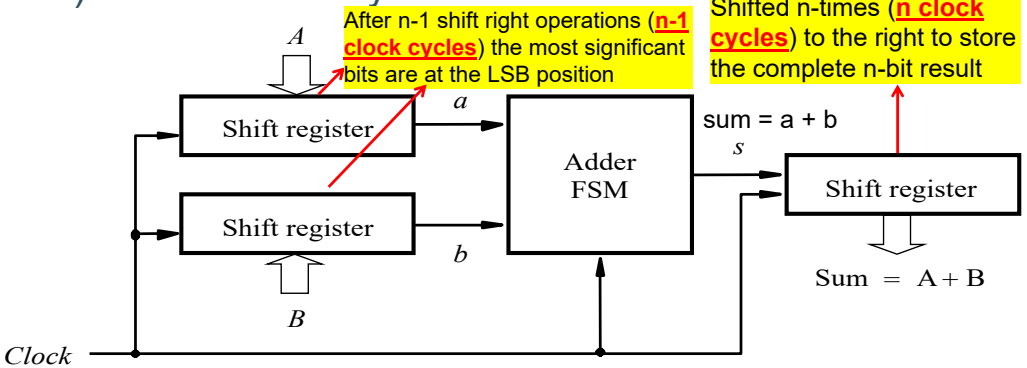
Some material from McGraw Hill

Outline

- Serial Adder
- Digital Door Lock
- Vending Machine
- State Minimization

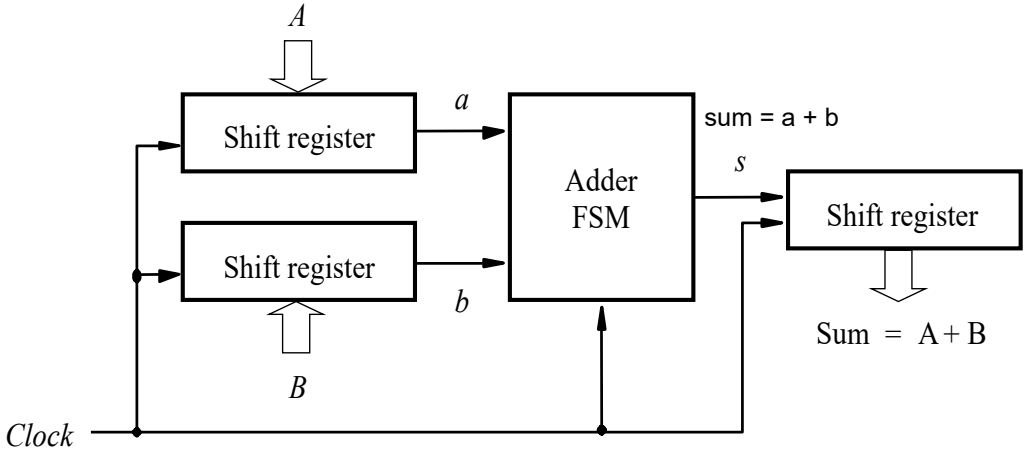
Serial Adder

- Parallel adders are fast but also complex and costly
- **If speed is not very important**, a **serial adder is a low-cost alternative**
- **Serial adder**: **bits are added a pair at a time**
- Design a serial adder that adds a pair of bits (a and b) in each clock cycle



Serial Adder FSM

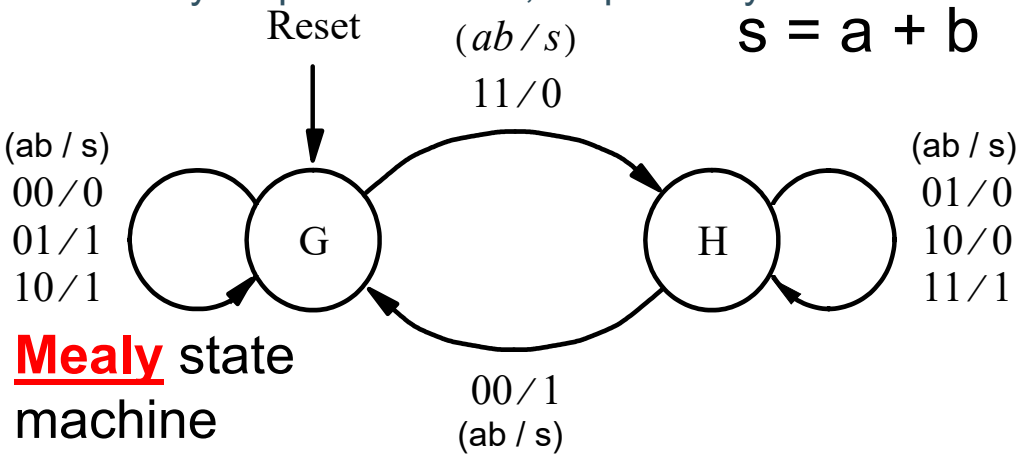
- The **adder needs to be an FSM** to be able to **hold the carry generated at each clock cycle**
- Two states G and H are needed for the states where the carry output is 0 and 1, respectively



Serial Adder State Diagram

- Two states G and H are needed for the states where the carry output is 0 and 1, respectively

s = a + b



Mealy state machine (**output on arcs**)

G: carry-in = 0

H: carry-in = 1

Serial Adder State Table

- A single flip-flop is needed to represent the two states G and H

Present state	Next state				Output s			
	ab =00	01	10	11	00	01	10	11
G	G	G	G	H	0	1	1	0
H	G	H	H	H	1	0	0	1

Serial Adder State Assigned Table

Present state	Next state				Output			
	ab =00	01	10	11	00	01	10	11
y	Y				s			
0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	0	0	1

- Derive the next state and output logic circuits

y \ ab	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Y = ab + yb + ya

y \ ab	00	01	11	10
0	0	1	0	1
1	1	0	1	0

s = ya'b' + y'a'b + yab + y'ab'

s = y(a'b' + ab) + y'(a'b + ab')

s = y(a ⊕ b)' + y'(a ⊕ b)

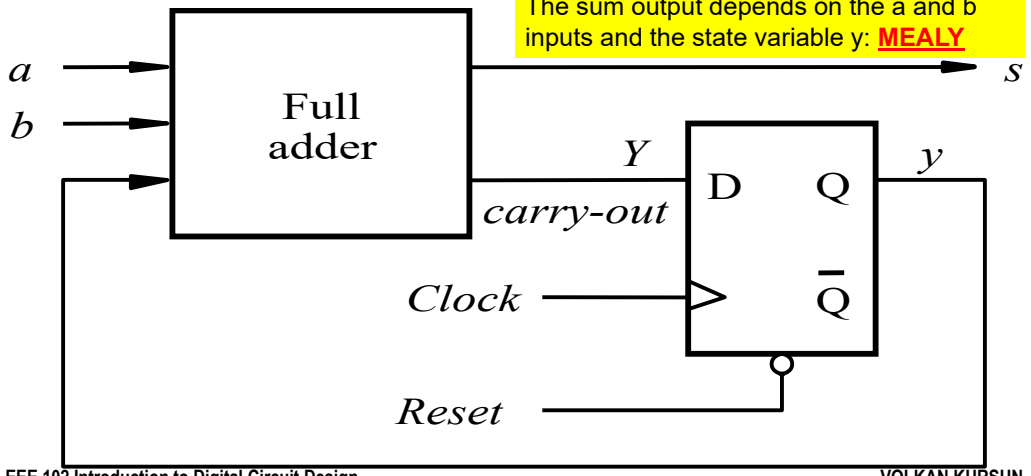
s = y ⊕ a ⊕ b

Serial Adder Mealy FSM Circuit

- The serial adder can be used to add numbers of any length depending on the size of the shifters (with **n-bit shifters** you can **serially add n-bit numbers**)

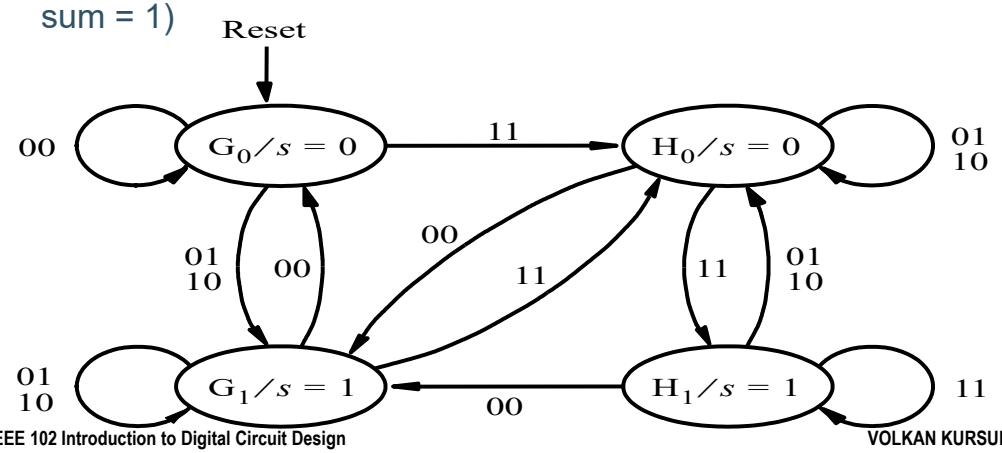
Y = ab + yb + ya s = y ⊕ a ⊕ b

The sum output depends on the a and b inputs and the state variable y: **MEALY**



Moore FSM For Serial Adder

- In a Moore machine, the outputs can only be determined by the state of the machine
- For both the carry-0 and carry-1 states, the sum can be either 0 or 1: split each of the G and H states of the Mealy machine into two states in the Moore machine: G0 (carry = 0, sum = 0), G1 (carry = 0, sum = 1), H0 (carry = 1, sum = 0), and H1 (carry = 1, sum = 1)



Moore Serial Adder State Table

- In a Moore machine, the outputs can only be determined by the state of the machine
- For both the carry-0 and carry-1 states, the sum output can be either 0 or 1: G0 (carry = 0, sum = 0), G1 (carry = 0, sum = 1), H0 (carry = 1, sum = 0), and H1 (carry = 1, sum = 1)

Present state	Nextstate				Output
	ab =00	01	10	11	s
G_0	G_0	G_1	G_1	H_0	0
G_1	G_0	G_1	G_1	H_0	1
H_0	G_1	H_0	H_0	H_1	0
H_1	G_1	H_0	H_0	H_1	1

Moore Serial Adder State Assigned Table

- G0 (carry = 0, sum = 0), G1 (carry = 0, sum = 1), H0 (carry = 1, sum = 0), and H1 (carry = 1, sum = 1)

Present state y_2y_1	Nextstate Y_2Y_1				Output s
	ab =00	01	10	11	
00	00	01	01	10	0
01	00	01	01	10	1
10	01	10	10	11	0
11	01	10	10	11	1

y_2y_1 \ ab	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	1	1	1
10	0	1	1	1

$Y_2 = ab + y_2b + y_2a$

y_2y_1 \ ab	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	1	0	1	0
10	1	0	1	0

$s = y_1$

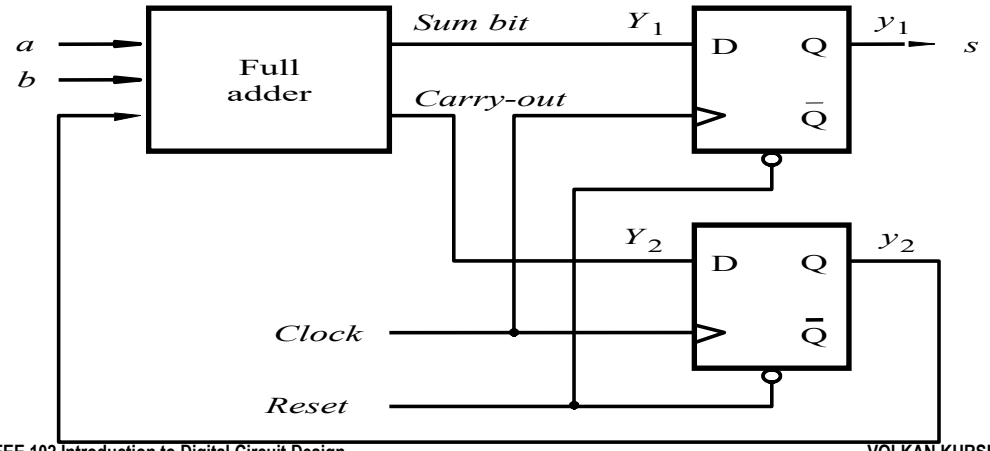
$Y_1 = y_2a'b' + y_2'a'b + y_2ab + y_2'ab' = y_2 \oplus a \oplus b$

Serial Adder Moore FSM Circuit

- In the Moore machine, the sum output is stored in a flip-flop, thereby delaying availability of the sum bit for the shifter by one clock cycle as compared to the Mealy machine

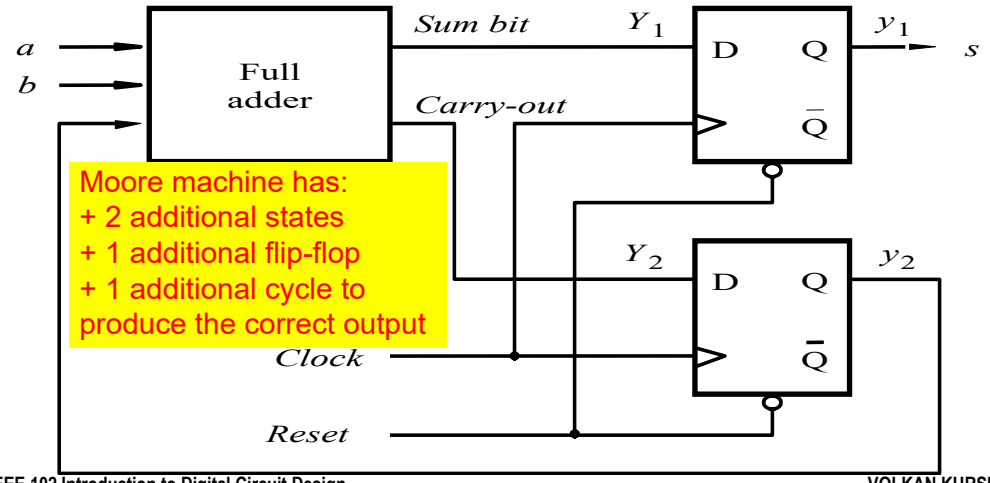
$Y_2 = ab + y_2b + y_2a$ $\quad s = y_1$

$Y_1 = y_2a'b' + y_2'a'b + y_2ab + y_2'ab' = y_2 \oplus a \oplus b$



Moore versus Mealy Serial Adder FSM

- In the **Mealy machine**, **change in the inputs is reflected in the output after the combinational circuit (full adder) delay**
- In the **Moore machine**, the sum output does not change until the change in the inputs moves the machine into a new state, which happens **one clock cycle later**



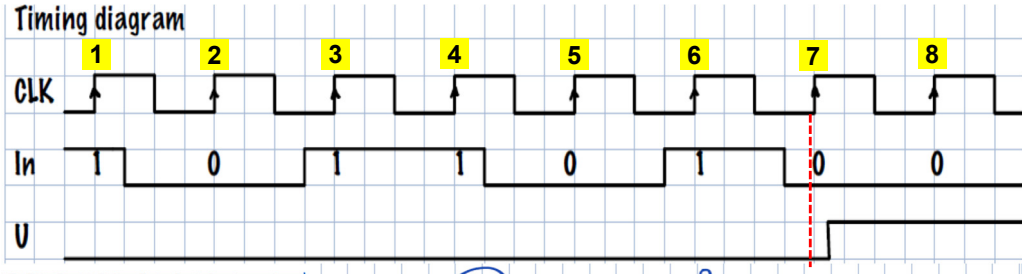
Moore machine has:
+ 2 additional states
+ 1 additional flip-flop
+ 1 additional cycle to produce the correct output

Outline

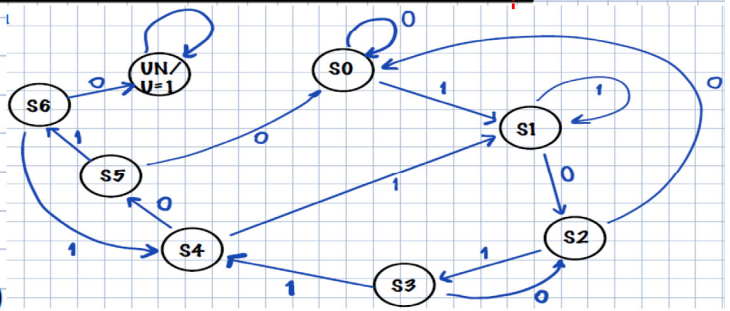
- Serial Adder
- Digital Door Lock
- Vending Machine
- State Minimization

Door Unlock Sequence

- Unlock the door when the 0b**1011010** bit sequence is detected
- State machine with 8 states: S0, S1, S2, S3, S4, S5, S6, UN



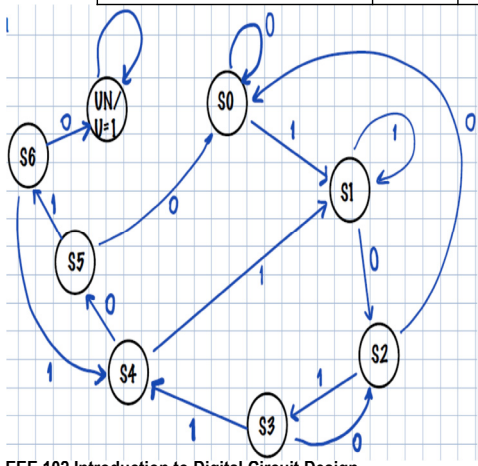
S0: Initial state
S1: Recall 1
S2: Recall 10
S3: Recall 101
S4: Recall 1011
S5: Recall 10110
S6: Recall 101101
UN: Recall 1011010



State Table

- 8 states (S0, S1, S2, S3, S4, S5, S6, UN): 3 flip-flops needed
- Unlock the door when the 0b**1011010** bit sequence is detected

Binary Sequence	1	0	1	1	0	1	0
State Name	S1	S2	S3	S4	S5	S6	UN
State Assigned Code	001	010	011	100	101	110	111

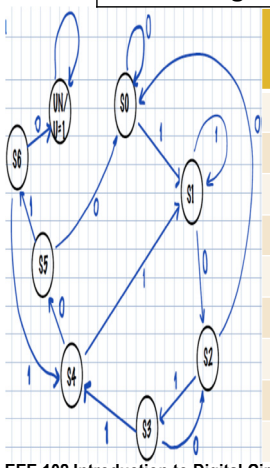


Present State	Next State		Output (U)
	In = 0	In = 1	
S0	S0	S1	0
S1	S2	S1	0
S2	S0	S3	0
S3	S2	S4	0
S4	S5	S1	0
S5	S0	S6	0
S6	UN	S4	0
UN	UN	UN	1

State Assigned Table

- 8 states (S0, S1, S2, S3, S4, S5, S6, UN): 3 flip-flops needed
- Unlock the door when the 0b**1011010** bit sequence is detected

Binary Sequence	1	0	1	1	0	1	0
State Name	S1	S2	S3	S4	S5	S6	UN
State Assigned Code	001	010	011	100	101	110	111



Present State Name	Present State Code			Next State Code In = 0			Next State Code In = 1			Output
	q_2	q_1	q_0	D_2	D_1	D_0	D_2	D_1	D_0	U
S0	0	0	0	0	0	0	0	0	1	0
S1	0	0	1	0	1	0	0	0	1	0
S2	0	1	0	0	0	0	0	1	1	0
S3	0	1	1	0	1	0	1	0	0	0
S4	1	0	0	1	0	1	0	0	1	0
S5	1	0	1	0	0	0	1	1	0	0
S6	1	1	0	1	1	1	1	0	0	0
UN	1	1	1	1	1	1	1	1	1	1

Next State Combinational Logic

- Derive the next state expressions using D flip-flops to implement the circuit: find the flip-flop input equations (next state equations, Q(t+1) = D)

Present State Code			Next State Code In = 0			Next State Code In = 1		
q_2	q_1	q_0	D_2	D_1	D_0	D_2	D_1	D_0
0	0	0	0	0	0	0	0	1
0	0	1	0	1	0	0	0	1
0	1	0	0	0	0	0	1	1
0	1	1	0	1	0	1	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	0	0	1	1	0
1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1

q_2q_1 \ q_0In	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	1	1	1	1
10	1	0	1	0

$$D_2 = q_2q_1 + q_1q_0In + q_2q_0In + q_2q_0'In'$$

Next State Combinational Logic

- Derive the next state expressions using D flip-flops to implement the circuit: find the flip-flop input equations (next state equations, Q(t+1) = D)

Present State Code			Next State Code In = 0			Next State Code In = 1		
q_2	q_1	q_0	D_2	D_1	D_0	D_2	D_1	D_0
0	0	0	0	0	0	0	0	1
0	0	1	0	1	0	0	0	1
0	1	0	0	0	0	0	1	1
0	1	1	0	1	0	1	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	0	0	1	1	0
1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1

q_2q_1 \ q_0In	00	01	11	10
00	0	0	0	1
01	0	1	0	1
11	1	0	1	1
10	0	0	1	0

$$D_1 = q_2q_1In' + q_2'q_0In' + q_2q_0In + q_2'q_1q_0'In$$

Next State Combinational Logic

- Derive the next state expressions using D flip-flops to implement the circuit: find the flip-flop input equations (next state equations, Q(t+1) = D)

Present State Code			Next State Code In = 0			Next State Code In = 1		
q_2	q_1	q_0	D_2	D_1	D_0	D_2	D_1	D_0
0	0	0	0	0	0	0	0	1
0	0	1	0	1	0	0	0	1
0	1	0	0	0	0	0	1	1
0	1	1	0	1	0	1	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	0	0	1	1	0
1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1

Cost: 4 inverters + 5*3-input AND gates + 1*5-input OR gate

q_2q_1 \ q_0In	00	01	11	10
00	0	1	1	0
01	0	1	0	0
11	1	0	1	1
10	1	1	0	0

$$D_0 = q_2q_0'In' + q_1'q_0'In + q_2'q_1'In + q_2'q_0'In + q_2q_1q_0$$

Next State Combinational Logic

- Derive the next state expressions using D flip-flops to implement the circuit: find the flip-flop input equations (next state equations, Q(t+1) = D)

Present State Code			Next State Code In = 0			Next State Code In = 1		
q_2	q_1	q_0	D_2	D_1	D_0	D_2	D_1	D_0
0	0	0	0	0	0	0	0	1
0	0	1	0	1	0	0	0	1
0	1	0	0	0	0	0	1	1
0	1	1	0	1	0	1	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	0	0	1	1	0
1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1

Alternative expression of D_0:

q_2q_1 \ q_0In	00	01	11	10
00	0	1	1	0
01	0	1	0	0
11	1	0	1	1
10	1	1	0	0

Cost-2: 4 inverters + 5*3-input AND gates + 1*5-input OR gate

$$D_0 = q_2q_0'In' + q_2q_1'q_0' + q_2'q_1'In + q_2'q_0'In + q_2q_1q_0$$

Next State Combinational Logic

- Derive the next state expressions using D flip-flops to implement the circuit: find the flip-flop input equations (next state equations, Q(t+1) = D)

Present State Code			Next State Code In = 0			Next State Code In = 1		
q_2	q_1	q_0	D_2	D_1	D_0	D_2	D_1	D_0
0	0	0	0	0	0	0	0	1
0	0	1	0	1	0	0	0	1
0	1	0	0	0	0	0	1	1
0	1	1	0	1	0	1	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	0	0	1	1	0
1	1	0	1	1	1	1	0	0
1	1	1	1	1	1	1	1	1

Second alternative expression of D_0:

q_2q_1 \ q_0In	00	01	11	10
00	0	1	1	0
01	0	1	0	0
11	1	0	1	1
10	1	1	0	0

Cost-3: 4 inverters + 5*3-input AND gates + 1*5-input OR gate

$$D_0 = q_2q_1In' + q_2q_1'q_0' + q_2'q_1'In + q_2'q_0'In + q_2q_1q_0$$

Output Combinational Logic

- Derive the output expression to unlock (U):

Moore machine output:

$$U = q_2q_1q_0$$

Present State Name	Present State Code			Next State Code In = 0			Next State Code In = 1			Output
	q_2	q_1	q_0	D_2	D_1	D_0	D_2	D_1	D_0	U
S0	0	0	0	0	0	0	0	0	1	0
S1	0	0	1	0	1	0	0	0	1	0
S2	0	1	0	0	0	0	0	1	1	0
S3	0	1	1	0	1	0	1	0	0	0
S4	1	0	0	1	0	1	0	0	1	0
S5	1	0	1	0	0	0	1	1	0	0
S6	1	1	0	1	1	1	1	0	0	0
UN	1	1	1	1	1	1	1	1	1	1

Door Lock with T Flip-Flops

- 3 T flip-flops are used: each state variable is held in a separate T flip-flop
- Derive the T flip-flop input equations for T_2, T_1, and T_0

T	Q(t+1)
0	Q(t)
1	$\overline{Q(t)}$

T Flip-Flop Excitation Table		
q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Present State Name	Present State Code			Next State Code In = 0						Next State Code In = 1						Output
	q_2	q_1	q_0	Q_2	Q_1	Q_0	T_2	T_1	T_0	Q_2	Q_1	Q_0	T_2	T_1	T_0	U
S0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
S1	0	0	1	0	1	0	0	1	1	0	0	1	0	0	0	0
S2	0	1	0	0	0	0	0	1	0	0	1	1	0	0	1	0
S3	0	1	1	0	1	0	0	0	1	1	0	0	1	1	1	0
S4	1	0	0	1	0	1	0	0	1	0	0	1	1	0	1	0
S5	1	0	1	0	0	0	1	0	1	1	1	0	0	1	1	0
S6	1	1	0	1	1	1	0	0	1	1	0	0	0	1	0	0
UN	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1

Door Lock with T Flip-Flops

q_2q_1 \ q_0In	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	0	0	0
10	0	1	0	1

$T_2 = q_2q_1'q_0'In + q_2'q_1q_0In + q_2q_1'q_0In'$

Present State Code			Next State Code											
			In = 0						In = 1					
q_2	q_1	q_0	Q_2	Q_1	Q_0	T_2	T_1	T_0	Q_2	Q_1	Q_0	T_2	T_1	T_0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1	0	0	1	0	0	0
0	1	0	0	0	0	0	1	0	0	1	1	0	0	1
0	1	1	0	1	0	0	0	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1	0	0	1	1	0	1
1	0	1	0	0	0	1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1	1	0	0	0	1	0
1	1	1	1	1	1	0	0	0	1	1	1	0	0	0

Door Lock with T Flip-Flops

q_2q_1 \ q_0In	00	01	11	10
00	0	0	0	1
01	1	0	1	0
11	0	1	0	0
10	0	0	1	0

$T_1 = q_2'q_1q_0'In' + q_2q_1q_0'In + q_2'q_1q_0In + q_2q_1'q_0In + q_2'q_1'q_0In'$

Present State Code			Next State Code											
			In = 0						In = 1					
q_2	q_1	q_0	Q_2	Q_1	Q_0	T_2	T_1	T_0	Q_2	Q_1	Q_0	T_2	T_1	T_0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1	0	0	1	0	0	0
0	1	0	0	0	0	0	1	0	0	1	1	0	0	1
0	1	1	0	1	0	0	0	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1	0	0	1	1	0	1
1	0	1	0	0	0	1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1	1	0	0	0	1	0
1	1	1	1	1	1	0	0	0	1	1	1	0	0	0

Door Lock with T Flip-Flops

q_2q_1 \ q_0In	00	01	11	10
00	0	1	0	1
01	0	1	1	1
11	1	0	0	0
10	1	1	1	1

Cost-1: 4 inverters + 4*3-input AND gates + 1*2-input AND gate + 1*5-input OR gate

$T_0 = q_2q_0'In' + q_2q_1' + q_2'q_0'In + q_2'q_1q_0 + q_2'q_0In'$

Present State Code			Next State Code											
			In = 0						In = 1					
q_2	q_1	q_0	Q_2	Q_1	Q_0	T_2	T_1	T_0	Q_2	Q_1	Q_0	T_2	T_1	T_0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1	0	0	1	0	0	0
0	1	0	0	0	0	0	1	0	0	1	1	0	0	1
0	1	1	0	1	0	0	0	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1	0	0	1	1	0	1
1	0	1	0	0	0	1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1	1	0	0	0	1	0
1	1	1	1	1	1	0	0	0	1	1	1	0	0	0

Door Lock with T Flip-Flops

Alternative expression for T_0:

q_2q_1 \ q_0In	00	01	11	10
00	0	1	0	1
01	0	1	1	1
11	1	0	0	0
10	1	1	1	1

Cost-2: 4 inverters + 4*3-input AND gates + 1*2-input AND gate + 1*5-input OR gate

$T_0 = q_2q_0'In' + q_2q_1' + q_1'q_0'In + q_2'q_1In + q_2'q_0In'$

Present State Code			Next State Code											
			In = 0						In = 1					
q_2	q_1	q_0	Q_2	Q_1	Q_0	T_2	T_1	T_0	Q_2	Q_1	Q_0	T_2	T_1	T_0
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1	0	0	1	0	0	0
0	1	0	0	0	0	0	1	0	0	1	1	0	0	1
0	1	1	0	1	0	0	0	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1	0	0	1	1	0	1
1	0	1	0	0	0	1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1	1	0	0	0	1	0
1	1	1	1	1	1	0	0	0	1	1	1	0	0	0

Door Lock with JK FF

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}$(t)

JK Flip-Flop Excitation Table			
q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Present State Code			Next State Code															Out
			In = 0								In = 1							
q_2	q_1	q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	U
0	0	0	0	0	0	0	X	0	X	0	X	0	0	1	0	X	0	X	1	X	0
0	0	1	0	1	0	0	X	1	X	X	1	0	0	1	0	X	0	X	X	0	0
0	1	0	0	0	0	0	X	X	1	0	X	0	1	1	0	X	X	0	1	X	0
0	1	1	0	1	0	0	X	X	0	X	1	1	0	0	1	X	X	1	X	1	0
1	0	0	1	0	1	X	0	0	X	1	X	0	0	1	X	1	0	X	1	X	0
1	0	1	0	0	0	X	1	0	X	X	1	1	1	0	X	0	1	X	X	1	0
1	1	0	1	1	1	X	0	X	0	1	X	1	0	0	X	0	X	1	0	X	0
1	1	1	1	1	1	X	0	X	0	X	0	1	1	1	X	0	X	0	X	0	1

Door Lock with JK FF

q_2q_1 \ q_0In	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	x	x	x	x
10	x	x	x	x

$J_2 = q_1q_0In$

q_2q_1 \ q_0In	00	01	11	10
00	x	x	x	x
01	x	x	x	x
11	0	0	0	0
10	0	1	0	1

$K_2 = q_1'q_0'In + q_1'q_0In'$

Present State Code			Next State Code															Out
			In = 0								In = 1							
q_2	q_1	q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	U
0	0	0	0	0	0	0	X	0	X	0	X	0	0	1	0	X	0	X	1	X	0
0	0	1	0	1	0	0	X	1	X	X	1	0	0	1	0	X	0	X	X	0	0
0	1	0	0	0	0	0	X	X	1	0	X	0	1	1	0	X	X	0	1	X	0
0	1	1	0	1	0	0	X	X	0	X	1	1	0	0	1	X	X	1	X	1	0
1	0	0	1	0	1	X	0	0	X	1	X	0	0	1	X	1	0	X	1	X	0
1	0	1	0	0	0	X	1	0	X	X	1	1	1	0	X	0	1	X	X	1	0
1	1	0	1	1	1	X	0	X	0	1	X	1	0	0	X	0	X	1	0	X	0
1	1	1	1	1	1	X	0	X	0	X	0	1	1	1	X	0	X	0	X	0	1

Door Lock with JK FF

q_2q_1 \ q_0In	00	01	11	10
00	0	0	0	1
01	x	x	x	x
11	x	x	x	x
10	0	0	1	0

$J_1 = q_2q_0In + q_2'q_0In'$

q_2q_1 \ q_0In	00	01	11	10
00	x	x	x	x
01	1	0	1	0
11	0	1	0	0
10	x	x	x	x

$K_1 = q_2'q_0'In' + q_2q_0'In + q_2'q_0In$

Present State Code			Next State Code															Out
			In = 0								In = 1							
q_2	q_1	q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	U
0	0	0	0	0	0	0	X	0	X	0	X	0	0	1	0	X	0	X	1	X	0
0	0	1	0	1	0	0	X	1	X	X	1	0	0	1	0	X	0	X	X	0	0
0	1	0	0	0	0	0	X	X	1	0	X	0	1	1	0	X	X	0	1	X	0
0	1	1	0	1	0	0	X	X	0	X	1	1	0	0	1	X	X	1	X	1	0
1	0	0	1	0	1	X	0	0	X	1	X	0	0	1	X	1	0	X	1	X	0
1	0	1	0	0	0	X	1	0	X	X	1	1	1	0	X	0	1	X	X	1	0
1	1	0	1	1	1	X	0	X	0	1	X	1	0	0	X	0	X	1	0	X	0
1	1	1	1	1	1	X	0	X	0	X	0	1	1	1	X	0	X	0	X	0	1

Door Lock with JK FF

q_2q_1 \ q_0In	00	01	11	10
00	0	1	x	x
01	0	1	x	x
11	1	0	x	x
10	1	1	x	x

$J_0 = q_2q_1' + q_2In' + q_2'In$

q_2q_1 \ q_0In	00	01	11	10
00	x	x	0	1
01	x	x	1	1
11	x	x	0	0
10	x	x	1	1

$K_0 = q_2'q_1 + q_1'In' + q_2q_1'$

Present State Code			Next State Code															Out
			In = 0								In = 1							
q_2	q_1	q_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	U
0	0	0	0	0	0	0	X	0	X	0	X	0	0	1	0	X	0	X	1	X	0
0	0	1	0	1	0	0	X	1	X	X	1	0	0	1	0	X	0	X	X	0	0
0	1	0	0	0	0	0	X	X	1	0	X	0	1	1	0	X	X	0	1	X	0
0	1	1	0	1	0	0	X	X	0	X	1	1	0	0	1	X	X	1	X	1	0
1	0	0	1	0	1	X	0	0	X	1	X	0	0	1	X	1	0	X	1	X	0
1	0	1	0	0	0	X	1	0	X	X	1	1	1	0	X	0	1	X	X	1	0
1	1	0	1	1	1	X	0	X	0	1	X	1	0	0	X	0	X	1	0	X	0
1	1	1	1	1	1	X	0	X	0	X	0	1	1	1	X	0	X	0	X	0	1

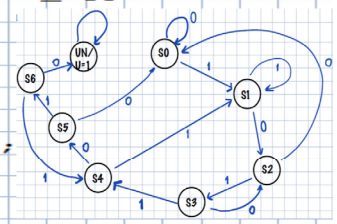
Homework

- Compare the D, T, and JK flip-flop based implementations of the digital door lock for cost. Identify the lowest cost solution.

Digital Door Lock VHDL

```
entity lock is
    port(I,CLK: in std_logic;
         U: out std_logic);
end lock;

architecture lock_arch of lock is
    type state_type is (S0,S1,S2,S3,S4,S5,S6,UN);
    signal next_state, state: state_type;
begin
    process(CLK) begin                          Process
        if rising_edge(CLK) then                  1
            state <= next_state;
        end if;
    end process;
    U<='1' when state=UN else '0';
    process(state,I) begin
        case state is
            when S0=>
                if I='0' then next_state<=S0;
                else next_state<=S1; end if;
            when S1=>                           Process
                if I='0' then next_state<=S2;     2
                else next_state<=S1; end if;
            ...
            ...
            ...
            when others=>
                next_state<=UN;
    end process;
end lock_arch;
```



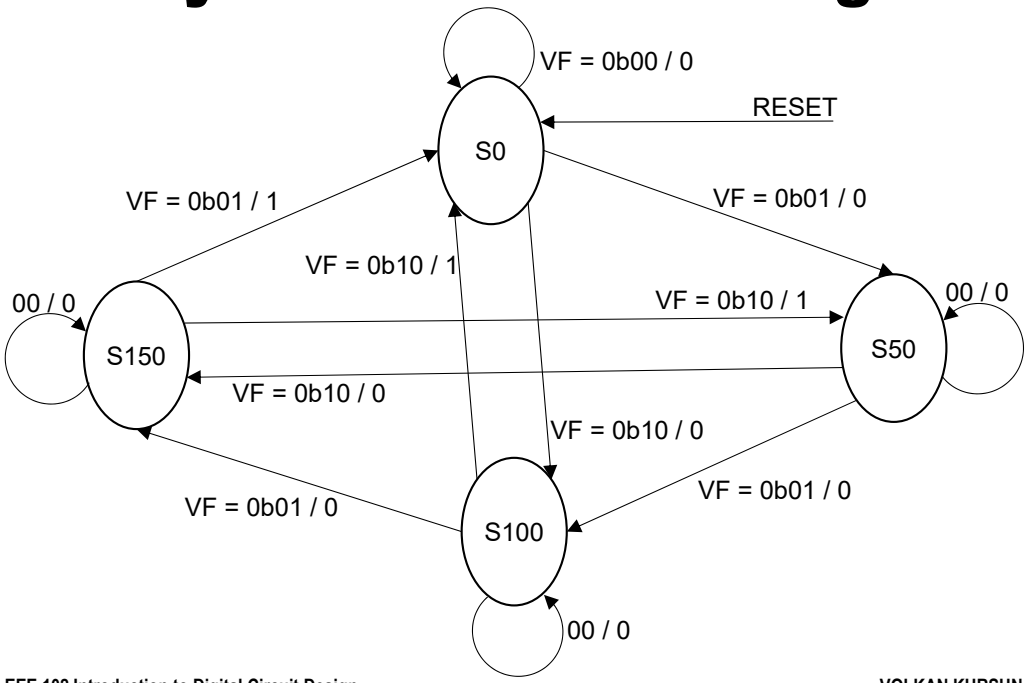
Outline

- Serial Adder
- Digital Door Lock
- Vending Machine
- State Minimization

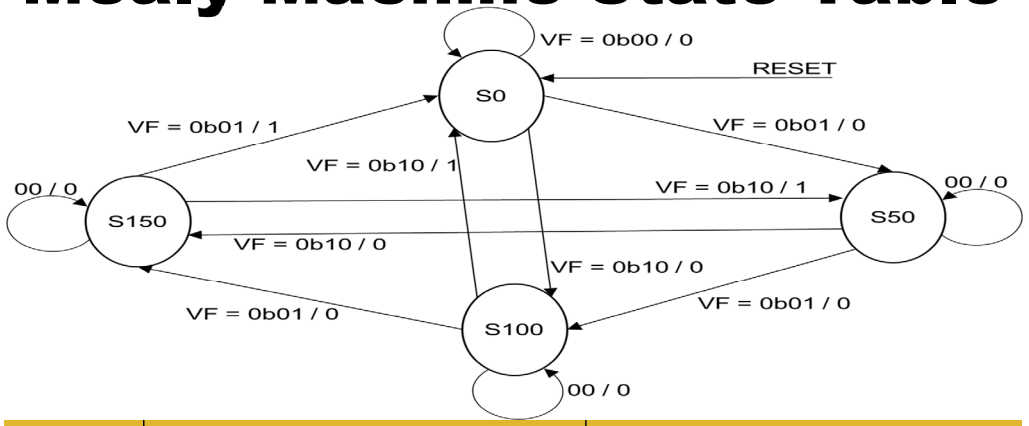
Mealy Coffee Vending Machine

- Mealy machine output = function (present_state, inputs)
- Design the controller for a coffee vending machine that accepts only 1€ and 0.5€ coins
- The mechanical coin slot produces a two bit digital signal (VF) that indicates whether a valid 1€ or 0.5€ coin has been inserted at any given time
- Let the V output of the coin sensor represent 1€ and the F output of the coin sensor represent 0.5€. At any given time only one of V (1€) or F (0.5€) outputs can be asserted by the coin sensor. Coffee price = 2€
- Output Release (R): Asserted if the total coin deposit reaches 2€. Otherwise, R is maintained deasserted
- The machine does not return change
- There is no expiration (time out) for the inserted coin credit

Mealy Machine State Diagram



Mealy Machine State Table



Present State	Next State				Output (R)			
	VF = 00	VF = 01	VF = 11	VF = 10	VF = 00	VF = 01	VF = 11	VF = 10
S0	S0	S50	XX	S100	0	0	X	0
S50	S50	S100	XX	S150	0	0	X	0
S100	S100	S150	XX	S0	0	0	X	1
S150	S150	S0	XX	S50	0	1	X	1

Mealy State Assigned Table

Present State	Present state code	Next State Code				Output			
		VF = 00	VF = 01	VF = 11	VF = 10	VF = 00	VF = 01	VF = 11	VF = 10
	q_1q_0	D_1D_0	D_1D_0	D_1D_0	D_1D_0	R			
S0	00	00	01	XX	11	0	0	X	0
S50	01	01	11	XX	10	0	0	X	0
S100	11	11	10	XX	00	0	0	X	1
S150	10	10	00	XX	01	0	1	X	1

- Derive the next state logic circuits

q_1q_0 \ VF	00	01	11	10
00	0	0	X	1
01	0	1	X	1
11	1	1	X	0
10	1	0	X	0

$D_1 = q_1V'F' + q_0F + q_1'V$

q_1q_0 \ VF	00	01	11	10
00	0	1	X	1
01	1	1	X	0
11	1	0	X	0
10	0	0	X	1

$D_0 = q_0V'F' + q_1'F + q_0'V$

Mealy State Assigned Table

Present State	Present state code	Next State Code				Output			
		VF = 00	VF = 01	VF = 11	VF = 10	VF = 00	VF = 01	VF = 11	VF = 10
	q_1q_0	D_1D_0	D_1D_0	D_1D_0	D_1D_0	R			
S0	00	00	01	XX	11	0	0	X	0
S50	01	01	11	XX	10	0	0	X	0
S100	11	11	10	XX	00	0	0	X	1
S150	10	10	00	XX	01	0	1	X	1

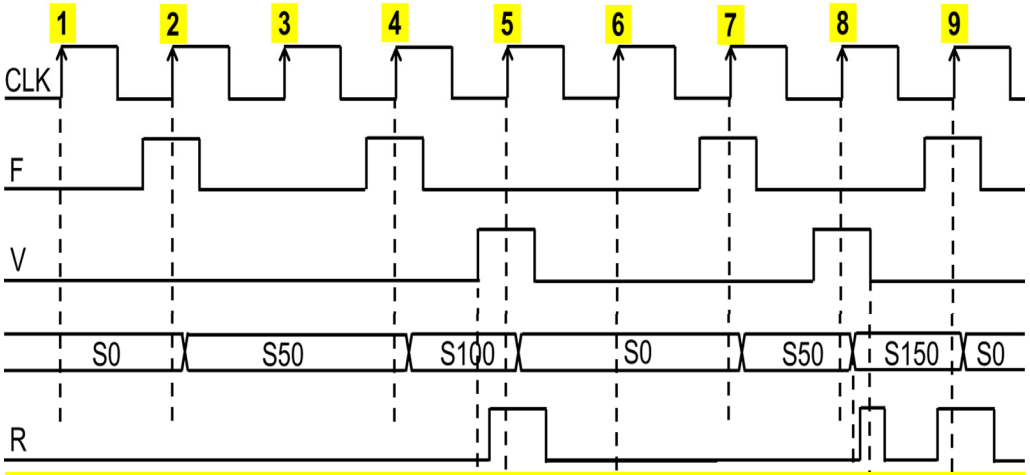
- Derive the output logic circuit

q_1q_0 \ VF	00	01	11	10
00	0	0	X	0
01	0	0	X	0
11	0	0	X	1
10	0	1	X	1

$R = q_1q_0'F + q_1V$

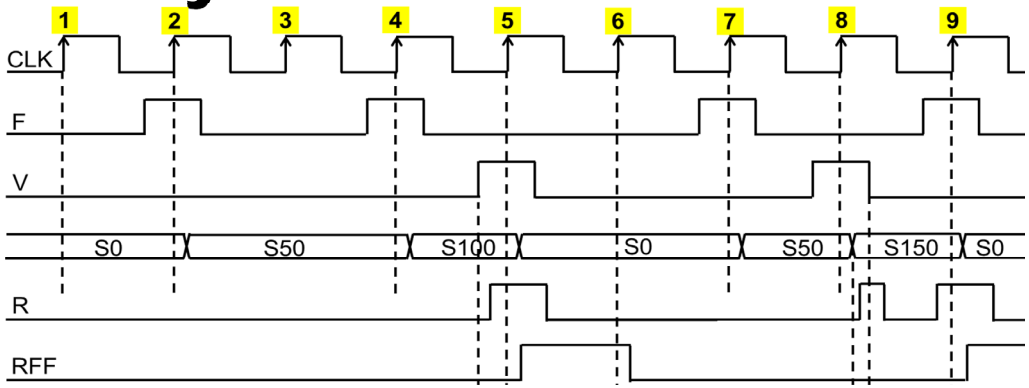
The digital signal that controls the coffee release mechanical system

Mealy Vending Machine Glitches

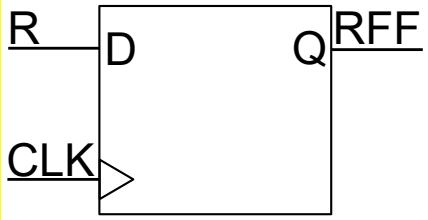


When the machine transitions to state S150 at the beginning of cycle 8, V input is still high, thereby causing a glitch on the R output and causing the vending machine to **erroneously release a cup of coffee**

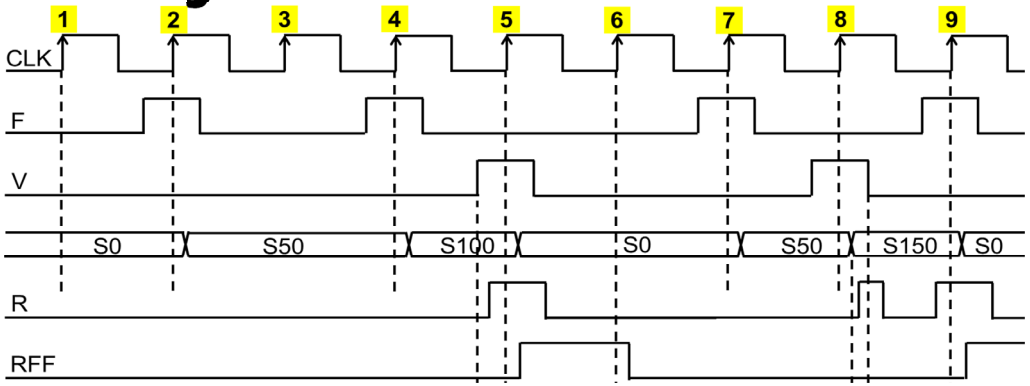
Mealy Glitch Removal Circuit



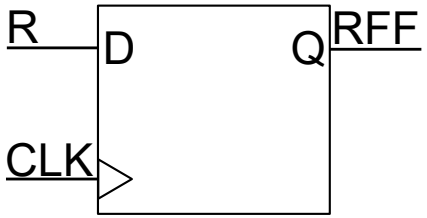
Solution: add a flip-flop to store the release (R) control signals with the positive edges of the clock signal. This way the glitches that happen after the positive clock edges are ignored.



Mealy Glitch Removal Circuit



Overhead: The glitch removal circuitry causes the latency and number of flip-flops to increase



Homework

- Design a Moore type coffee vending machine

Outline

- Serial Adder
- Digital Door Lock
- Vending Machine
- State Minimization

State Minimization

- An initial design may have more states than the actual requirement
- **Minimize the number of states** to **reduce the number of flip-flops** that represent the states and lower **the complexity of the combinational circuit**
- If the **number of states** in an FSM can be **reduced**, then **some states** in the original design must be **equivalent**
- Two states are **equivalent if and only if** for **every possible input sequence**, **the same output and next state sequence is produced**

State Minimization Example

E and G are equivalent: remove the present state G row and replace all G with E in the rest of the Table

Present State	Next State		Output	
	X = 0	X = 1	X = 0	X = 1
A	A	B	0	0
B	C	D	0	0
C	A	D	0	0
D	E	F	0	1
E	A	F	0	1
F	G	F	0	1
G	A	F	0	1

Present State	Next State		Output	
	X = 0	X = 1	X = 0	X = 1
A	A	B	0	0
B	C	D	0	0
C	A	D	0	0
D	E	F	0	1
E	A	F	0	1
F	~~G~~ E	F	0	1

D and F are equivalent: remove the present state F row and replace all F with D in the rest of the Table

Present State	Next State		Output	
	X = 0	X = 1	X = 0	X = 1
A	A	B	0	0
B	C	D	0	0
C	A	D	0	0
D	E	~~F~~ D	0	1
E	A	~~F~~ D	0	1

State Minimization with Partitioning

- Instead of trying to show that some states are equivalent, an alternative approach is **to show that some states are definitely NOT equivalent**
- Let k represent the set of all possible input combinations in a state machine. **If states S_i and S_j are equivalent, then their k-successors (for all k) are also equivalent.**
- **A partition consists of** one or more **blocks**, where each block comprises a set of states that **may be** equivalent, but the **states in a given block are definitely NOT equivalent to the states in other blocks**

FSM Partitioning Procedure

- **Step 0**: initially **assume all states are equivalent**. **All states are in one block P0 in the initial step**.
- **Step 1**: form the first partition P1 in which the set of states is **partitioned into blocks** such that the **states in each block generate the same output values**. The **states that generate different outputs can NOT be equivalent and must be placed in different blocks**.
- **Step 2**: continue to form new partitions by **testing whether the k-successors of the states in each block are contained in one block**. Those **states whose k-successors are in different blocks can NOT be equivalent and must be partitioned into new blocks**.
- **Step final**: The process ends when further partitioning is not possible. Then all states in one block are equivalent and the FSM can be simplified by eliminating the equivalent states.

FSM Partitioning Example

- Consider the following state table of an FSM with 7 states.

Present state	Next state $w=0$	Next state $w=1$	Output z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

- **Step 0 (initial step)**: contains all states in one block P0

P0 = (ABCDEFG)

- **Step 1**: separate the states that have different outputs. States A, B, and D must be different from states C, E, F, and G since they have different outputs. The first partition with two blocks is P1 = (ABD)(CEFG)

FSM Partitioning Example Step 2

Present state	Next state $w=0$	Next state $w=1$	Output z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

- P1 = (ABD)(CEFG)
- **Step 2**: Consider all 0 and 1 successors of the states in each block. The **0-successors** of block (ABD) are (**BDB**) and they are all in the same block. The **1-successors** of block (ABD) are (**CFG**) and they are also all in the same block. **A, B, and D therefore may be equivalent and will remain in the same block in the next partition P2**.

FSM Partitioning Example Step 2

Present state	Next state $w=0$	Next state $w=1$	Output z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

- P1 = (ABD)(CEFG)
- **Step 2 continues**: Consider all 0 and 1 successors of the states in (CEFG) block. The **0-successors** of block (CEFG) are (**FFEF**) and they are all in the same block. The **1-successors** of block (CEFG) are (**ECDG**) and they are **NOT** all in the same block in P1. **F must be different** from the states C, E, and G since its 1-successor is D which is in a different block from C, E, and G's 1-successors.

FSM Partitioning Example Step 3

Present state	Next state $w=0$	Next state $w=1$	Output z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

- Separate F in partition 2: P2 = (ABD)(CEG)(F)
- **Step 3**: Consider all 0 and 1 successors of the states in each block. The **0-successors** of block (CEG) are (**FFF**) and they are all in the same block. The **1-successors** of block (CEG) are (**ECG**) and they are all in the same block in P2. **C, E, and G therefore may be equivalent and will remain in the same block in the next partition P3**.

FSM Partitioning Example Step 3

Present state	Next state $w=0$	Next state $w=1$	Output z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

- Separate F in partition 2: P2 = (ABD)(CEG)(F)
- **Step 3 continues**: Consider all 0 and 1 successors of the states in the (ABD) block. The **0-successors** of block (ABD) are (**BDB**) and they are all in the same block. The **1-successors** of block (A**B**D) are (C**F**G) and they are **NOT** all in the same block in P3. **B must be different** from the states A and D since its 1-successor is F which is in a different block from A and D's 1-successors.

FSM Partitioning Example Step 4

Present state	Next state $w=0$	Next state $w=1$	Output z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

- Separate B in partition 3: P3 = (AD)(B)(CEG)(F)
- **Step 4**: Consider all 0 and 1 successors of the states in each block. The **0-successors** of block (AD) are (**BB**) and they are all in the same block. The **1-successors** of block (AD) are (**CG**) and they are all in the same block in P3. **A and D therefore may be equivalent and will remain in the same block in the next partition P4.**

FSM Partitioning Example Step 4

Present state	Next state $w=0$	Next state $w=1$	Output z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

- Separate B in partition 3: P3 = (AD)(B)(CEG)(F)
- **Step 4**: Consider all 0 and 1 successors of the states in the (CEG) block. The **0-successors** of block (CEG) are (**FFF**) and they are all in the same block. The **1-successors** of block (CEG) are (**ECG**) and they are all in the same block in P3. **C, E, and G therefore may be equivalent and will remain in the same block in the next partition P4**.

FSM Final Partition: P4

Present state	Next state $w=0$	Next state $w=1$	Output z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

- Partition 4: P4 = **(AD)**(B)**(CEG)**(F) = P3
- **No new blocks were generated for P4**. **Since P4 = P3, the states in each block in P4 are now guaranteed to be equivalent.**
- **Block (AD)**: states A and D are equivalent
- **Block (CEG)**: states C, E, and G are equivalent
- **Each block can be represented by a single state in the final partition**

Minimized State Table

Present state	Nextstate w = 0	Nextstate w = 1	Output z
A	B	C	1
B	~~D~~ A	F	1
C	F	~~E~~ C	0
F	~~E~~ C	~~D~~ A	0

- Partition 4: P4 = (AD)(B)(CEG)(F) = P3
- **Each block can be represented by a single state in the final partition**
- **Block (AD)**: states A and D are equivalent. Let A represent both states A and D in the minimized state table
- **Block (CEG)**: states C, E, and G are equivalent. Let state C represent states C, E, and G in the minimized state table.
- **Number of states reduced from 7 to 4: only 2 flip-flops are needed instead of the original design with 3 flip-flops**