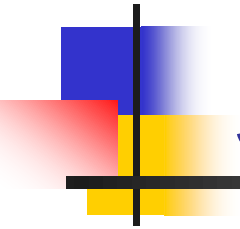


Synchronous Sequential Circuits: Design Procedure and Examples

Overview

- Sequential Circuit Design
- Sequential Circuit Design Procedure
- Design Example1: Sequence Recognizer
  - Sequence Recognizer as ***Mealy*** Finite State Machine
  - Design using ***JK*** Flip-Flops
  - Design using ***D*** Flip-Flops
  - Design Comparison
- Design Example2: Cyclic Shifter
  - Cyclic Shifter as ***Moore*** Finite State Machine
  - Sequential circuits with unused states
  - Design using ***don't care*** conditions for unused states
  - Design using ***explicit*** specification for unused states

Sequential Circuit Design

- In sequential circuit design, we turn some description into a working circuit.
- Start: With a list of specifications (descriptions):
  - Behavior description of the circuit
  - Type of Flip-Flops to be used (SR or JK or D or T)
  - Type of gates to be used
  - …
- End: With a logic diagram OR list of Boolean functions.
- NOTE:
  - # Flip-Flops to be used depends on the # of states. At most 2^n states can be represented with n Flip-Flops.
  - The binary coding of the states and the type of the Flip-Flops determine the complexity of the circuit.

Sequential Circuit Design Procedure

- Step 1: Given the problem statement, derive the state table:
  - The table should show inputs, present states, next states and outputs.
  - It may be easier to find a state diagram first, and then convert that to a table.
- Step 2 (optional): Apply state-reduction methods to reduce (if possible) the number of states.
  - We will not discuss state-reduction methods in this course.
- Step 3: Assign binary codes to the states in the state table, if you haven't already.
  - If you have n states, your binary codes will have at least $\lceil \log_2 n \rceil$ bits.
- Step 4: Determine the number of Flip-Flops needed and the type of Flip-Flops to be used:
  - If you have n states, your circuit will have at least $\lceil \log_2 n \rceil$ Flip-Flops.
  - The types of Flip-Flops may be given in the initial specification.
  - If not, select the type according to some criteria, e.g., to get simpler circuit or to make the design procedure easier.

Sequential Circuit Design Procedure (cont.)

- Step 5: For each flip-flop and each row of your state table, find the flip-flop input values that are needed to generate the next state from the present state:
  - You can use Flip-Flop excitation tables here.
- Step 6: Derive the characteristic (Flip-Flop input) equations from the state table.
- Step 7: Derive the primary output equations from the state table.
- Step 8: Simplify the Flip-Flop input equations and output equations:
  - Use K-maps or
  - Other simplification methods
- Step 9: Draw the logic diagram of the circuit.

Sequence Recognizers

- I will explain the Design Procedure in detail by designing a sequence recognizer circuit.
- A sequence recognizer is a special kind of sequential circuit that looks for a special bit pattern in some input.
- The recognizer circuit has only one input, X.
  - One bit of input is supplied on every clock cycle. For example, it would take 20 cycles to scan a 20-bit input.
  - This is an easy way to permit arbitrarily long input sequences.
- There is one output, Z, which is 1 when the desired pattern is found.
- Our example will detect the bit pattern "1001":
  - Input X: …00011100110100100110 …
  - Output Z: …00000000100000100100 …
  - Here, one input and one output bit appear every clock cycle.
- This requires a sequential circuit because the circuit has to "remember" the inputs from previous clock cycles, in order to determine whether or not a match was found.

Step 1: Deriving the State Table

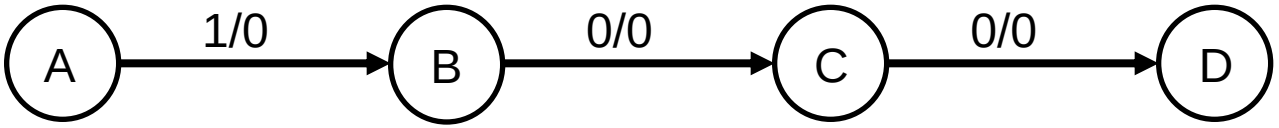
- Problem Statement: Design the sequence recognizer circuit described in the previous slide.
- The first thing to figure out is precisely how the use of states will help to solve the given problem.
  - Make a state table based on the problem statement. The table should show the present states, inputs, next states and outputs.
  - Sometimes it is easier to first find a state diagram and then convert that to a table.
- This is usually the most difficult step. Why?
  - There is not a formal procedure how to derive a state table or state diagram from a problem specification such as the one we have here.
  - In Step 1 you have to relay on your knowledge and design experience.
  - Currently, Step 1 is more an art than a science!
- Once you have the state table, the rest of the design procedure is the same for all sequential circuits.

Step 1: Deriving the State Table (cont.)

- How many and What states do we need for the sequence recognizer?
  - We have to “remember” inputs from previous clock cycles.
  - For example, if the previous three inputs were 100 and the current input is 1, then the output should be 1.
  - In general, we will have to remember occurrences of parts of the desired pattern - in this case, 1, 10, and 100.
  - So, we need the following four states:

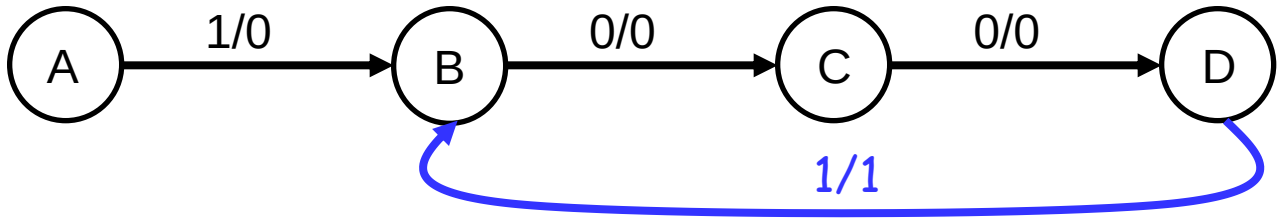
State	Meaning
A	None of the desired pattern (1001) has been input yet.
B	We’ve already seen the first bit (1) of the desired pattern.
C	We’ve already seen the first two bits (10) of the desired pattern.
D	We’ve already seen the first three bits (100) of the desired pattern.

- We will derive a state diagram before deriving the state table.
  - First, we draw a part of the state diagram:



Step 1: Deriving the State Table (cont.)

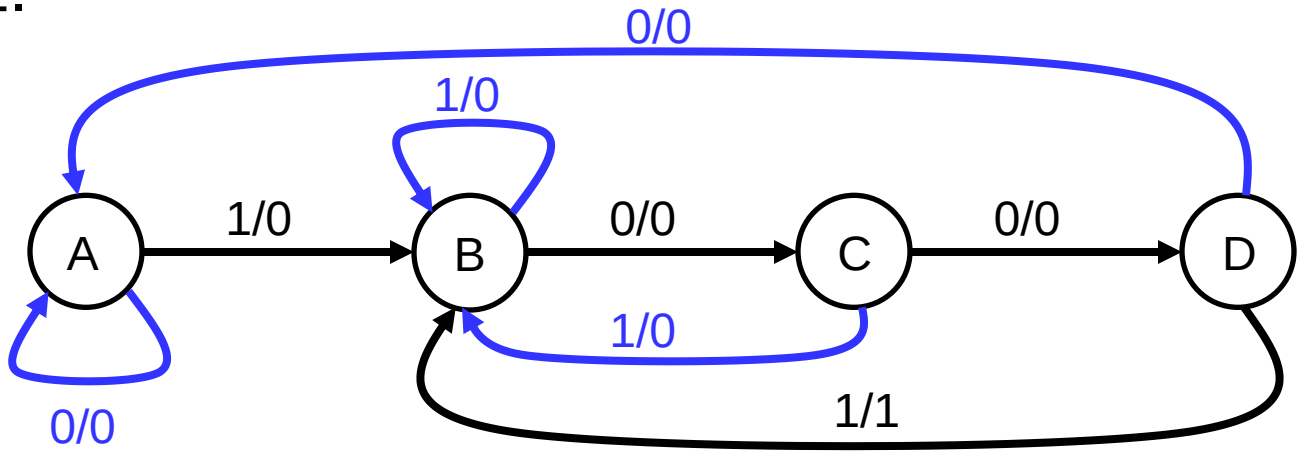
- What happens if we are in state D (the last three inputs were 100), and the current input is 1?
  - The output should be a 1, because we've found the desired pattern.
  - But this last 1 could also be the start of another occurrence of the pattern! For example, 1001001 contains *two* occurrences of 1001.
  - To detect overlapping occurrences of the pattern, the next state should be B.



State	Meaning
A	None of the desired pattern (1001) has been input yet.
B	We've already seen the first bit (1) of the desired pattern.
C	We've already seen the first two bits (10) of the desired pattern.
D	We've already seen the first three bits (100) of the desired pattern.

Step 1: Deriving the State Table (cont.)

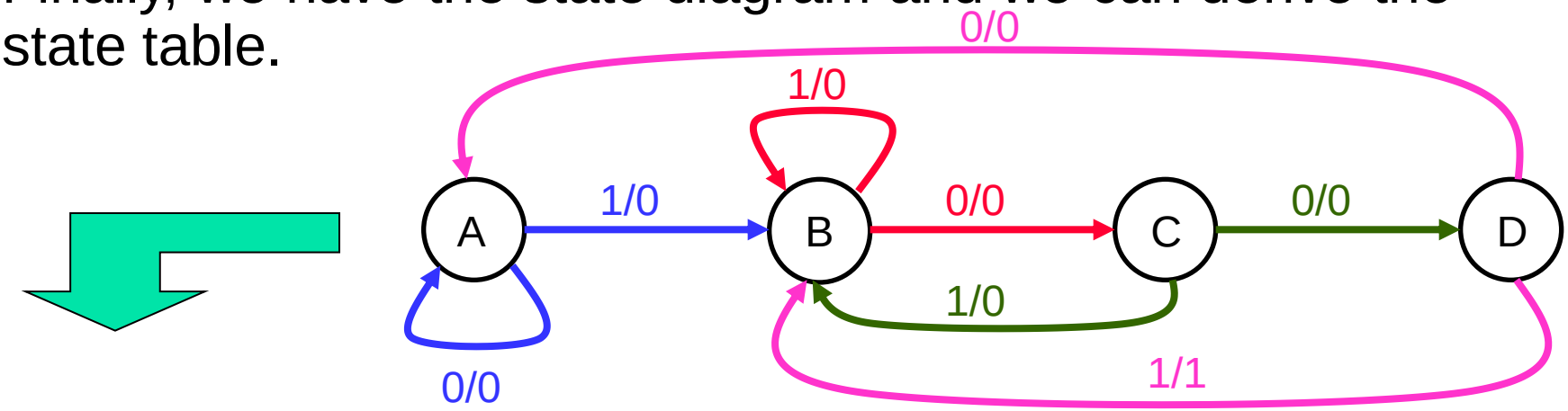
- Remember that we need *two* outgoing arrows for each node, to account for the possibilities of X = 0 and X = 1.
- The remaining arrows we need are shown in blue. They also allow for the correct detection of overlapping occurrences of 1001.



State	Meaning
A	None of the desired pattern (1001) has been input yet.
B	We've already seen the first bit (1) of the desired pattern.
C	We've already seen the first two bits (10) of the desired pattern.
D	We've already seen the first three bits (100) of the desired pattern.

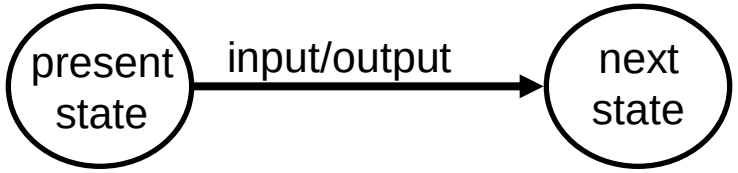
Step 1: Deriving the State Table (cont.)

- Finally, we have the state diagram and we can derive the state table.



Input X	Present State	Next State	Output Z
0	A	A	0
1	A	B	0
0	B	C	0
1	B	B	0
0	C	D	0
1	C	B	0
0	D	A	0
1	D	B	1

Remember how the state diagram arrows correspond to rows of the state table:



Step 3: Assigning Binary Codes to States

- We skip Step 2 because in this course we will not discuss state-reduction methods.
- We have four states A,B,C, and D, so we need at least two bits Q_1 and Q_2 to encode the states.
- There are many possible ways to encode the states:

State		Q_1Q_2	Q_1Q_2	Q_1Q_2	Q_1Q_2	…
A		0 0	0 1	1 0	1 1	…
B		0 1	0 0	0 0	0 1	…
C		1 0	1 0	0 1	0 0	…
D		1 1	1 1	1 1	1 0	…

- The state code can have a big impact on circuit complexity, but we will not study this in this course.
- So, we take an arbitrary code. For example, we represent state A with Q_1Q_2 = **10**, B with **00**, C with **01**, and D with **11**.

Step 3: Assigning Binary Codes to States (cont.)

- We fill the state table with the selected state code.
- Recall, we selected to represent state **A** with **10**, **B** with **00**, **C** with **01**, and **D** with **11**.

State Table

Input X	Present State	Next State	Output Z
0	A	A	0
1	A	B	0
0	B	C	0
1	B	B	0
0	C	D	0
1	C	B	0
0	D	A	0
1	D	B	1

Encoded State Table

	Input	Present State		Next State		Output
	X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	Z(t)
2	0	1	0	1	0	0
6	1	1	0	0	0	0
0	0	0	0	0	1	0
4	1	0	0	0	0	0
1	0	0	1	1	1	0
5	1	0	1	0	0	0
3	0	1	1	1	0	0
7	1	1	1	0	0	1

NOTE: The rows of the **Encoded State Table** are not ordered as we are used to. Reorder the rows to make the table ordered. This will make further design steps easier.

Step 3: Assigning Binary Codes to States (cont.)

- Ordered Encoded State Table:

	Input	Present State		Next State		Output
	X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	Z(t)
0	0	0	0	0	1	0
1	0	0	1	1	1	0
2	0	1	0	1	0	0
3	0	1	1	1	0	0
4	1	0	0	0	0	0
5	1	0	1	0	0	0
6	1	1	0	0	0	0
7	1	1	1	0	0	1

- This table contains the same information as the table from the previous slide. Just, the order of the rows is different.

Step 4: Determine the number and type of Flip-Flops to be used

- We have 4 states and we have encoded them with 2 bits Q_1 and Q_2. Thus, we need 2 Flip-Flops.
- Here, we will use ***JK*** Flip-Flops (later you will see the same example using D Flip-Flops).
- Thus, for each Flip-Flip Q_i, we add two columns (for J_i and K_i) in the state table.

	Input	Present State		Next State		Flip-Flop Inputs for Q_1		Flip-Flop Inputs for Q_2		Output
	X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	J_1	K_1	J_2	K_2	Z(t)
0	0	0	0	0	1					0
1	0	0	1	1	1					0
2	0	1	0	1	0					0
3	0	1	1	1	0					0
4	1	0	0	0	0					0
5	1	0	1	0	0					0
6	1	1	0	0	0					0
7	1	1	1	0	0					1

Step 5: Finding Flip-Flop Input Values

- How to actually make the Flip-Flops change from their present state into the desired next state.
- For each Flip-Flip Q_i, look at its present and next states, and determine what the inputs J_i and K_i should be in order to make that state change.
  - Use the ***JK*** execution table.

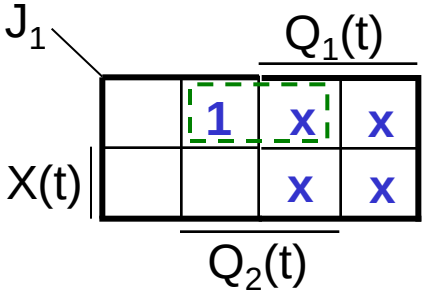
_JK_ Execution Table

Q(t)	Q(t+1)	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

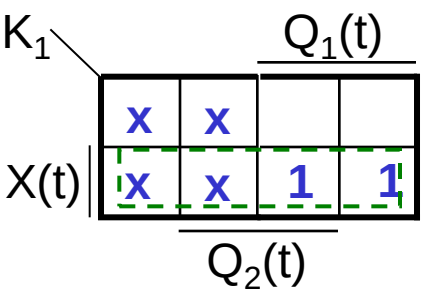
Input	Present State		Next State		Flip-Flop Inputs for Q_1		Flip-Flop Inputs for Q_2		Output
X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	J_1	K_1	J_2	K_2	Z(t)
0	0	0	0	1	0	x	1	x	0
0	0	1	1	1	1	x	x	0	0
0	1	0	1	0	x	0	0	x	0
0	1	1	1	0	x	0	x	1	0
1	0	0	0	0	0	x	0	x	0
1	0	1	0	0	0	x	x	1	0
1	1	0	0	0	x	1	0	x	0
1	1	1	0	0	x	1	x	1	1

Step 6 and 8: Deriving Simplified Flip-Flop Input Equations

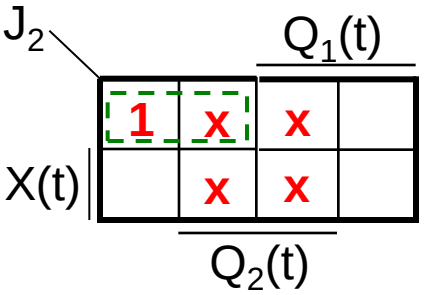
Input	Present State		Next State		Flip-Flop Inputs for Q_1		Flip-Flop Inputs for Q_2		Output
X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	J_1	K_1	J_2	K_2	Z(t)
0	0	0	0	1	0	x	1	x	0
0	0	1	1	1	1	x	x	0	0
0	1	0	1	0	x	0	0	x	0
0	1	1	1	0	x	0	x	1	0
1	0	0	0	0	0	x	0	x	0
1	0	1	0	0	0	x	x	1	0
1	1	0	0	0	x	1	0	x	0
1	1	1	0	0	x	1	x	1	1



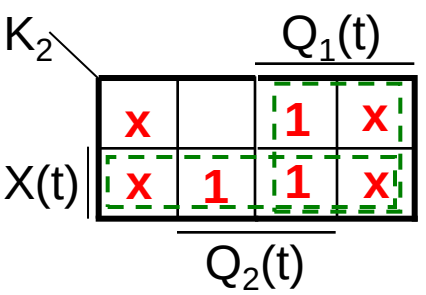
$J_1 = X(t)' \cdot Q_2(t)$



$K_1 = X(t)$



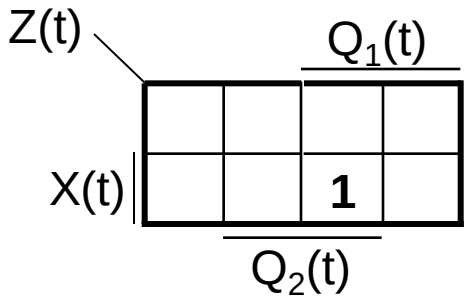
$J_2 = X(t)' \cdot Q_1(t)'$



$K_2 = X(t) + Q_1(t)$

Step 7 and 8: Deriving Simplified Primary Output Equations

Input	Present State		Next State		Flip-Flop Inputs for Q_1		Flip-Flop Inputs for Q_2		Output
X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	J_1	K_1	J_2	K_2	Z(t)
0	0	0	0	1	0	x	1	x	0
0	0	1	1	1	1	x	x	0	0
0	1	0	1	0	x	0	0	x	0
0	1	1	1	0	x	0	x	1	0
1	0	0	0	0	0	x	0	x	0
1	0	1	0	0	0	x	x	1	0
1	1	0	0	0	x	1	0	x	0
1	1	1	0	0	x	1	x	1	1



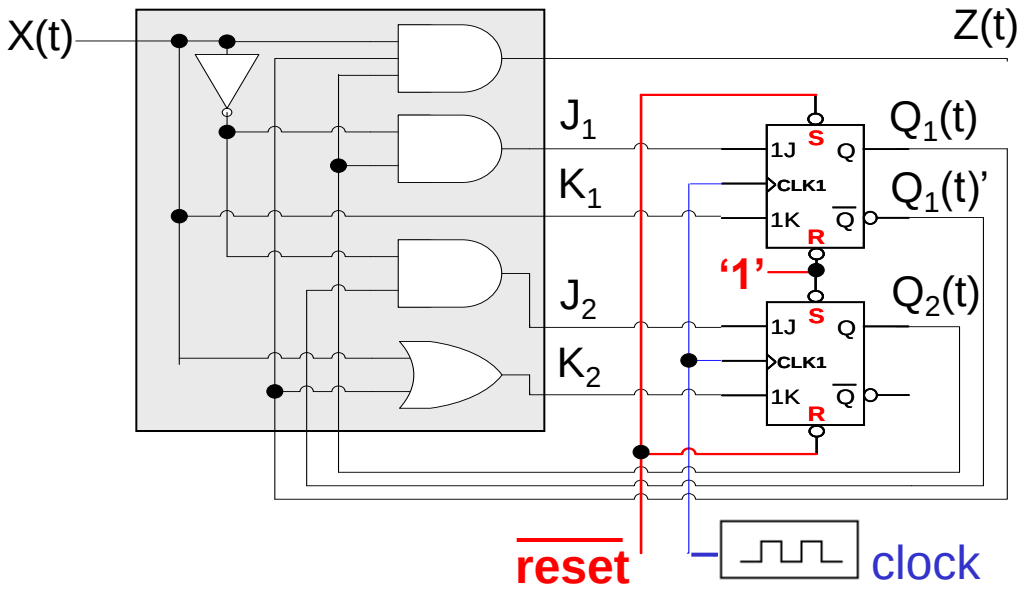
$$Z(t) = X(t) \cdot Q_1(t) \cdot Q_2(t)$$

Step 9: Drawing The Logic Diagram

$$Q_1(t+1)\begin{cases} J_1 = X(t)'\bullet Q_2(t) \\ K_1 = X(t) \end{cases}$$

$$Q_2(t+1)\begin{cases} J_2 = X(t)'\bullet Q_1(t)' \\ K_2 = X(t) + Q_1(t) \end{cases}$$

$$Z(t) = X(t)\bullet Q_1(t) \bullet Q_2(t)$$



- IMPORTANT: Do not forget to connect a ***clock*** signal to the Flip-Flops.
- IMPORTANT: Do not forget to connect a ***reset*** signal to the Flip-Flops:
  - The ***reset*** signal must switch the circuit to the initial state. Every sequential circuit must have one initial state. Always, first switch the circuit to the initial state.
  - To switch to the initial state, use the asynchronous signals *preset (S)* and/or *clear (R)* of the Flip-Flops.
- For our sequence recognizer the initial state is **A** with Q_1Q_2 = **10**, thus:
  - While $\overline{reset}$ is active ($\overline{reset}$ = 0), Q_1 is set to **1** and Q_2 to **0** – the circuit stays in the initial state.
  - While $\overline{reset}$ is non-active ($\overline{reset}$ = 1), the circuit operates normally.

Designing the Same Circuit with *D* Flip-Flops (Step 4)

- What if we want to design the circuit using ***D*** Flip-Flops instead of ***JK***?
- We already have the ***ordered encoded state table***, so we can just start from Step 4, determining the number of Flip-Flops.
- We have 4 states and we have encoded them with 2 bits Q_1 and Q_2. Thus, we need 2 Flip-Flops.
- For each Flip-Flip Q_i, we add one column (for D_i) in the state table.

Input	Present State		Next State		Flip-Flop Inputs		Output
					for Q_1	for Q_2	
X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	D_1	D_2	Z(t)
0	0	0	0	1			0
0	0	1	1	1			0
0	1	0	1	0			0
0	1	1	1	0			0
1	0	0	0	0			0
1	0	1	0	0			0
1	1	0	0	0			0
1	1	1	0	0			1

Finding Flip-Flop Input Values (Step 5)

- Again, for each Flip-Flip Q_i, look at its present and next states, and determine what the input D_i should be in order to make that state change. Use the ***D*** execution table.
- The D excitation table is pretty boring; set the D input to whatever the next state should be.
- You don't even need to show separate columns for D_1 and D_2 ; you can just use the Next State columns.

_D_ Execution Table

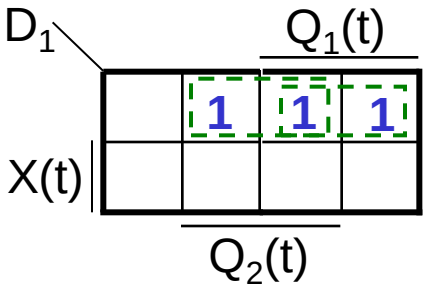
Q(t)	Q(t+1)	D(t)
0	**0**	**0**
0	**1**	**1**
1	**0**	**0**
1	**1**	**1**

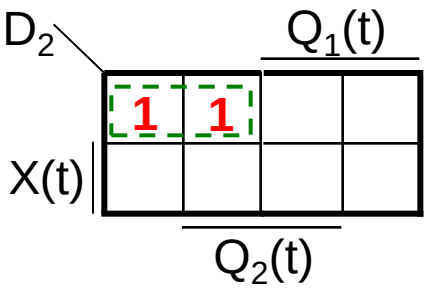
Input	Present State		Next State		Flip-Flop Inputs for Q_1	Flip-Flop Inputs for Q_2	Output
X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	D_1	D_2	Z(t)
0	0	0	0	1	0	1	0
0	0	1	1	1	1	1	0
0	1	0	1	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	1

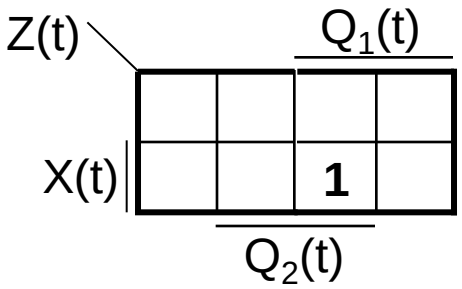
Deriving Simplified Equations (Steps 6, 7, and 8)

- We can use K-maps again, to simplify:

Input	Present State		Next State		Flip-Flop Inputs for Q_1	Flip-Flop Inputs for Q_2	Output
X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	D_1	D_2	Z(t)
0	0	0	0	1	0	1	0
0	0	1	1	1	1	1	0
0	1	0	1	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	1







$D_1 = X(t)' \cdot Q_2(t) + X(t)' \cdot Q_1(t)$ **$D_2 = X(t)' \cdot Q_1(t)'$** **$Z(t) = X(t) \cdot Q_1(t) \cdot Q_2(t)$**

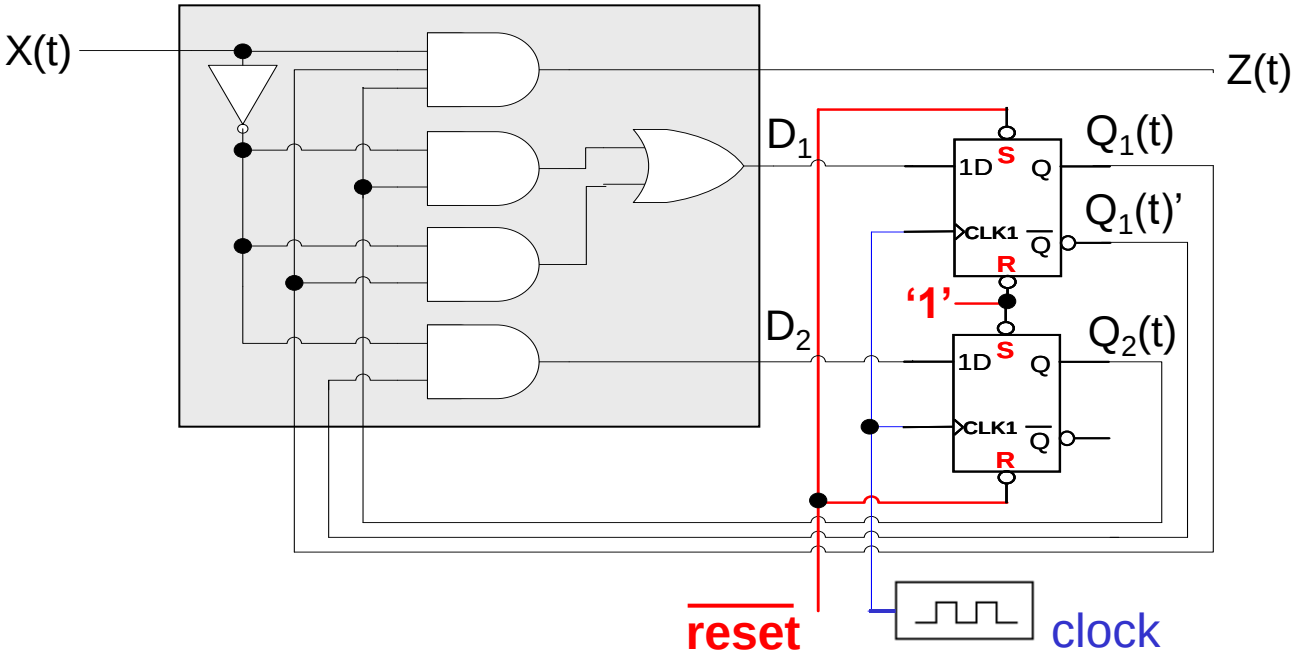
Drawing The Logic Diagram (Step 9)

- Use the equations on the right to draw the logic diagram of the circuit.
- IMPORTANT: Again, do not forget to connect properly the ***clock*** and ***reset*** signals to the Flip-Flops.

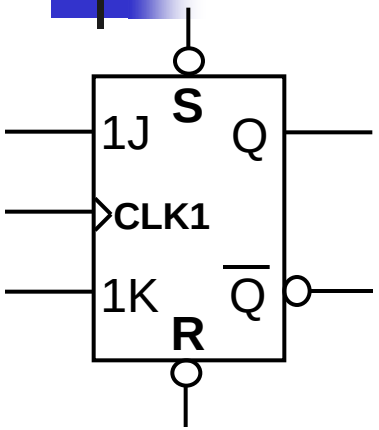
$$Q_1(t+1) = D1 = X(t)' \cdot Q_2(t) + X(t)' \cdot Q_1(t)$$

$$Q_2(t+1) = D2 = X(t)' \cdot Q_1(t)'$$

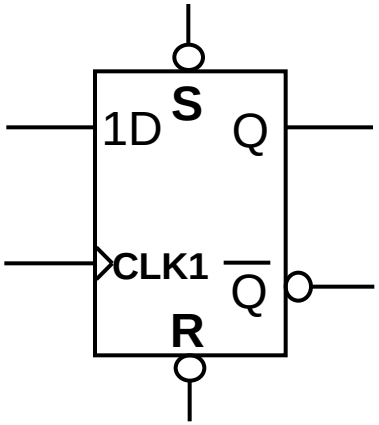
$$Z(t) = X(t) \cdot Q_1(t) \cdot Q_2(t)$$



Design Comparison



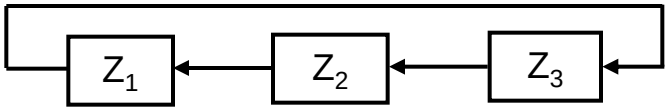
- ***JK*** Flip-Flops are good because there are many ***don't care*** values in the Flip-Flop inputs, which can lead to a simpler circuit.



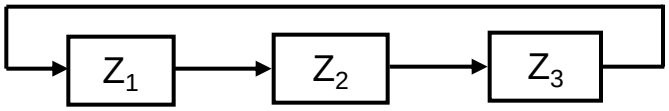
- ***D*** Flip-Flops have the advantage that you do not have to set up Flip-Flop inputs at all, since Q(t+1) = D. However, the D input equations are usually more complex than JK input equations.
- In practice, ***D*** Flip-Flops are used more often.
  - There is only one input for each Flip-Flop, not two.
  - There are no execution tables to worry about.
  - D Flip-Flops can be implemented with slightly less hardware than JK Flip-Flops.

Another Example: A Cyclic Shifter

- Design a circuit which has one input X and three outputs Z_1, Z_2, and Z_3 and the following behavior:
- Initially, the output values are: $Z_1Z_2Z_3 = 010$;
- On every clock cycle:
  - If X = 1 then $Z_1 = Z_2$, $Z_2 = Z_3$, $Z_3 = Z_1$ (cyclic shift-left)



  - If X = 0 then $Z_1 = Z_3$, $Z_2 = Z_1$, $Z_3 = Z_2$ (cyclic shift-right)



- This circuit we will call 3-bit cyclic shifter.

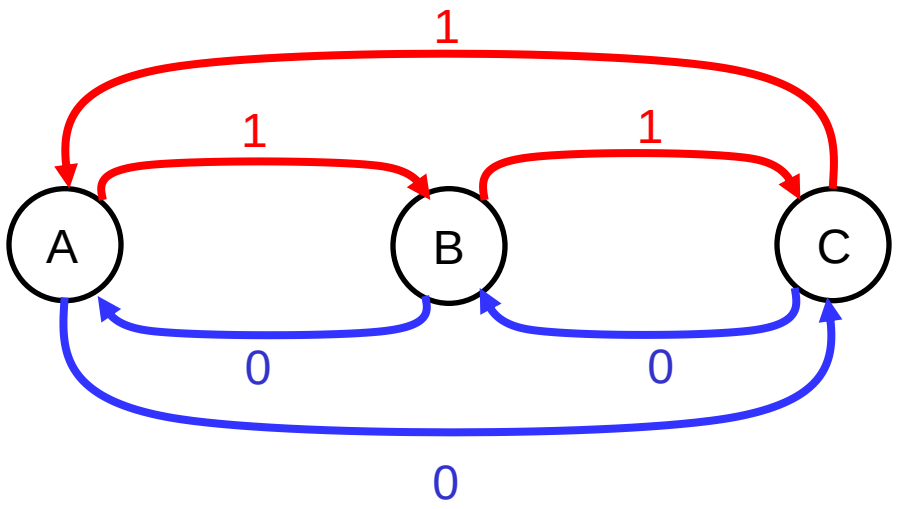
Deriving the State Table

- How many and What states do we need for our circuit?
  - We have initial state where outputs are “010”.
  - If we make cyclic shift-left we can have the following possible output values: “100” or “001” or “010” (which is the initial output value)
  - If we make cyclic shift-right we can have the following possible output values: “001” or “100” or “010” (which is the initial output value)
- We see that we can use only 3 states, one for each of the three distinct output values “010”, “100”, “001”.
- So, for each possible output value we define a state as follows:

State	Meaning
A	Initial state where the outputs $Z_1Z_2Z_3$ = “010”.
B	While at this state the circuit has outputs $Z_1Z_2Z_3$ = “100”.
C	While at this state the circuit has outputs $Z_1Z_2Z_3$ = “001”.

Deriving the State Table (cont.)

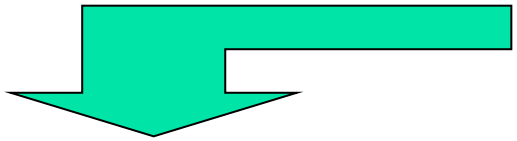
- First, we will derive the state diagram of the circuit.
- We need ***two*** outgoing arrows for each node, to account for the possibilities of X = 0 (cyclic shift-right) and X = 1 (cyclic shift-left).

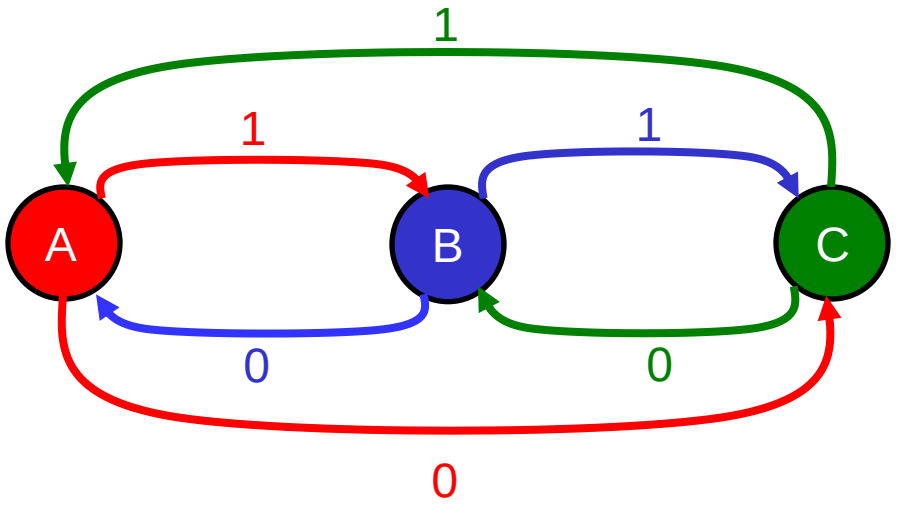


State	Meaning
A	Initial state where the outputs $Z_1Z_2Z_3$ = “010”.
B	While at this state the circuit has outputs $Z_1Z_2Z_3$ = “100”.
C	While at this state the circuit has outputs $Z_1Z_2Z_3$ = “001”.

Deriving the State Table (cont.)

- We have the state diagram and we can derive the state table.





Input X	Present State	Next State	Outputs Z_1	Z_2	Z_3
0	A	C	0	1	0
1	A	B	0	1	0
0	B	A	1	0	0
1	B	C	1	0	0
0	C	B	0	0	1
1	C	A	0	0	1

Remember that we have defined that:

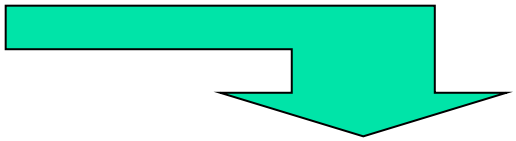
- In state A the circuit outputs $Z_1Z_2Z_3 = 010$
- In state B the circuit outputs $Z_1Z_2Z_3 = 100$
- In state C the circuit outputs $Z_1Z_2Z_3 = 001$

Assigning Binary Codes to States

State Table

Input X	Present State	Next State	Outputs Z_1	Z_2	Z_3
0	A	C	0	1	0
1	A	B	0	1	0
0	B	A	1	0	0
1	B	C	1	0	0
0	C	B	0	0	1
1	C	A	0	0	1

- We will represent state **A** with **00**, **B** with **10**, **C** with **01**.



Encoded State Table

	Input	Present State		Next State		Outputs		
	X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	$Z_1(t)$	$Z_2(t)$	$Z_3(t)$
0	0	0	0	0	1	0	1	0
4	1	0	0	1	0	0	1	0
2	0	1	0	0	0	1	0	0
6	1	1	0	0	1	1	0	0
1	0	0	1	1	0	0	0	1
5	1	0	1	0	0	0	0	1

NOTE: We have to order the encoded state table.

Ordered Encoded State Table

- For this circuit the ***ordered encoded state table*** is not complete. Why?
- We have 3 states that we encode with 2 bits, i.e., state A with $Q_1Q_2 = 00$, B with $Q_1Q_2 = 10$, and C with $Q_1Q_2 = 01$.
- Using two bits we can encode 4 states from which we use only 3 states => one state is unused, i.e., **$Q_1Q_2 = 11$**
- Thus, two rows (**3** and **7**) in the table are incomplete.
- We have two main options to complete the table (see next slides).

	Input	Present State		Next State		Outputs		
	X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	$Z_1(t)$	$Z_2(t)$	$Z_3(t)$
0	0	0	0	0	1	0	1	0
1	0	0	1	1	0	0	0	1
2	0	1	0	0	0	1	0	0
3	0	1	1					
4	1	0	0	1	0	0	1	0
5	1	0	1	0	0	0	0	1
6	1	1	0	0	1	1	0	0
7	1	1	1					

Option 1: Use ***don't-care*** Conditions

- We can use don't-care conditions because if the circuit operates correctly it will never enter unused states.

	Input	Present State		Next State		Outputs		
	X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	$Z_1(t)$	$Z_2(t)$	$Z_3(t)$
0	0	0	0	0	1	0	1	0
1	0	0	1	1	0	0	0	1
2	0	1	0	0	0	1	0	0
3	0	1	1	X	X	X	X	X
4	1	0	0	1	0	0	1	0
5	1	0	1	0	0	0	0	1
6	1	1	0	0	1	1	0	0
7	1	1	1	X	X	X	X	X

- Now, you can apply Steps 4 to 9 of the design procedure to get the complete circuit.
- Do this at home as an exercise!
- Similar example will be given at the tutorials!

Option 2: ***Explicitly*** Specify Unused States

- It is possible that outside interference or a malfunction will cause the circuit to enter one of the unused states causing temporary or permanent incorrect behavior of the circuit.
- Temporary or permanent incorrect behavior may be harmful.
- Thus, in such case it is necessary to explicitly specify, fully or at least partially, the next state values or the output values for the unused states.
  - This will make the behavior of the circuit predictable.
  - Undesired harmful behavior can be avoided if the circuit enters an unused state.

Option 2: ***Explicitly*** Specify Unused States (cont.)

- Depending on the function and application of the circuit, a number of ideas may be applied:
  - First, the outputs for the unused states are specified such that any action that results from entry into and transitions between unused states are not harmful.
  - Second, an unused output combination may be employed which indicates that the circuit has entered an unused (incorrect) state.
  - Third, Next-State for each unused state is selected such that one of the normal occurring states is reached within a few clock cycles, regardless of the input values.
    - Typically, the next state for an unused state is selected to be the initial state.
- The ideas above may be applied in combination.

Option 2: An Example

- Example: Assume that our 3-bit cyclic shifter circuit is used within a system with three other devices:
  - Each output of our circuit controls only one device.
  - Logic '1' on an output enables the corresponding device.
  - A harmful situation for the system will occur if more than one device is enabled.
- So, we can avoid harmful situations by specifying the unused states as shown below (see rows **3** and **7**):

	Input	Present State		Next State		Outputs		
	X(t)	$Q_1(t)$	$Q_2(t)$	$Q_1(t+1)$	$Q_2(t+1)$	$Z_1(t)$	$Z_2(t)$	$Z_3(t)$
0	0	0	0	0	1	0	1	0
1	0	0	1	1	0	0	0	1
2	0	1	0	0	0	1	0	0
3	0	1	1	**0**	**0**	**0**	**0**	**0**
4	1	0	0	1	0	0	1	0
5	1	0	1	0	0	0	0	1
6	1	1	0	0	1	1	0	0
7	1	1	1	**0**	**0**	**0**	**0**	**0**

The output: $Z_1Z_2Z_3$ = **000** is not harmful and indicates that the circuit has entered unused (incorrect) state. Why?

Next-State of the unused state (Q_1Q_2=**11**) is the initial state (Q_1Q_2=**00**)