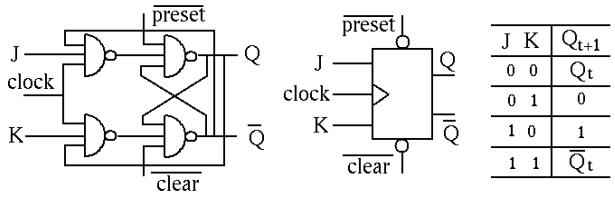
EEE 102
INTRODUCTION TO DIGITAL CIRCUIT DESIGN
RECITATION PROBLEM SET 5

ARASH ASHRAFNEJAD

1. JK-flipflop Review

Problem: Verify the behavior of JK-flipflop discussed in class by filling out the tables below. Make comment on each case in terms of whether there is a state change, if so, whether it is a set (Q=1) or reset (Q=0).

Clock	J	K	Q_t	$\overline{Q}_t$	$\overline{set}$	$\overline{reset}$	Q_{t+1}	$\overline{Q}_{t+1}$	comment
0	x	x	x	x					
1	0	0	x	x					
1	0	1	0	1					
1	0	1	1	0					
1	1	0	0	1					
1	1	0	1	0					
1	1	1	0	1					
1	1	1	1	0					



Here $\overline{set}$ and $\overline{reset}$ are the outputs of the two NAND gates on the left with input J and K, respectively.

Solution:

Clock	J	K	Q_t	$\overline{Q}_t$	$\overline{set}$	$\overline{reset}$	Q_{t+1}	$\overline{Q}_{t+1}$	
0	x	x	x	x	1	1	Q	$\overline{Q}$	(no change)
1	0	0	x	x	1	1	Q	$\overline{Q}$	(no change)
1	0	1	0	1	1	1	0	1	(no change)
1	0	1	1	0	1	0	0	1	(reset)
1	1	0	0	1	0	1	1	0	(set)
1	1	0	1	0	1	1	1	0	(no change)
1	1	1	0	1	0	1	1	0	(set)
1	1	1	1	0	1	0	0	1	(reset)

□

2. Convert Flip-Flops

Problem:

(1) Convert a given D-flipflop to a JK-flipflop with minimal additional logic gates.
(2) Convert a T Flip-Flop to a JK Flip-Flop.
(3) Convert a RS Flip-Flop to a T Flip-Flop.
(4) Build a JK-flipflop using a D-flipflop (of master-slave type) and a 4x1 MUX. No additional gates are allowed.

Solution:

(1)

$$D = Q'J + K'Q$$

(2)

$$T = Q'J + QK$$

Q_t	Q_{t+1}	J	K	T
0	0	0	x	0
0	1	1	x	1
1	0	x	1	1
1	1	x	0	0

(3)

$S = Q'T, \qquad R = QT$

Q_t	Q_{t+1}	T	S	R
0	0	0	0	x
0	1	1	1	0
1	0	1	0	1
1	1	0	x	0

(4) use Q, 0, 1 and Q' as the 4 inputs and J,K as the selections of MUX.

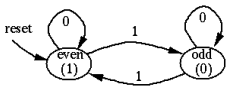
□

3. Even Parity Checker

Problem: An *even parity checker* is a sequential logic circuit that counts the number of 1's in a bit serial input stream and outputs 1 whenever this number is even (including 0). For example, if the input stream is 0110100101...., the corresponding output stream is 1011000110.... Implement such an even parity checker as a finite state machine using (1) RS-FFs, (2) JK-FF's, (3) D-FFs, and (4) T-FF's. Draw the logic diagram for each of the implementations. Which one requires minimum combinational logic?

Solution:

The state diagram:

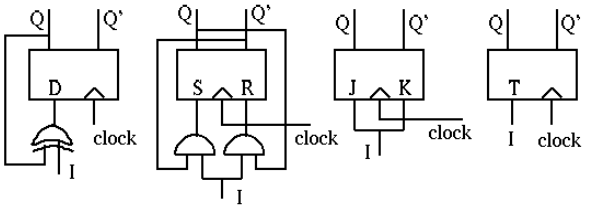


The state table:

Present	Input	Next						
State (PS)	I	State (NS)	S	R	J	K	D	T
even 1	0	even 1	0	x	0	x	0	0
even 1	1	odd 0	1	0	1	x	1	1
odd 0	0	odd 0	x	0	x	0	1	0
odd 0	1	even 1	0	1	x	1	0	1

The next state decoder:

- **D-FF:** $D = PS'\, I + PS\, I' = PS \oplus I$
- **RS-FF:** $S = PS'\, I, \quad R = PS\, I$
- **JK-FF:** $J = K = I$
- **T-FF:** $T = I$



□

4. FSM Combinational Lock

Problem: Design a digital combination lock which will set the output signal "unlock"to 1 only after receiving a key, a specific serial binary pattern 101. (Of course for the lock to be practically useful, the binary pattern need to be much longer).

Hint: This lock can be designed as a finite state machines with one input (which receives a bit in the binary sequence at a time) and one output (which is 1 only if the pattern 101 is detected). A sample input sequence and the corresponding output sequence of this FSM are shown below.

Input		0	1	0	0	1	0	1	0	0	1	1	0	1	1
Output		0	0	0	0	0	0	1	0	0	0	0	0	1	0
State	A								A						A

This FSM has the following 4 states:

- state A (00): the initial state of the FSM and it should always be in this state when more than one 0 in a sequence is received (impossible to be part of the key).
- state B (01): the FSM should be in this state whenever a 1 is detected as it could be the first 1 of the key.
- state C (10): when the partial pattern 10 is detected;
- state D (11): when the complete pattern 101 is detected, then the FSM is reset so that the next state is the initial state A (independent of the input).

Design this single-input, single-output, 4-state FSM using D-FF's in the following steps:

(1) Fill out the 3nd row of the table above to show the state transition corresponding to the current input;
(2) Draw the state diagram;
(3) Complete the state transition table below;
(4) Implement the next state decoder and the output decoder. (Hint: this could be designed as a Moore machine and no output decoder is needed.)

The state transition table:

	Present State Q_1	Q_0	Input	Next State Q_1	Q_0	Output
A	0	0	0			
	0	0	1			
B	0	1	0			
	0	1	1			
C	1	0	0			
	1	0	1			
D	1	1	0			
	1	1	1			

Solution:

Input		0	1	0	0	1	0	1	0	0	1	1	0	1	1
Output		0	0	0	0	0	0	1	0	0	0	0	0	1	0
State (key)	A	A	B	C	A	B	C	D	A	A	B	B	C	D	A

$D_1 = Q_1'Q_0I' + Q_1Q_0'I$, $D_0 = I(Q_1' + Q_0')$, $Output1 = Q_1Q_0'I$, $Output2 = Q_1Q_0$,

	Present State Q_1	Q_0	Input		Next State Q_1	Q_0	Output
A	0	0	0	A	0	0	0
	0	0	1	B	0	1	0
B	0	1	0	C	1	0	0
	0	1	1	B	0	1	0
C	1	0	0	A	0	0	0
	1	0	1	D	1	1	0 (1)
D	1	1	0	A	0	0	1 (0)
	1	1	1	A	0	0	1 (0)

□

5. Modulo 4 Up-down Counter

Problem: Design a modulo 4 non-ripple up-down counter as a finite state machine. The counter counts down when input $I = 0$, and it counts up when $I = 1$. Use four T-type flipflops to encode the four states of the FSM ("one-hot" encoding).

(1) Fill out the state table below for the next state and the signals needed to tigger the T-FFs.

	Present state				Input	Next state				signals needed			
	Q_3	Q_2	Q_1	Q_0	I	Q_3'	Q_2'	Q_1'	Q_0'	T_3	T_2	T_1	T_0
0	0	0	0	1	0								
0	0	0	0	1	1								
1	0	0	1	0	0								
1	0	0	1	0	1								
2	0	1	0	0	0								
2	0	1	0	0	1								
3	1	0	0	0	0								
3	1	0	0	0	1								

(2) Give the logic expression for each of the four signals T_i $(i = 0, 1, 2, 3)$

(3) Draw the complete diagram of the FSM in terms of the four FFs and additional logic gates needed for the next state decoder.

Solution:

	Present state				Input	Next state				signals needed			
	Q_3	Q_2	Q_1	Q_0	I	Q_3'	Q_2'	Q_1'	Q_0'	T_3	T_2	T_1	T_0
0	0	0	0	1	0	1	0	0	0	1	0	0	1
0	0	0	0	1	1	0	0	1	0	0	0	1	1
1	0	0	1	0	0	0	0	0	1	0	0	1	1
1	0	0	1	0	1	0	1	0	0	0	1	1	0
2	0	1	0	0	0	0	0	1	0	0	1	1	0
2	0	1	0	0	1	1	0	0	0	1	1	0	0
3	1	0	0	0	0	0	1	0	0	1	1	0	0
3	1	0	0	0	1	0	0	0	1	1	0	0	1

$T_3 = Q_0 I' + Q_2 I + Q_3$, $T_2 = Q_1 I + Q_2 + Q_3 I'$, $T_1 = Q_0 I + Q_1 + Q_2 I'$, $T_0 = Q_0 + Q_1 I' + Q_3 I$,
□