XMUT 202
Digital Electronics

Felix Yan

School of Engineering and Computer Science
Victoria University of Wellington

Victoria
UNIVERSITY OF WELLINGTON
Te Whare Wānanga
o te Ūpoko o te Ika a Māui

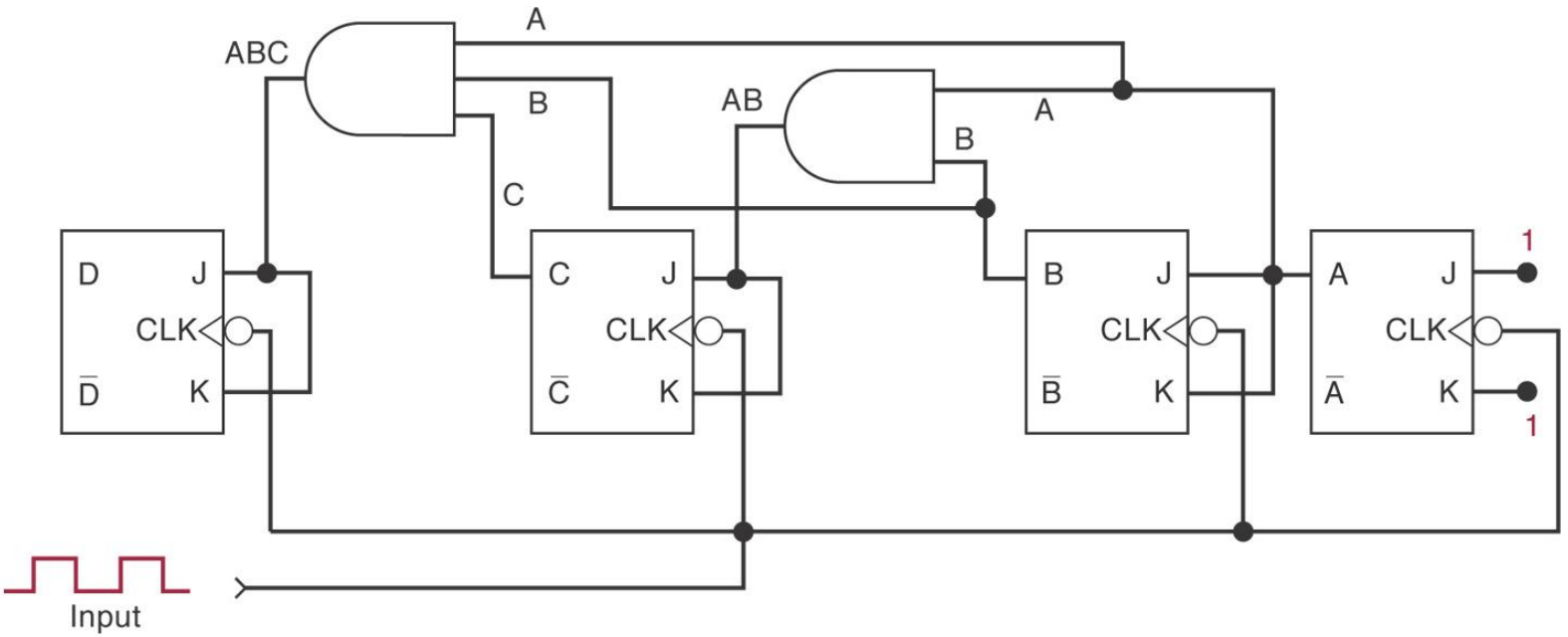
CAPITAL CITY UNIVERSITY

Week 14

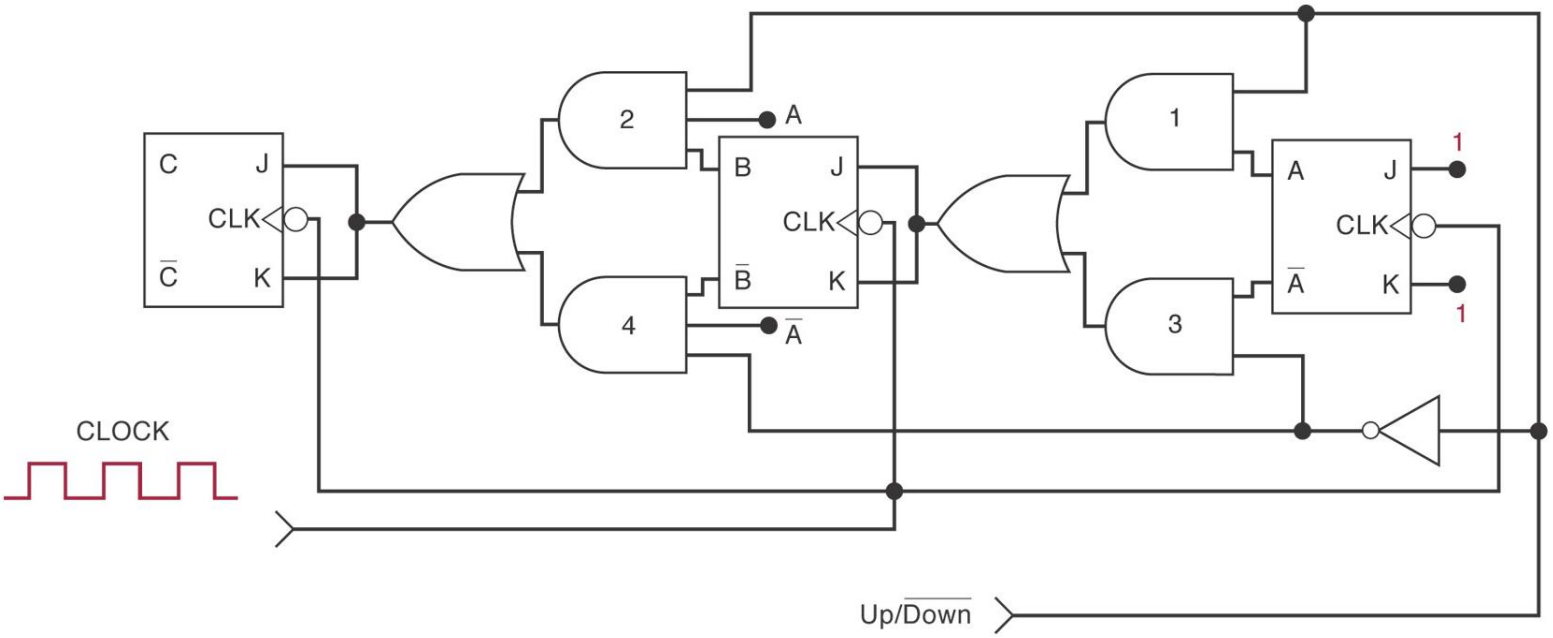
- Synchronous counter design
- Even Odd Counter Design

What kind of circuit counter is this?

What will this circuit do?



And this ?



In the case of the previous two binary counters we find that the counters consist of:

- Sequential element (the flip flops) that provides an output (Q) based on the state of their J, K inputs when the synchronous clock pulse occurs.
- The combination logic elements that ensures the J,K inputs are as desired to achieve the correct transitions in the FF.
- We could design the 4 bit up counter rather intuitively and with a little bit more thinking provide the up-down facility for the 3-bit counter.
- Will now look at a general procedure for designing synchronous counters, taking into account that the states may not be the normal binary order.

Two examples were of simple binary counters, but often real counters have:

- Some arbitrary (non-binary) counting sequence
- Possible undesired (hang-up) states.
- Up/down count ability or special requirements.

Examples:

- Gray code counter

Sequential circuit design will now look at a formal method to design such circuits that contain both memory and combinatorial logic elements

Start by looking at the truth table for J-K flip flops.

J	K	CLK	Q
0	0	↑	Q_0 (no change)
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q_0}$ (toggles)

From the truth table we can construct an excitation table that states the conditions for J and K in order for a certain transition to take place.

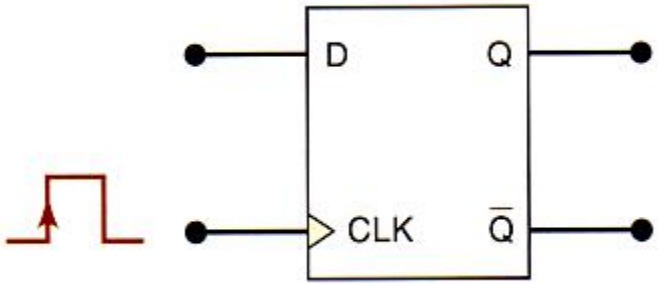
Present State Q(n)	Next State Q(n+1)		J	K
0	0		0	0
		or	0	1
		thus	0	x
0	1		1	0
		or	1	1
		thus	1	x
1	0		0	1
		or	1	1
		thus	x	1
1	1		1	0
		or	0	0
		thus	x	0

In short, the excitation table for J-K flip flops

Present Q	Next Q	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Similarly for D FF's

Truth Table



D	CLK	Q
0	↑	0
1	↑	1

Excitation table

Present State Q(n)	Next State Q(n+1)	Input D
0	0	0
0	1	1
1	0	0
1	1	1

Synchronous Counter Design Method

Example 1:

Design a synchronous up counter using J-K FF's that will count the first eight binary states.

Step 1: Determine the counting sequence and the desired number of bits (FF's) needed.

Counting Sequence:

$000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100 \rightarrow 101 \rightarrow 110 \rightarrow 111 \rightarrow 000$

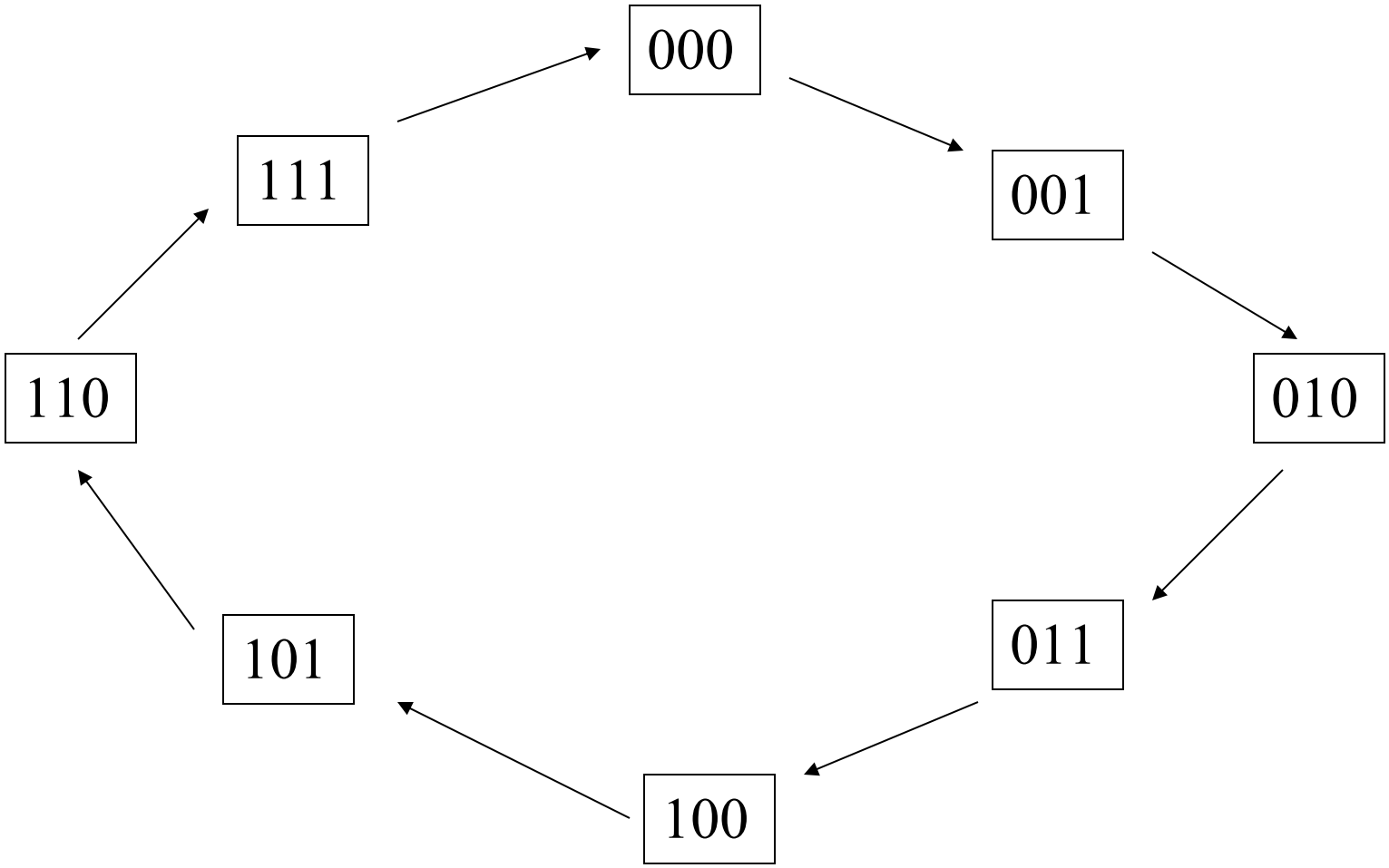
Number of FF's: MOD 8, thus 3 FF's

Note notation used here:

C,B and A are the outputs of the three counter stages (J-K flip flops)

J_a, K_a, J_b, K_b, J_c and K_c are the inputs for each of the J-K flip flops

Step 2: Draw the state transition diagram, showing all states including the undesired states.



Step 3: Use the state transition diagram to draw up a table that lists all PRESENT state and their NEXT states.

Present State Q(n)				Next State Q(n+1)		
C	B	A		C	B	A
0	0	0		0	0	1
0	0	1		0	1	0
0	1	0		0	1	1
0	1	1		1	0	0
1	0	0		1	0	1
1	0	1		1	1	0
1	1	0		1	1	1
1	1	1		0	0	0

Step 4: Add a column to this table for each of the FF input that you plan to use. Fill in these columns using the FF excitation tables such that the desired transition from the PRESENT to the NEXT state is possible.

Present State Q(n)				Next State Q(n+1)			FF States Needed					
C	B	A		C	B	A	Jc	Kc	Jb	Kb	Ja	Ka
0	0	0		0	0	1	0	x	0	x	1	x
0	0	1		0	1	0	0	x	1	x	x	1
0	1	0		0	1	1	0	x	x	0	1	x
0	1	1		1	0	0	1	x	x	1	x	1
1	0	0		1	0	1	x	0	0	x	1	x
1	0	1		1	1	0	x	0	1	x	x	1
1	1	0		1	1	1	x	0	x	0	1	x
1	1	1		0	0	0	x	1	x	1	x	1

Step 5: Design the logic circuits needed to generate the levels required at each J and K input.

Method 1: Use K-maps for simplification and implement the combinational logic circuit for each of the J,K inputs.

For Jc

	/A	A
/C/B	0	0
/C.B	0	1
C.B	x	x
C./B	x	x

Jc=AB

For Kc

	/A	A
/C/B	x	x
/C.B	x	x
C.B	0	1
C./B	0	x

Kc=AB

For Jb

	/A	A
/C/B	0	1
/C.B	x	x
C.B	x	x
C./B	0	1

Jb=A

For Kb

	/A	A
/C/B	x	x
/C.B	0	1
C.B	0	1
C./B	x	x

Kb=A

For Ja

	/A	A
/C/B	1	x
/C.B	1	x
C.B	1	x
C./B	1	x

Ja=1

For Ka

	/A	A
/C/B	x	1
/C.B	x	1
C.B	x	1
C./B	x	1

Ka=1

We thus have logic requirements:

Ja = Ka = 1

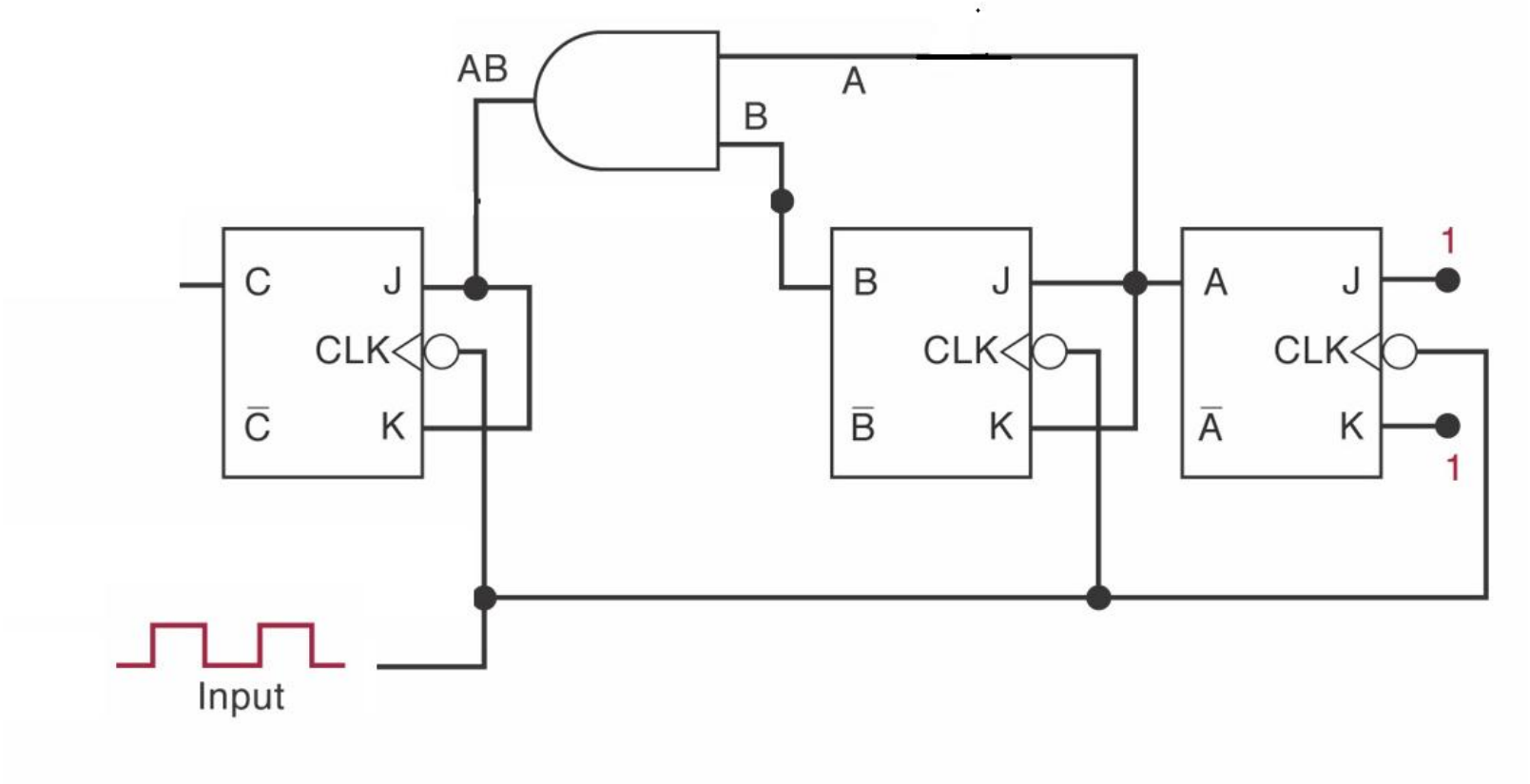
Jb = Kb = A

Jc = Kc = AB

SIMILAR TO OUR RATHER INTUITIVE DESIGN OF EARLIER !

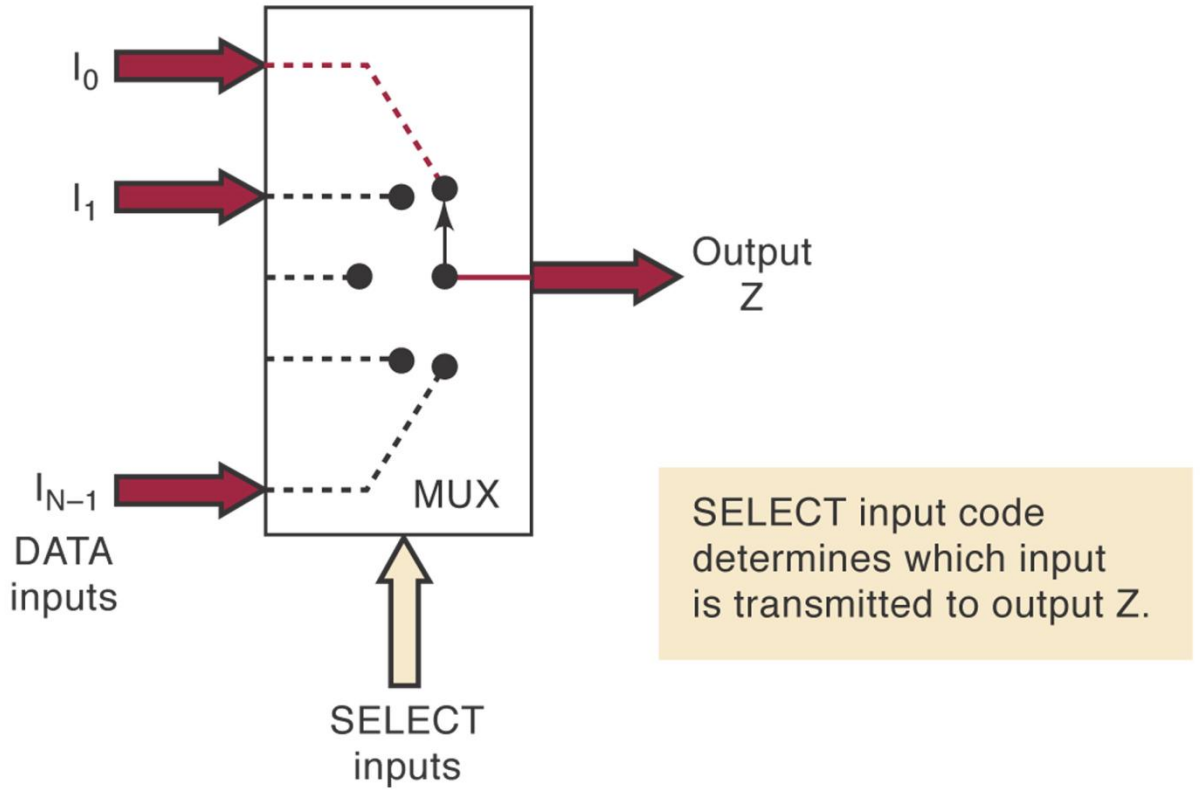
This logic can be implemented either using logic gates or using MUXes

Logic implemented with logic gates for a 3-bit synchronous UP counter – same what we have intuitively done

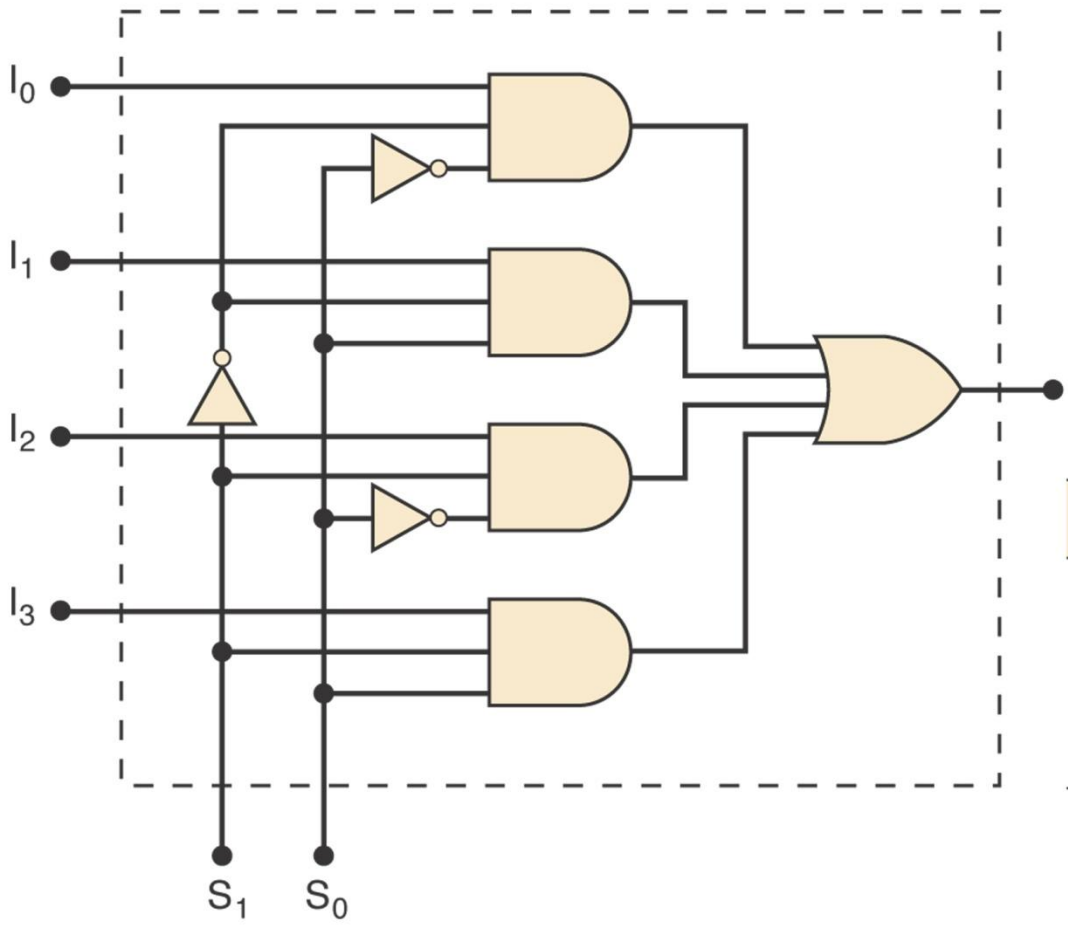


We can also construct our logic circuit using multiplexers

Recall from earlier: A multiplexer (MUX) selects one of multiple input signals and passes it to the output.

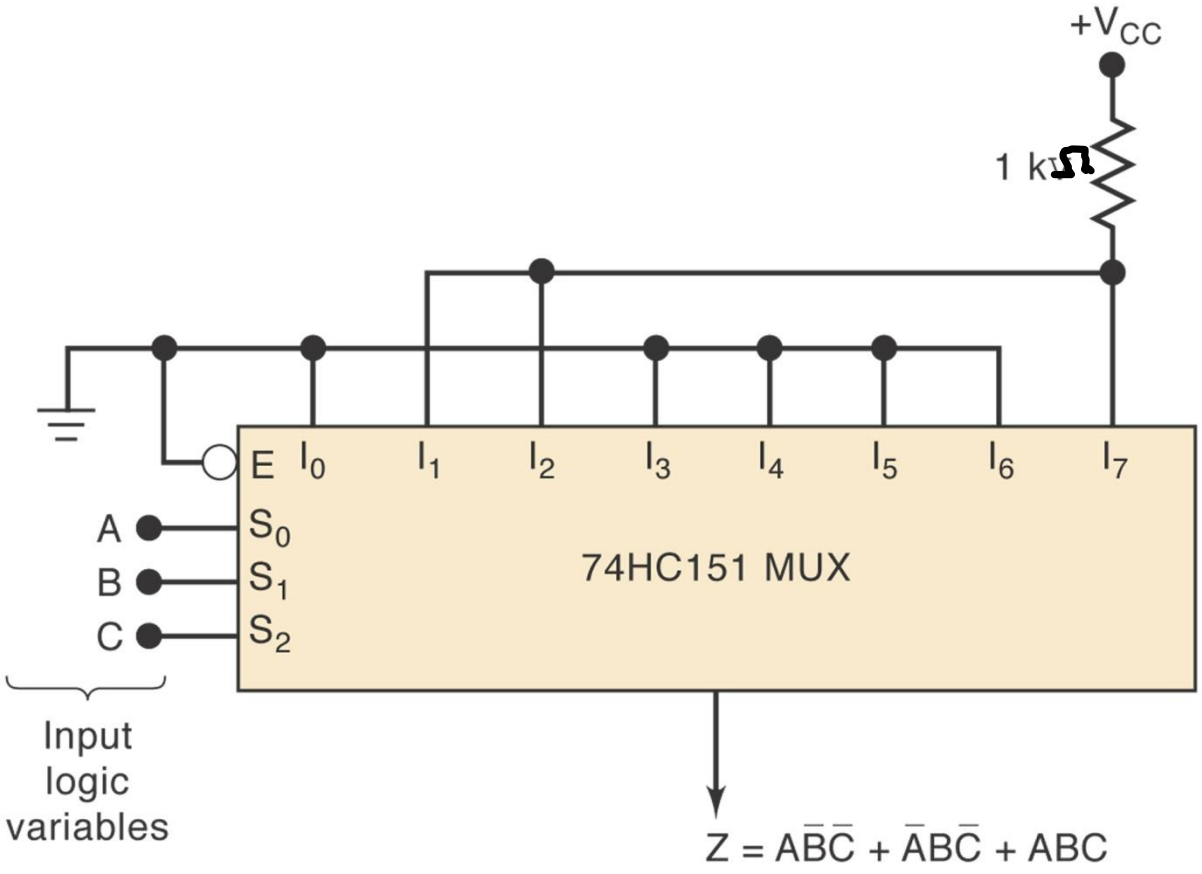


Four-input multiplexer.



S_1	S_0	Output
0	0	$Z = I_0$
0	1	$Z = I_1$
1	0	$Z = I_2$
1	1	$Z = I_3$

Example: Multiplexer used to implement a logic function described by the truth table.

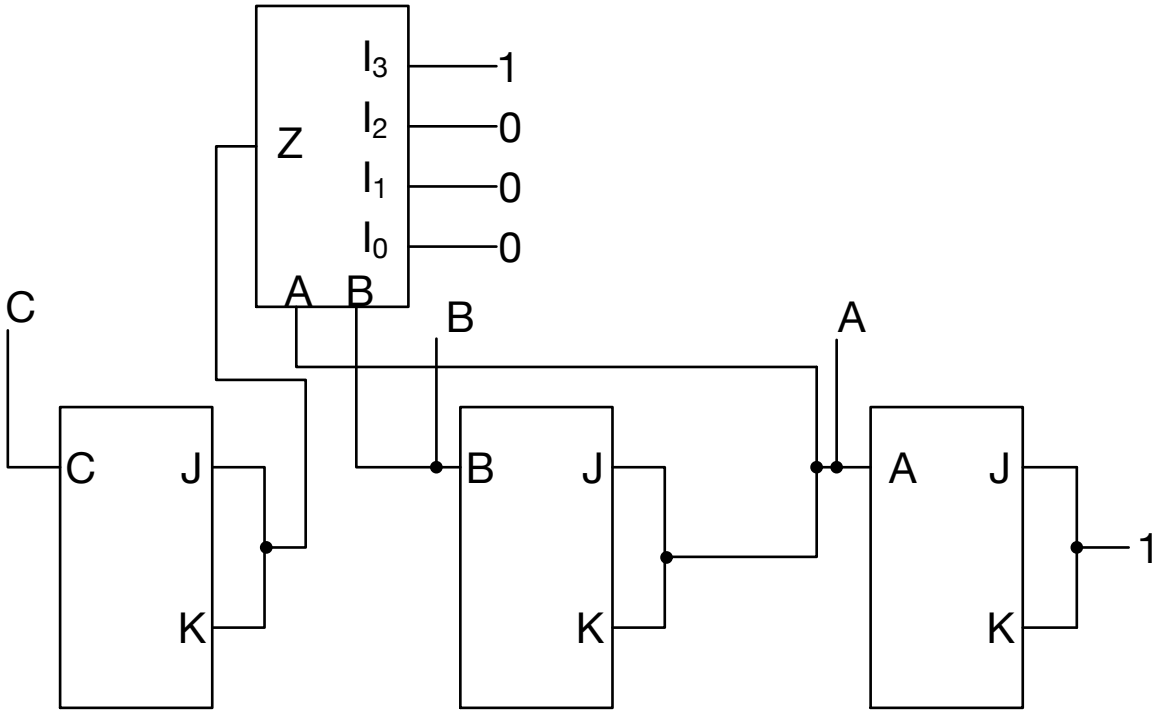


C	B	A	Z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(b)

It would not be very practical to implement the required logic for the 3-bit counter using MUXes, as only the input of the third counter stage would require a MUX (For the first stage J=K=1 and for the second stage J=K=A)

For the third stage J=K=AB, thus a 4 to 1 MUX as below:



Not a good use of a 4:1 MUX in building our 3-bit counter ! An AND gate would have been simpler !

Example 2:

Design a 3–bit synchronous counter going through the states 000, 001, 011, 010, 110, 100, 000. Use D FF's and implement the logic using both logic gates and MUXes.

Step 1: Determine the counting sequence and the desired number of bits (FF's) needed.

Number of FF's:

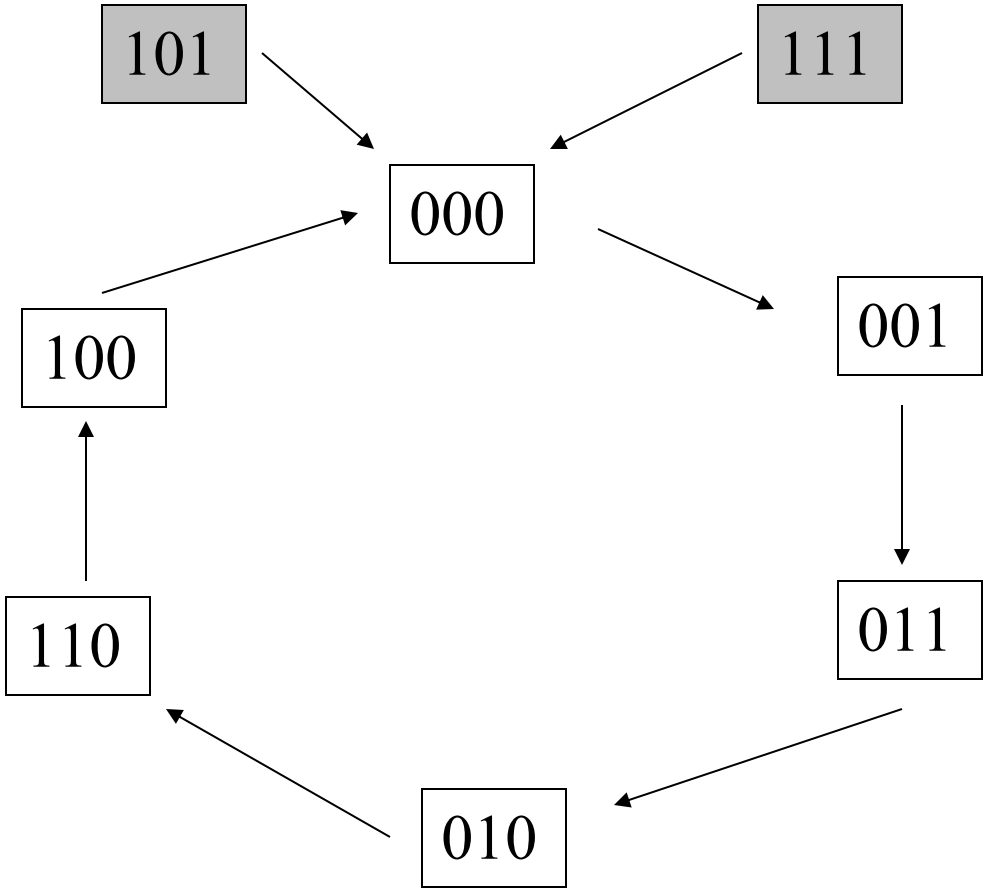
MOD 6, thus 3 FF's, but with two undesired states.

Please note the notation used:

Q_0, Q_1 and Q_2 are the outputs of the three D flip flops

D_0, D_1 and D_2 are the inputs of the three D flip flops

Step 2: Draw the state transition diagram, showing all states including the undesired states.



Steps 3 & 4: Draw excitation table indicating states of FF's needed for transitions.

Present State Q(n)				Next State Q(n+1)			FF States Needed		
Q2	Q1	Q0		Q2	Q1	Q0	D2	D1	D0
0	0	0		0	0	1	0	0	1
0	0	1		0	1	1	0	1	1
0	1	1		0	1	0	0	1	0
0	1	0		1	1	0	1	1	0
1	1	0		1	0	0	1	0	0
1	0	0		0	0	0	0	0	0
1	0	1		0	0	0	0	0	0
1	1	1		0	0	0	0	0	0
Undesired States									

Excitation table easy for D flip-flops: The input simply is whatever we need the next state to be !

Step 5: Design the logic circuits – done here using K maps.

	/Q0	Q0
/Q2/Q1	0	0
/Q2 Q1	1	0
Q2 Q1	1	0
Q2 /Q1	0	0

$D_2 = Q_1\overline{Q_0}$

	/Q0	Q0
/Q2/Q1	0	1
/Q2 Q1	1	1
Q2 Q1	0	0
Q2 /Q1	0	0

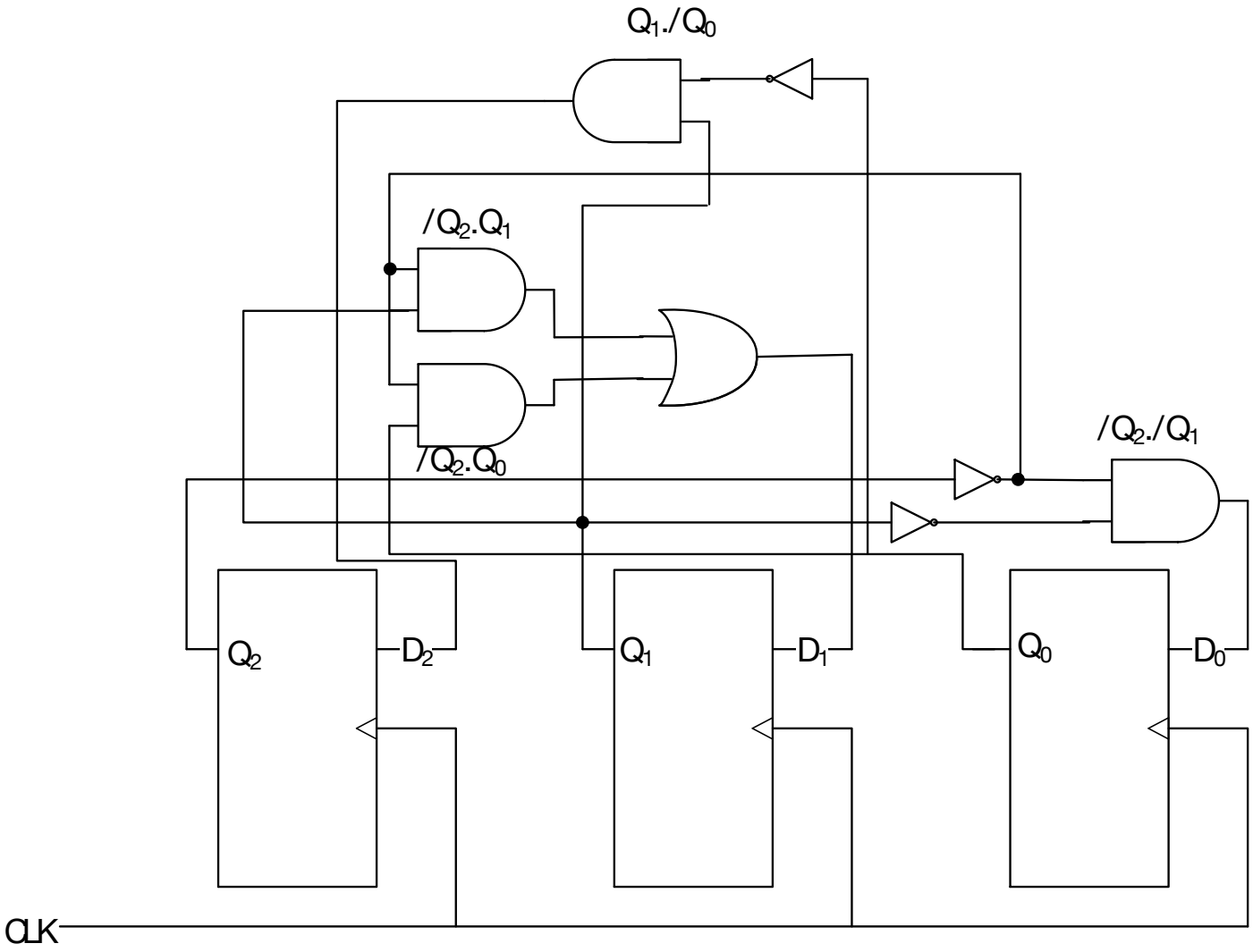
$D1 = \overline{Q_2}Q_1 + \overline{Q_2}Q_0$

	/Q0	Q0
/Q2/Q1	1	1
/Q2 Q1	0	0
Q2 Q1	0	0
Q2 /Q1	0	0

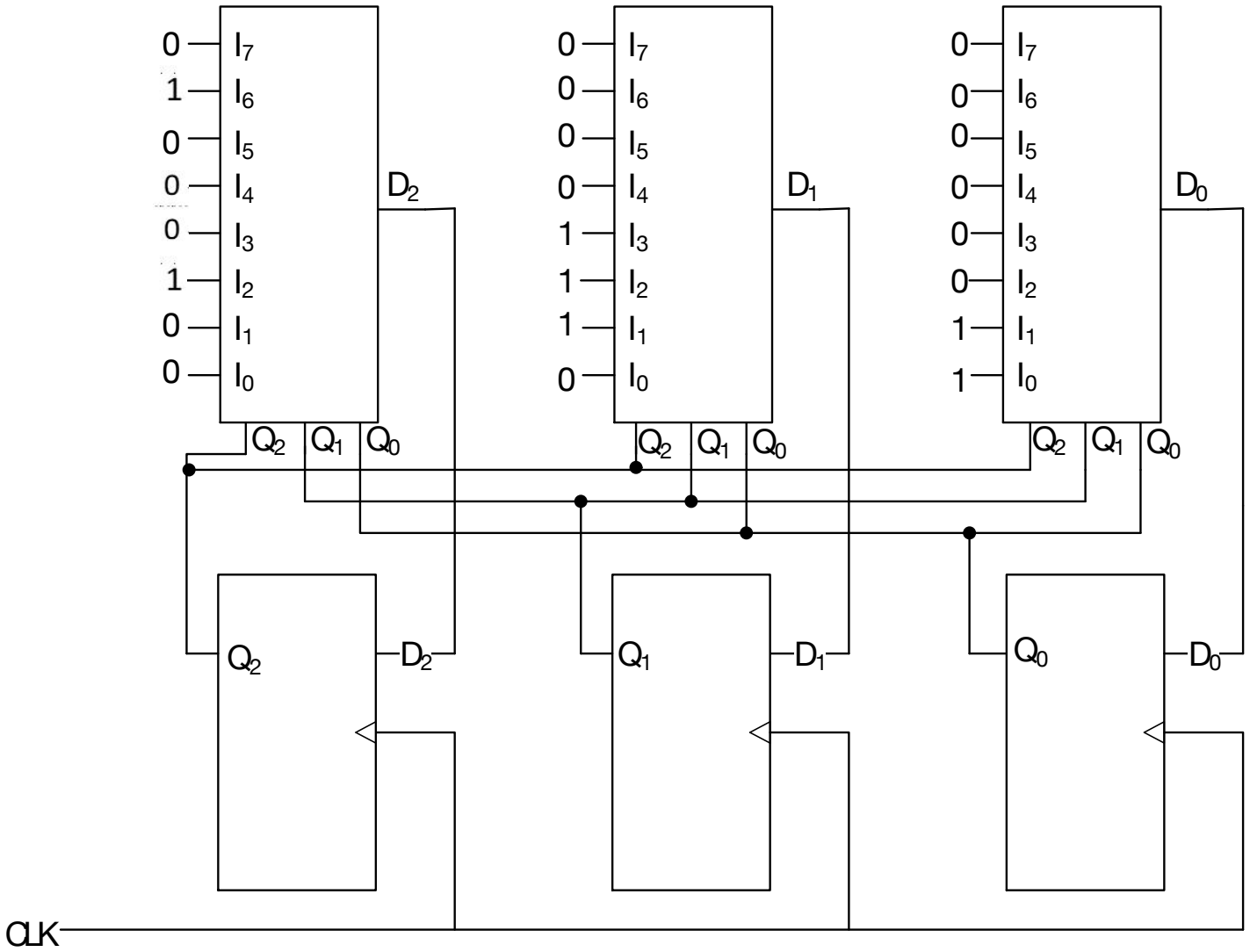
$D_0 = \overline{Q_2}\,\overline{Q_1}$

The K-maps relate the input parameters (the current state of the system) to the desired value of D_0, D_1 and D_2. We thus need three K-maps to simplify the logic as shown above.

Step 6: Implement the logic using either logic gates or MUXes.



Implementation using MUXes. Note that it is the current state of the flip flops (plus any control inputs) that act as the select lines on the MUXes. The required input values can be obtained directly from the truth table.



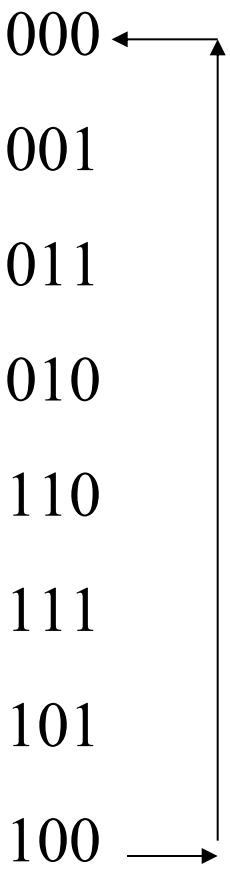
Exercise:

You need to design a 3-bit Gray code counter. Do this design using:

(a) J-K FF's and logic gates

(b) D FF's and MUXes

Remember: Gray code – only one bit changes at a time



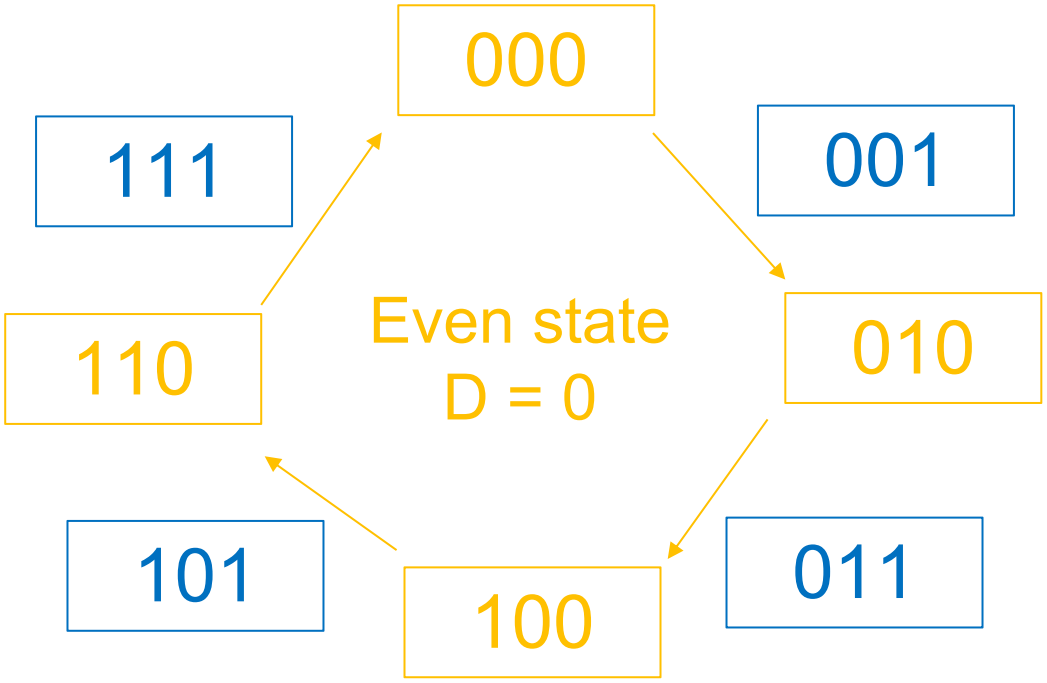
Challenge: Odd – Even counter design

Design a counter that will count through even 3-bit binary numbers (000 → 010 → 100 → 110 → 000) when a select input D=0 and will count through odd 3-bit binary numbers (001 → 011 → 101 → 111 → 001) when D=1.

When it is in the even count cycle and D changes to D=1, it should go to the first count in the odd sequence (001) and continue. Similarly, when it is in the odd count cycle and D changes to D=0, it should go to the first count in the even sequence (000) and continue.

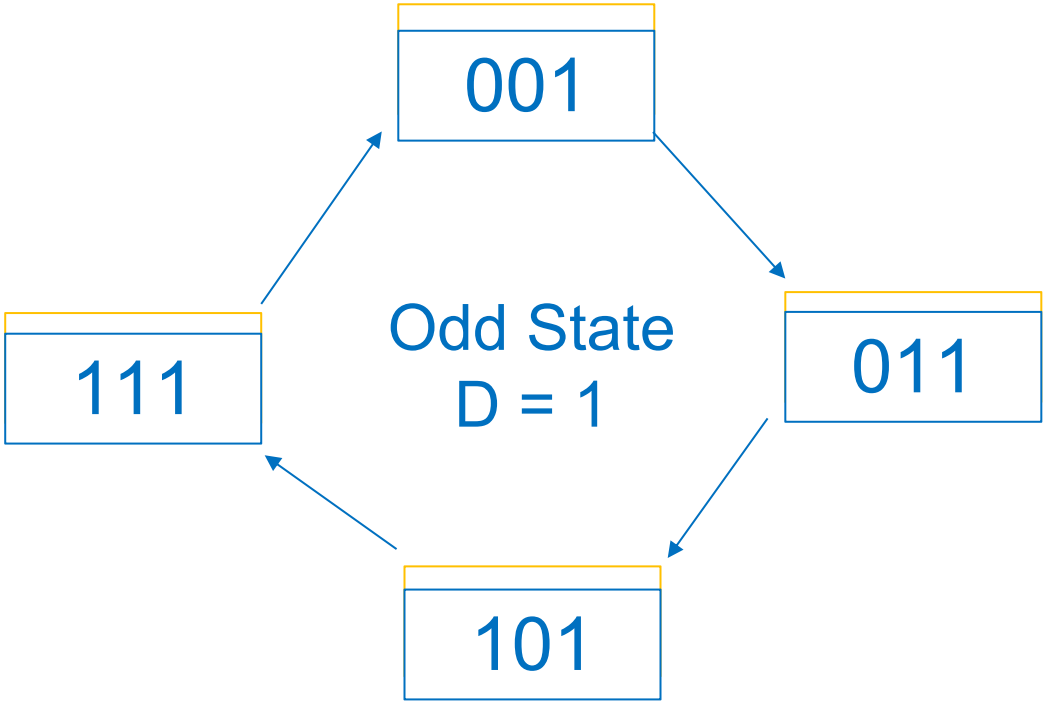
Use J-K flip-flops and logic gates or MUXes for your design.

The State Transition Diagram



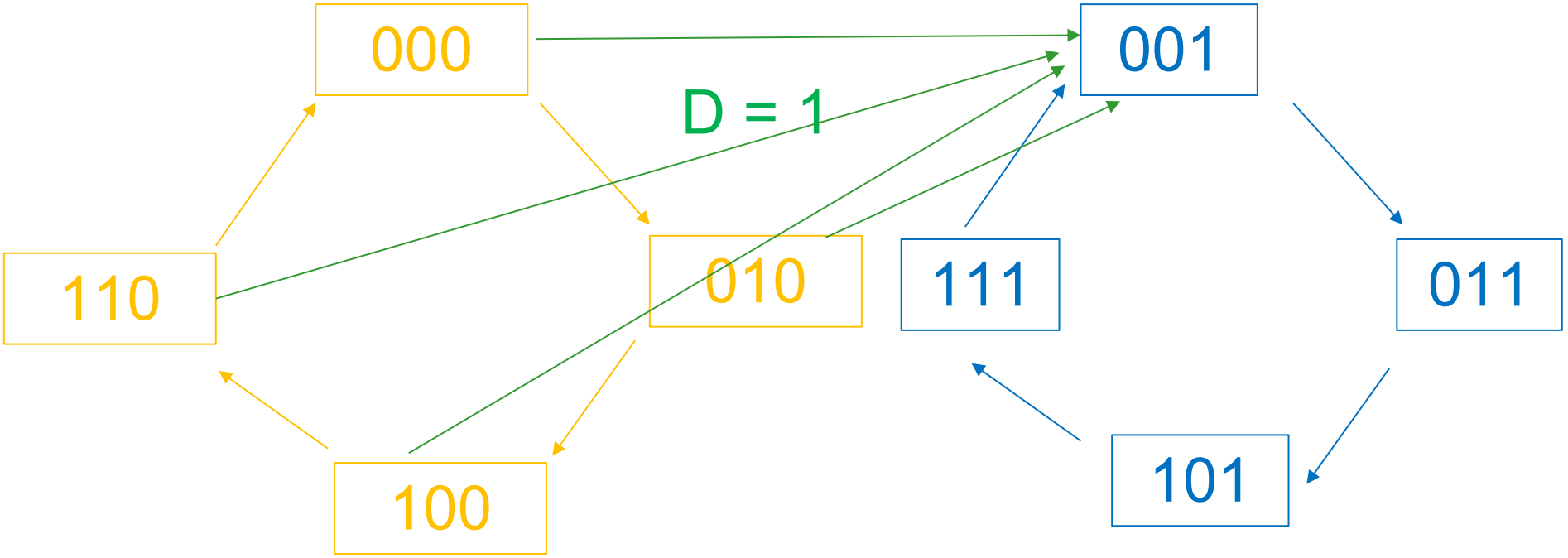
If the counter is in an even state (orange) and D=0 it will go to the next even state (ie 000 →010 → 100 → 110)

The State Transition Diagram



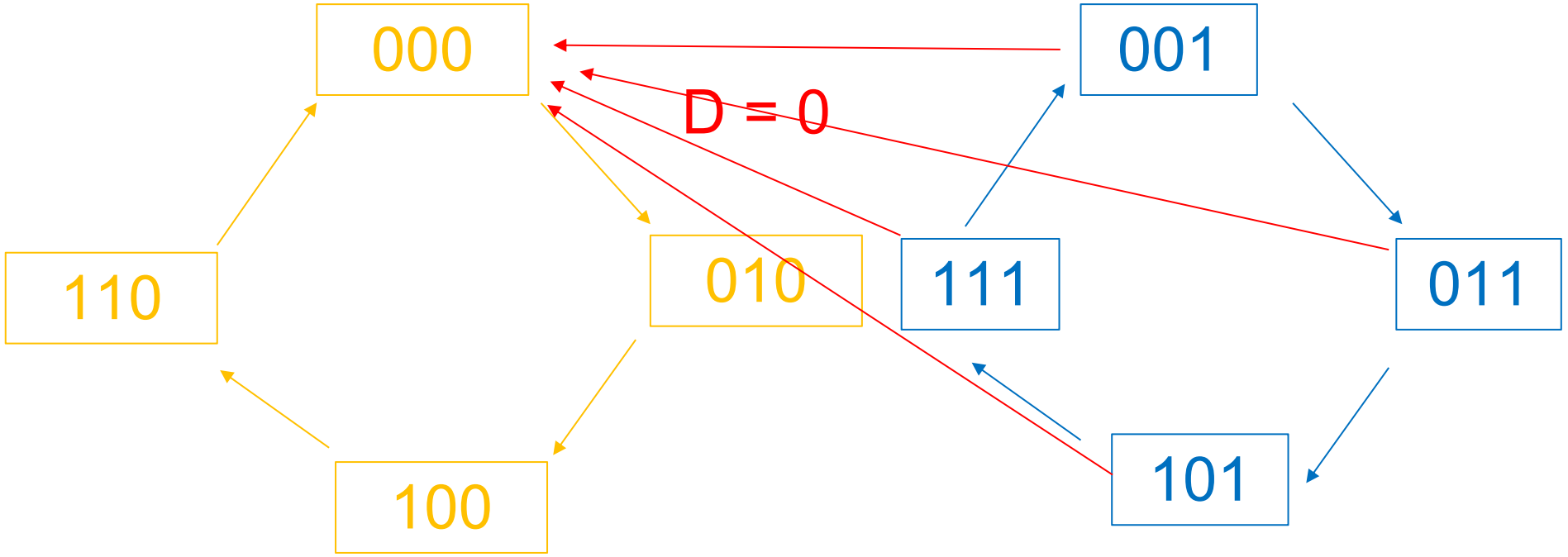
If the counter is in an odd state (blue) and D=1 it will go to the next odd state

The State Transition Diagram



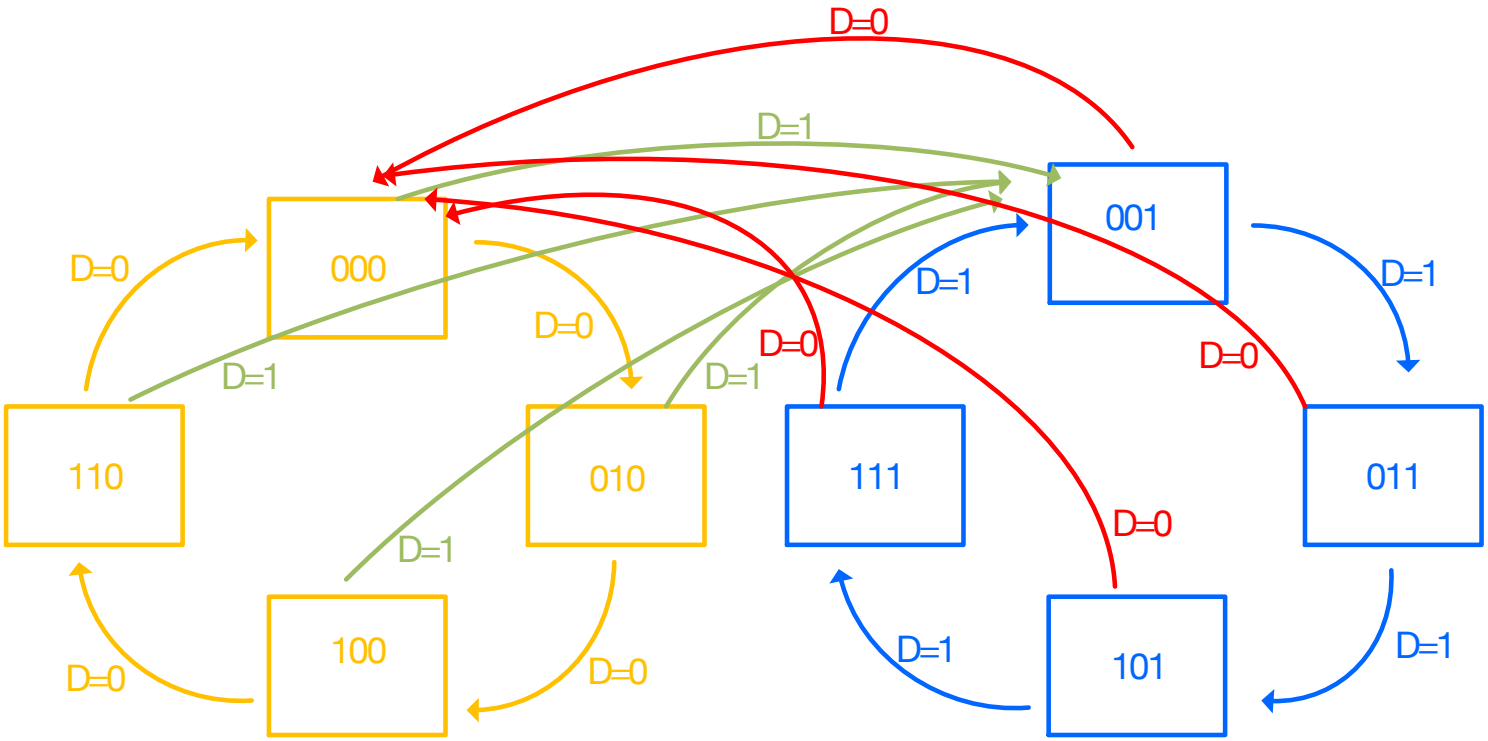
If it is in an even state and D=1 it will transition (green arrows) to the 001 state

The State Transition Diagram



If it is in an odd state and D=0 it will transition (red arrow lines) to the 000 state

The State Transition Diagram



1. If the counter is in an even state (orange) and D=0 it will go to the next even state

2. If the counter is in an odd state (blue) and D=1 it will go to the next odd state

3. If it is in an even state and D=1 it will transition (green) to the 001 state

4. If it is in an odd state and D=0 it will transition (red) to the 000 state

Reminder: the excitation table for J-K flip flops

J	K	CLK	Q
0	0	↑	Q_0 (no change)
1	0	↑	1
0	1	↑	0
1	1	↑	$\overline{Q_0}$ (toggles)

J	K	CLK	Q
0	0	↓	Q_0 (no change)
1	0	↓	1
0	1	↓	0
1	1	↓	$\overline{Q_0}$ (toggles)

Present Q	Next Q	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

	Present			Next								
D	Q2	Q1	Q0	Q2	Q1	Q0	J2	K2	J1	K1	J0	K0
0	0	0	0	0	1	0	0	X	1	X	0	X
0	0	0	1	0	0	0	0	X	0	X	X	1
0	0	1	0	1	0	0	1	X	X	1	0	X
0	0	1	1	0	0	0	0	X	X	1	X	1
0	1	0	0	1	1	0	X	0	1	X	0	X
0	1	0	1	0	0	0	X	1	0	X	X	1
0	1	1	0	0	0	0	X	1	X	1	0	X
0	1	1	1	0	0	0	X	1	X	1	X	1
1	0	0	0	0	0	1	0	X	0	X	1	X
1	0	0	1	0	1	1	0	X	1	X	X	0
1	0	1	0	0	0	1	0	X	X	1	1	X
1	0	1	1	1	0	1	1	X	X	1	X	0
1	1	0	0	0	0	1	X	1	0	X	1	X
1	1	0	1	1	1	1	X	0	1	X	X	0
1	1	1	0	0	0	1	X	1	X	1	1	X
1	1	1	1	0	0	1	X	1	X	1	X	0

	Present				Next									
D	Q2	Q1	Q0		Q2	Q1	Q0		J2	K2	J1	K1	J0	K0
0	0	0	0		0	1	0		0	X	1	X	0	X
0	0	0	1		0	0	0		0	X	0	X	X	1
0	0	1	0		1	0	0		1	X	X	1	0	X
0	0	1	1		0	0	0		0	X	X	1	X	1
0	1	0	0		1	1	0		X	0	1	X	0	X
0	1	0	1		0	0	0		X	1	0	X	X	1
0	1	1	0		0	0	0		X	1	X	1	0	X
0	1	1	1		0	0	0		X	1	X	1	X	1
1	0	0	0		0	0	1		0	X	0	X	1	X
1	0	0	1		0	1	1		0	X	1	X	X	0
1	0	1	0		0	0	1		0	X	X	1	1	X
1	0	1	1		1	0	1		1	X	X	1	X	0
1	1	0	0		0	0	1		X	1	0	X	1	X
1	1	0	1		1	1	1		X	0	1	X	X	0
1	1	1	0		0	0	1		X	1	X	1	1	X
1	1	1	1		0	0	1		X	1	X	1	X	0

For J_2

J_2	/Q1./Q0	/Q1.Q0	Q1.Q0	Q1./Q0
/D/Q2	0	0	0	1
/D.Q2	X	X	X	1(X)
D.Q2	X	X	(1)X	X
D./Q2	0	0	1	0

$$J_2 = DQ_1\overline{Q_0} + \overline{D}Q_1\overline{Q_0} = Q_1(DQ_0 + \overline{DQ_0}) = Q_1(\overline{D \oplus Q_0})$$

For K_2

K_2	/Q1./Q0	/Q1.Q0	Q1.Q0	Q1./Q0
/D/Q2	X	1(X)	1(X)	1(X)
/D.Q2	0	1	1	1
D.Q2	1	0	1	1
D./Q2	1(X)	X	1(X)	1(X)

$$K_2 = Q_1 + D\overline{Q_0} + \overline{D}Q_0 = Q_1 + (D \oplus Q_0)$$

For J_1

J_1	/Q1./Q0	/Q1.Q0	Q1.Q0	Q1./Q0
/D/Q2	1	0	X	1(X)
/D.Q2	1	0	X	1(X)
D.Q2	0	1	1(X)	X
D./Q2	0	1	1(X)	X

$$J_1 = D\overline{Q}_0 + \overline{D}\overline{Q}_0$$

For K_1

K_1	/Q1./Q0	/Q1.Q0	Q1.Q0	Q1./Q0
/D/Q2	1(X)	1(X)	1	1
/D.Q2	1(X)	1(X)	1	1
D.Q2	1(X)	1(X)	1	1
D./Q2	1(X)	1(X)	1	1

$$K_1 = 1$$

For J_0

J_0	/Q1./Q0	/Q1.Q0	Q1.Q0	Q1./Q0
/D/Q2	0	0(X)	0(X)	0
/D.Q2	0	0(X)	0(X)	0
D.Q2	1	1(X)	1(X)	1
D./Q2	1	1(X)	1(X)	1

$$J_0 = D$$

For K_0

K_0	/Q1./Q0	/Q1.Q0	Q1.Q0	Q1./Q0
/D/Q2	1(X)	1	1	1(X)
/D.Q2	1(X)	1	1	1(X)
D.Q2	0(X)	0	0	0(X)
D./Q2	0(X)	0	0	0(X)

$$K_0 = \overline{D}$$

We can easily implement this logic with the appropriate logic gates based on the logic expressions shown below:

$$J_2 = DQ_1Q_0 + \overline{D}Q_1\overline{Q_0} = Q_1(DQ_0 + \overline{D}\overline{Q_0}) = Q_1(\overline{D \oplus Q_0})$$

$$K_2 = Q_1 + D\overline{Q_0} + \overline{D}Q_0 = Q_1 + (D \oplus Q_0)$$

$$J_1 = D\,Q_0 + \overline{D}\,\overline{Q}_0 \qquad K_1 = 1$$

$$J_0 = D \qquad K_0 = \overline{D}$$

We can easily implement this logic with the appropriate logic gates. For you to sketch the required logic circuits ….

However, we would like to check if the logic is easier to implement by means of MUXes.

As we have four variables (D, Q2, Q1, Q0) we would most likely need 6 of 16-1 MUXes.

However, we can make several simplifications:

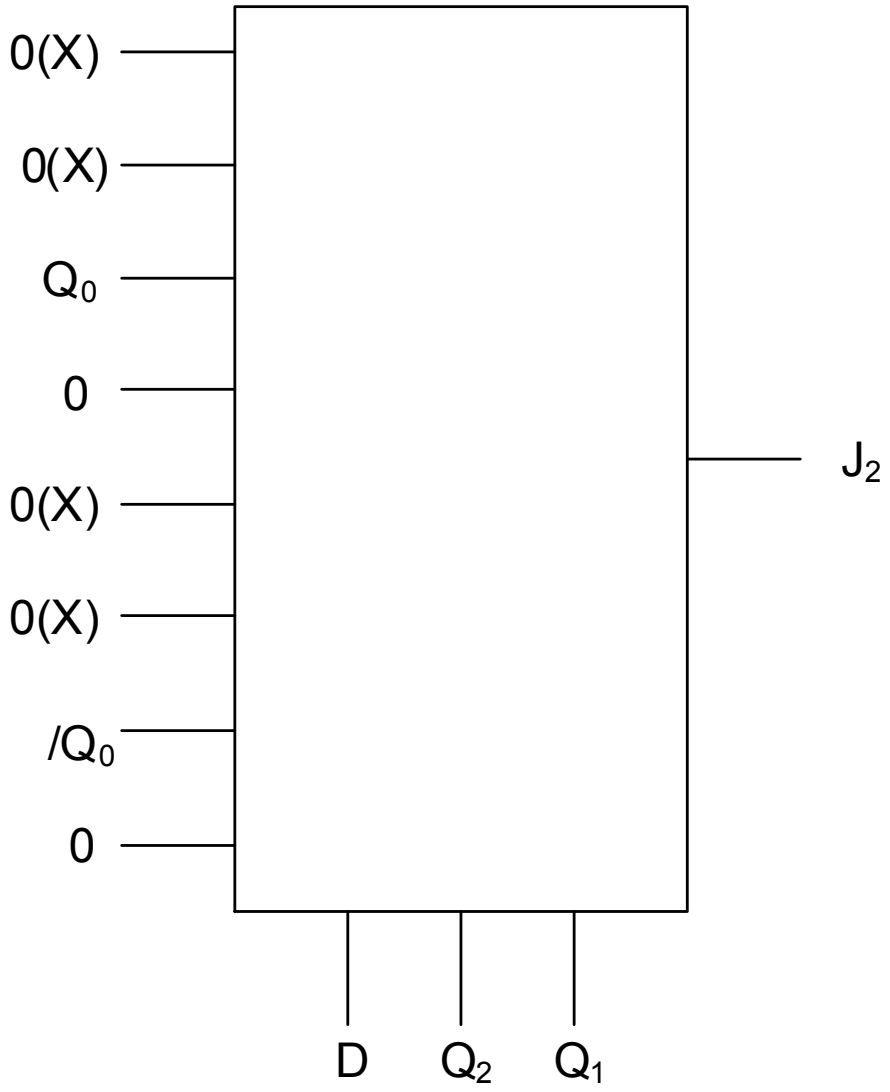
1. We can use 8:1 Muxes for all six outputs if we only use D, Q_2 and Q_1 as the select lines on the Muxes and use Q_0 as an input into the Mux where needed. We should thus look at the output pattern required in terms of Q0.

2. It is obvious that a Mux is not needed for K_1, as K_1 =1

3. We also do not need to use Muxes for J_0 or K_0 as we can simply write these in terms of D or /D.

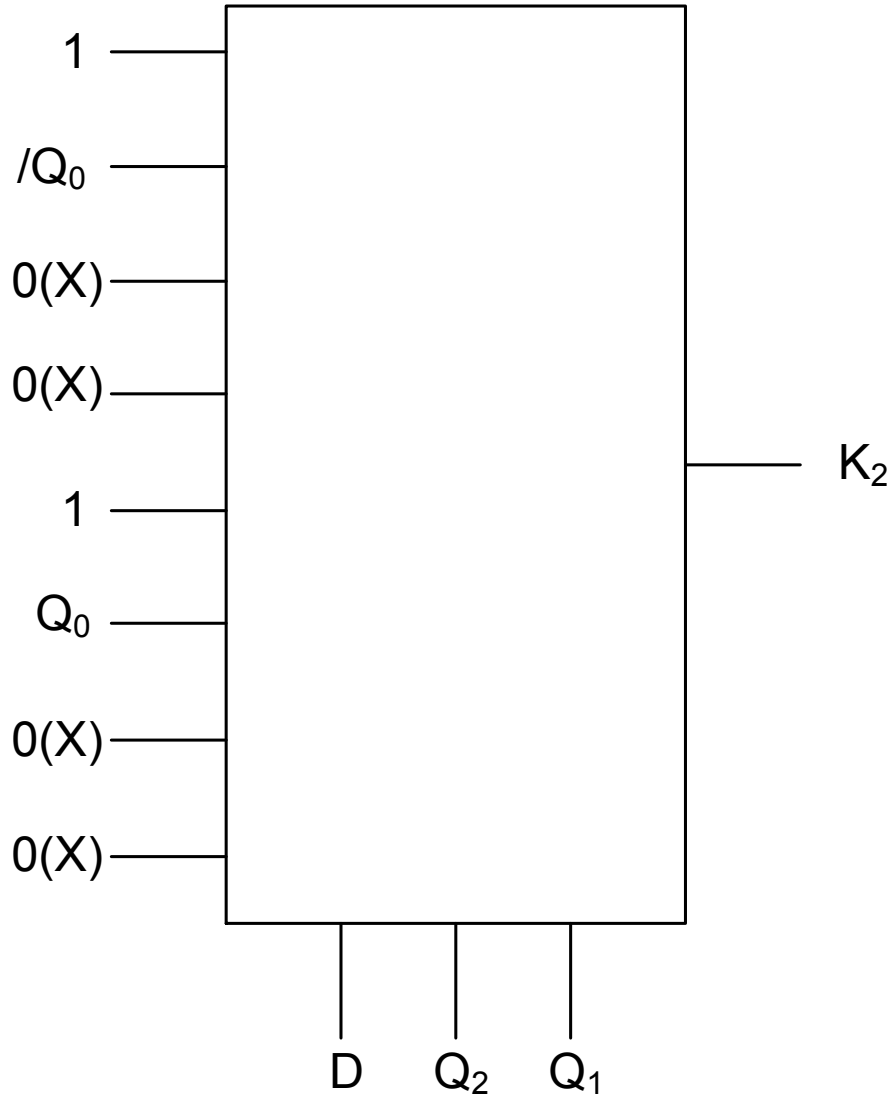
	Present				Next									
D	Q2	Q1	Q0		Q2	Q1	Q0		J2	K2	J1	K1	J0	K0
0	0	0	0		0	1	0		0	X	1	X	0	X
0	0	0	1		0	0	0		0	X	0	X	X	1
0	0	1	0		1	0	0		1	X	X	1	0	X
0	0	1	1		0	0	0		0	X	X	1	X	1
0	1	0	0		1	1	0		X	0	1	X	0	X
0	1	0	1		0	0	0		X	1	0	X	X	1
0	1	1	0		0	0	0		X	1	X	1	0	X
0	1	1	1		0	0	0		X	1	X	1	X	1
1	0	0	0		0	0	1		0	X	0	X	1	X
1	0	0	1		0	1	1		0	X	1	X	X	0
1	0	1	0		0	0	1		0	X	X	1	1	X
1	0	1	1		1	0	1		1	X	X	1	X	0
1	1	0	0		0	0	1		X	1	0	X	1	X
1	1	0	1		1	1	1		X	0	1	X	X	0
1	1	1	0		0	0	1		X	1	X	1	1	X
1	1	1	1		0	0	1		X	1	X	1	X	0

$$J_2 = DQ_1Q_0 + \overline{D}Q_1\overline{Q_0} = Q_1(DQ_0 + \overline{D}\overline{Q_0}) = Q_1(\overline{D \oplus Q_0})$$

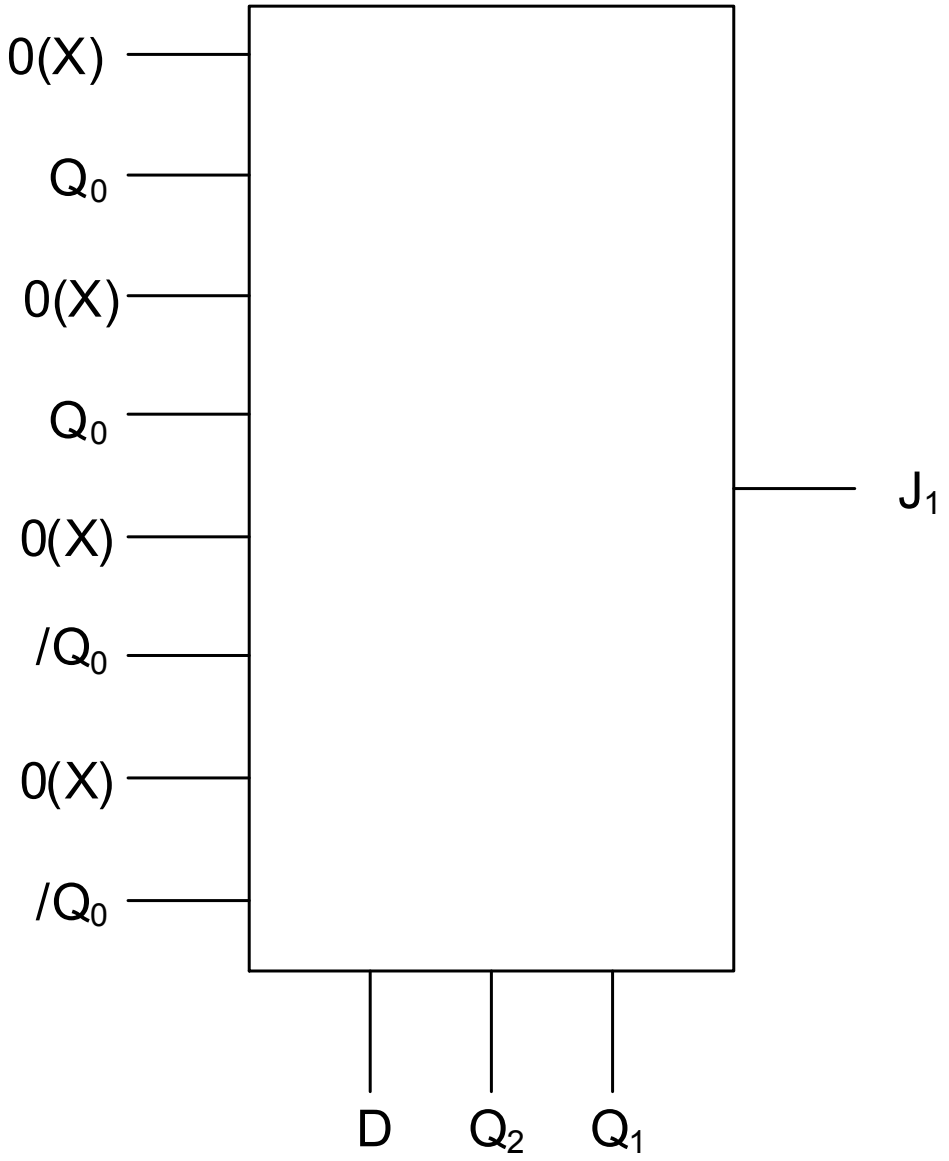


	Present				Next									
D	Q2	Q1	Q0		Q2	Q1	Q0		J2	K2	J1	K1	J0	K0
0	0	0	0		0	1	0		0	X	1	X	0	X
0	0	0	1		0	0	0		0	X	0	X	X	1
0	0	1	0		1	0	0		1	X	X	1	0	X
0	0	1	1		0	0	0		0	X	X	1	X	1
0	1	0	0		1	1	0		X	0	1	X	0	X
0	1	0	1		0	0	0		X	1	0	X	X	1
0	1	1	0		0	0	0		X	1	X	1	0	X
0	1	1	1		0	0	0		X	1	X	1	X	1
1	0	0	0		0	0	1		0	X	0	X	1	X
1	0	0	1		0	1	1		0	X	1	X	X	0
1	0	1	0		0	0	1		0	X	X	1	1	X
1	0	1	1		1	0	1		1	X	X	1	X	0
1	1	0	0		0	0	1		X	1	0	X	1	X
1	1	0	1		1	1	1		X	0	1	X	X	0
1	1	1	0		0	0	1		X	1	X	1	1	X
1	1	1	1		0	0	1		X	1	X	1	X	0

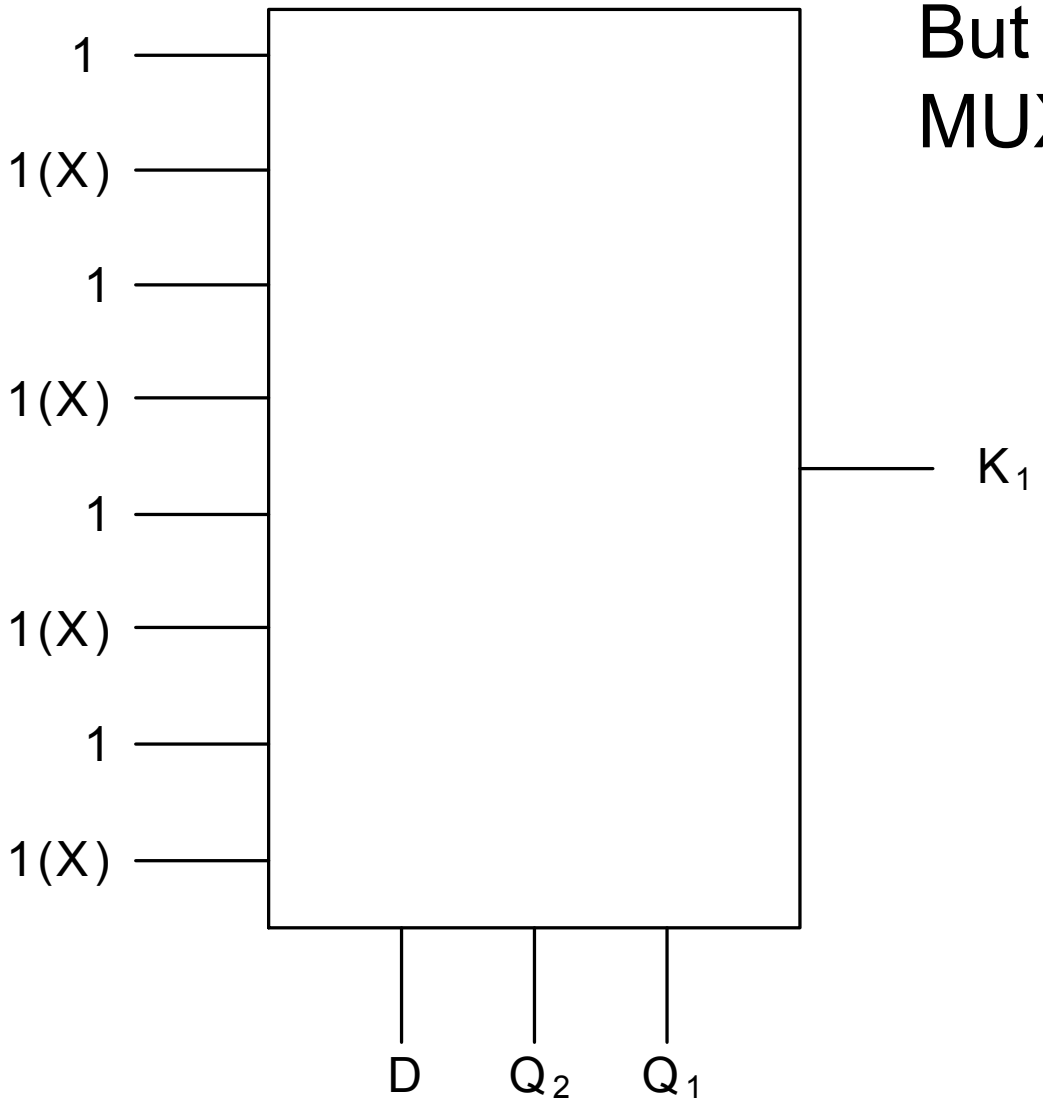
$$K_2 = Q_1 + D\overline{Q_0} + \overline{D}Q_0 = Q_1 + (D \oplus Q_0)$$



$$J_1 = D\,Q_0 + \overline{D}\,\overline{Q}_0$$

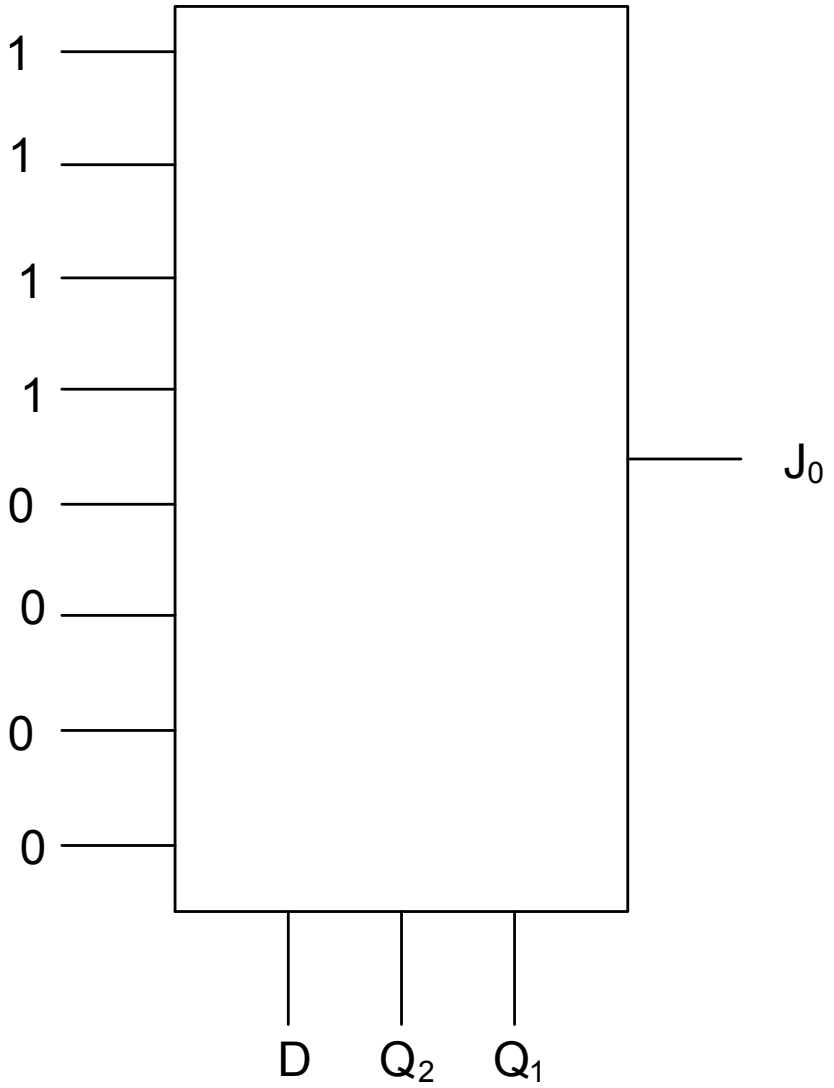


$K_1 = 1$



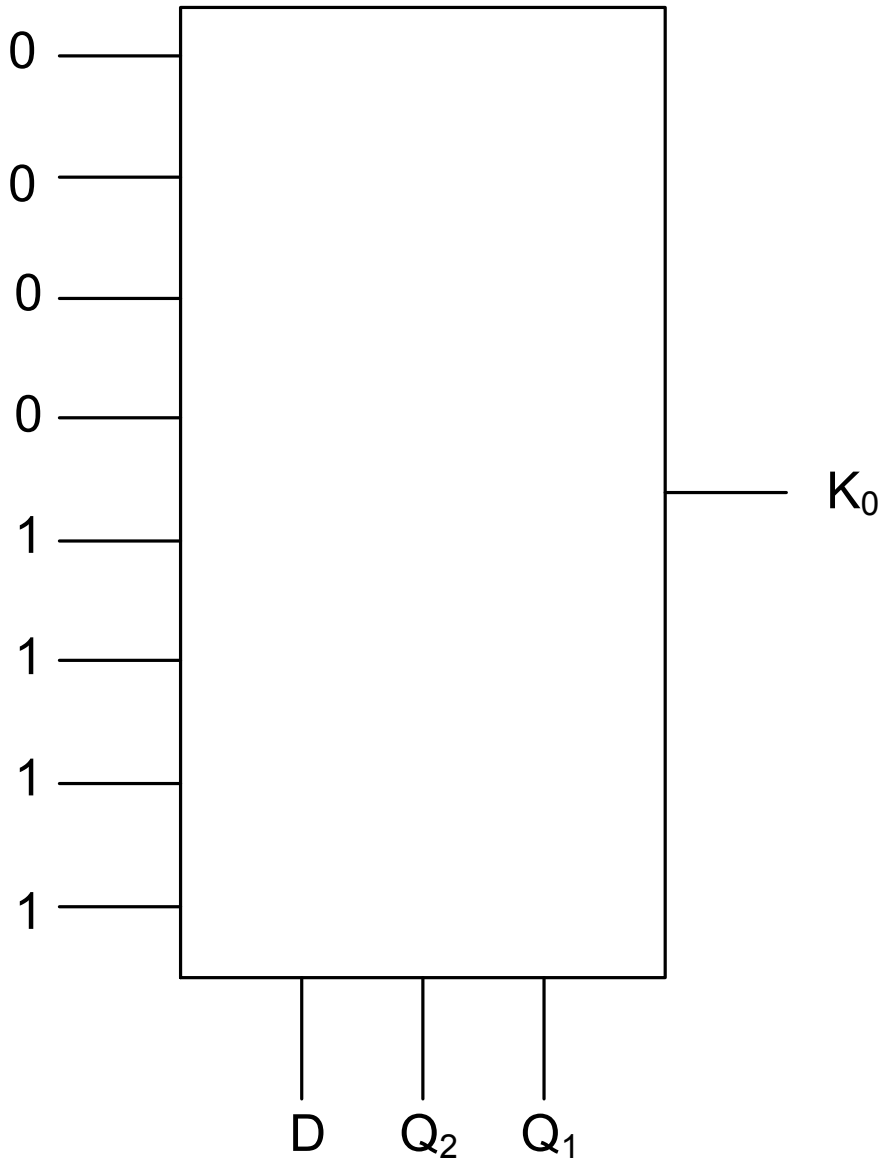
But we would not use a MUX as $K_1 = 1$

$J_0 = D$



Again do not use a MUX as we have shown that $J_0 = D$

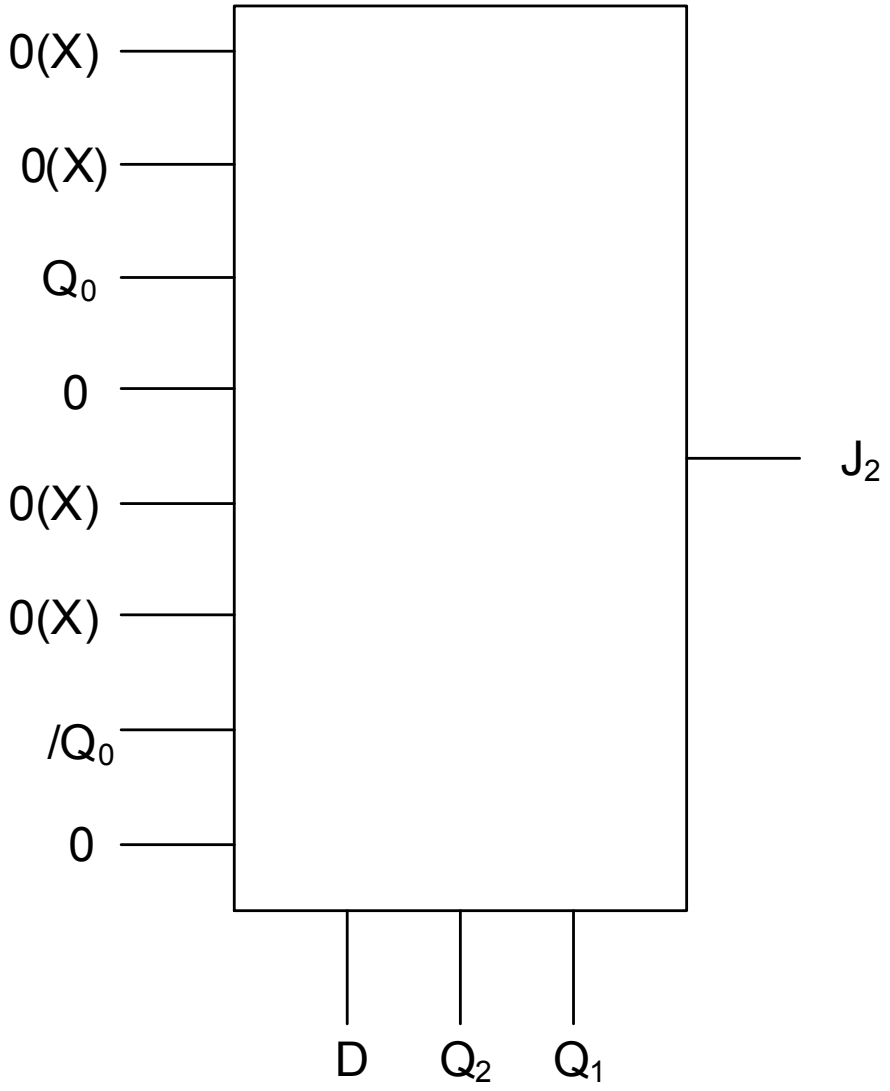
$K_0 = \overline{D}$

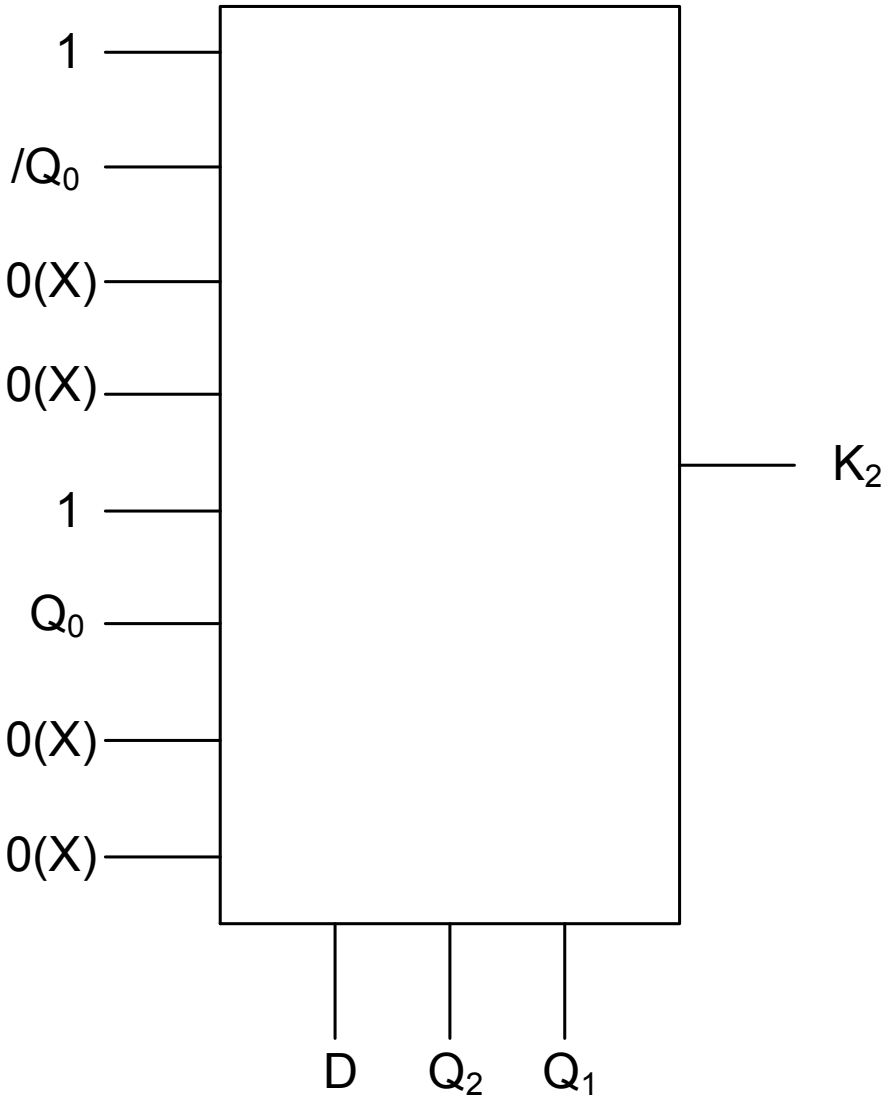


Again do not use a MUX as we have shown that $K_0 = /D$

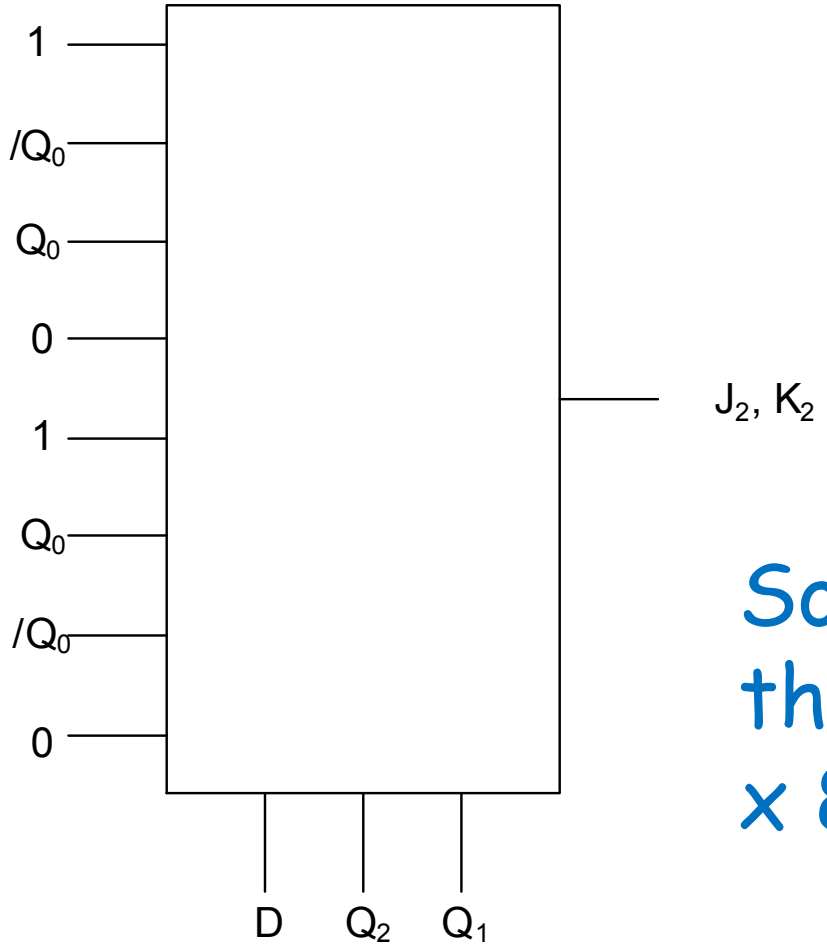
So we can implement this logic using only 3 x 8:1 muxes ! How does this compare to the number of logic gates we will have to use in the discrete solution ?

Can we simplify the design further?





We can make a further simplification by combining J_2 and K_2 into a single mux by careful choice of the X states:



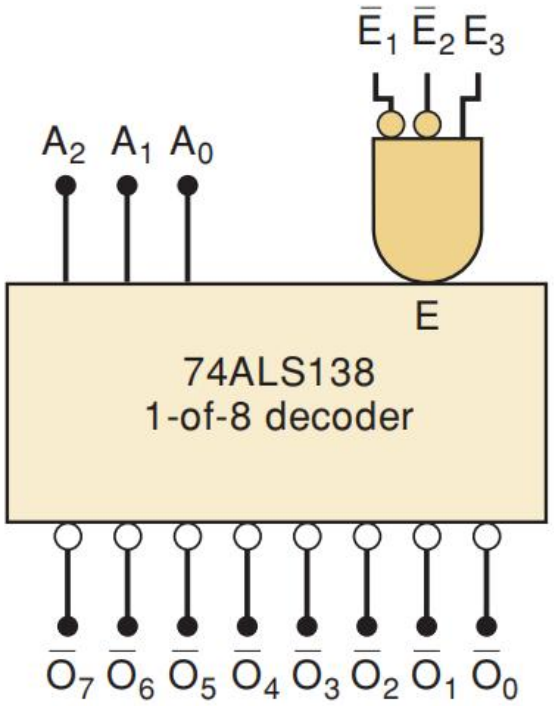
So we can implement this logic using only 2 x 8:1 muxes !

74ALS138 decoder 1

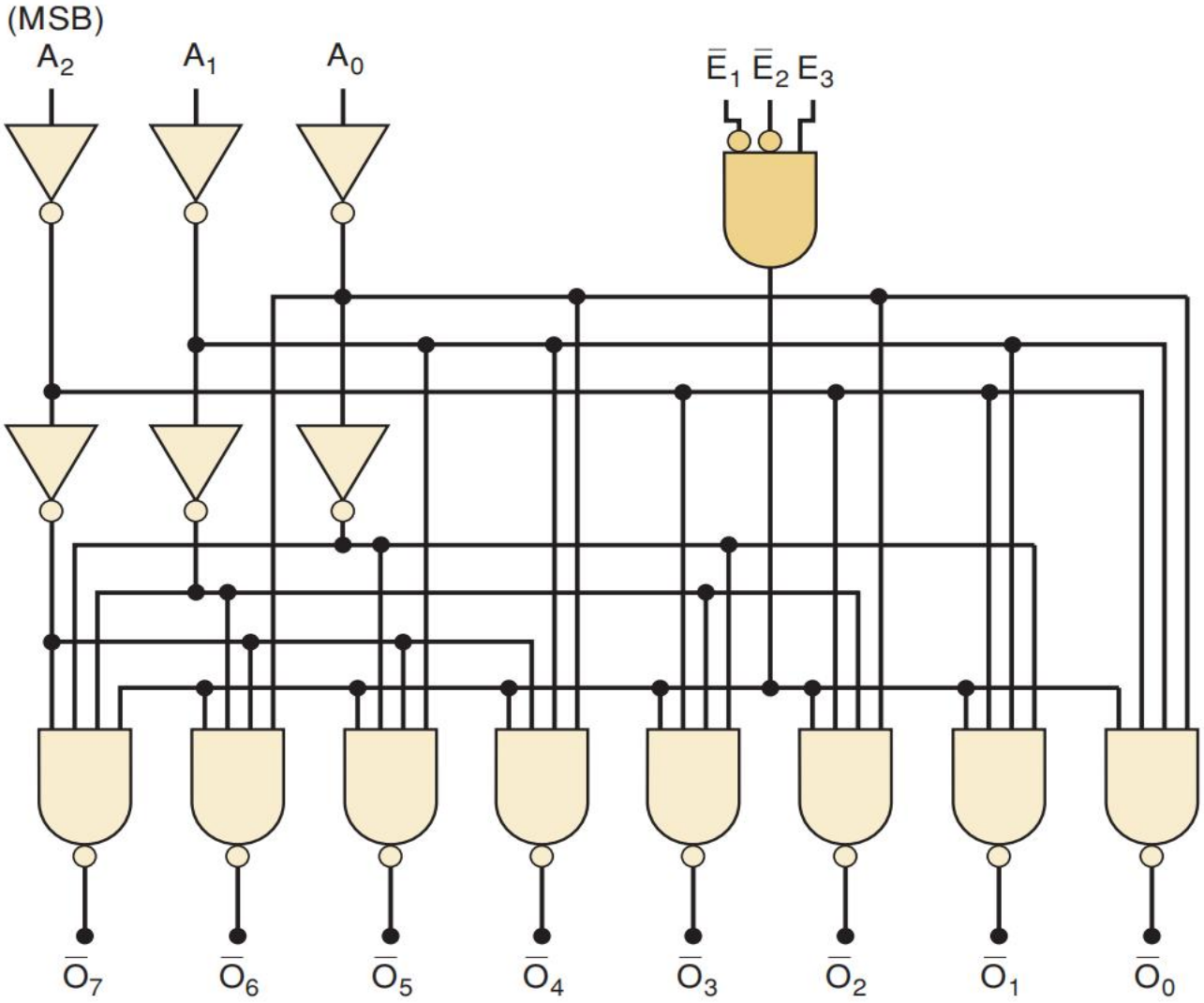
C	B	A	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

74ALS138 decoder 2

$\overline{E}_1$	$\overline{E}_2$	E_3	Outputs
0	0	1	Respond to input code $A_2A_1A_0$
1	X	X	Disabled – all HIGH
X	1	X	Disabled – all HIGH
X	X	0	Disabled – all HIGH



74ALS138 decoder 3



Exercise: Binary Coded Decimal

Consider again the 4-bit binary number abcd, driving a 7-segment LED display. Simplify the expression that will light up the top right led on a 7-segment display.

dc \ ba	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11				
10	8	9		

dc \ ba	00	01	11	10
00				
01				
11				
10				