Sequential Logic and Finite State Machines

Sequential Machines using T,D,SR & JK Flip-flops

Yul Williams, D.Sc.

Goals

- At the conclusion of this session, the student should be able to:
  - Design logic functions using PLDs
  - Understand the concepts of finite state machines
  - Understand the operation of 4 basic flip-flops
  - Design sequential counters using flip-flops
  - Design sequential circuits using Moore and Mealy design methodologies

Outline

- Completing the Fundamentals
- Finite State Machines
- State Machine Types
- FSM Design Concepts

Completing the Fundamentals

Programmable Logic Devices

PLDs

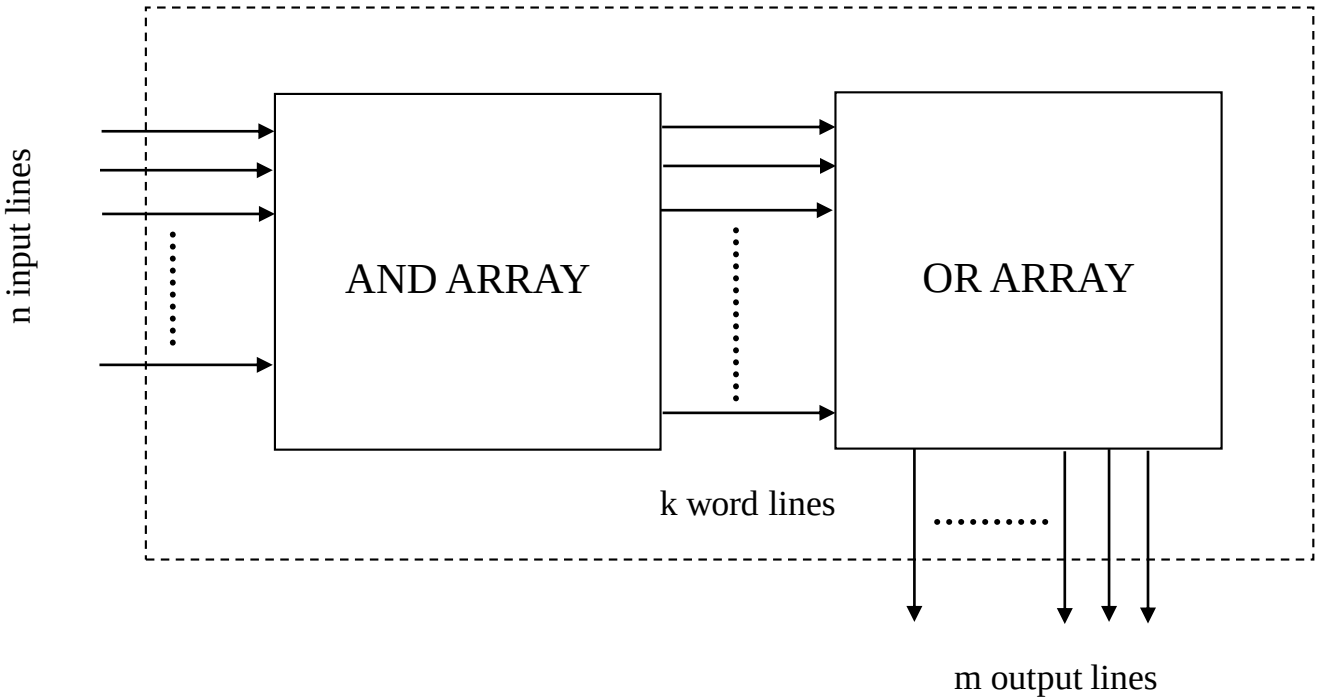
Programmable Logic Devices

- A programmable logic device (PLD) is constructed as an array of generic logic elements
- These elements may be structured (through programming) to perform any desired logic function(s)
- The two basic PLDs are called the Programmable Logic Array (PLA) and the Programmable Array Logic (PAL), respectively

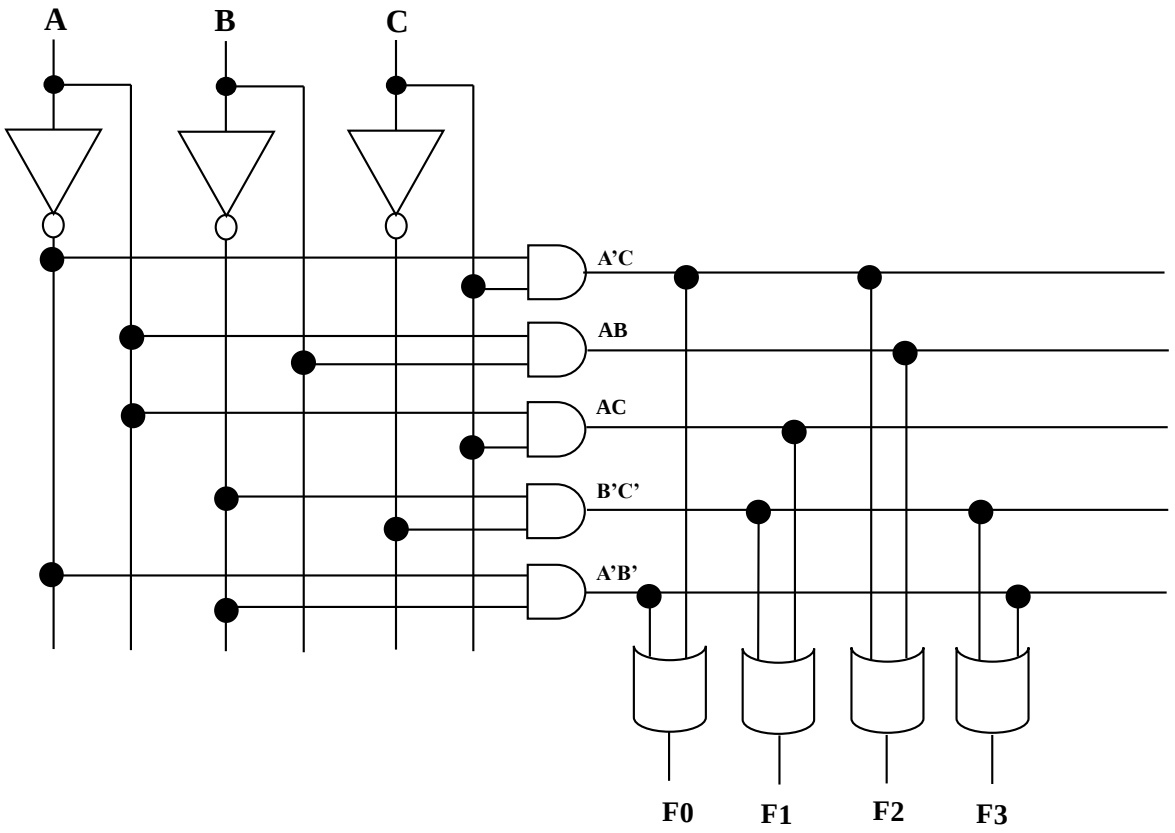
Programmable Logic Array

- The PLA is a device that is structured with one or more AND-OR arrays
- The AND array accepts n input signals and the OR array outputs m number of signals
  - Formally, the PLA can realize m functions of n variables

The PLA



The PLA Unwrapped



F0 = A'B' + A'C

F1 = B'C' + AC

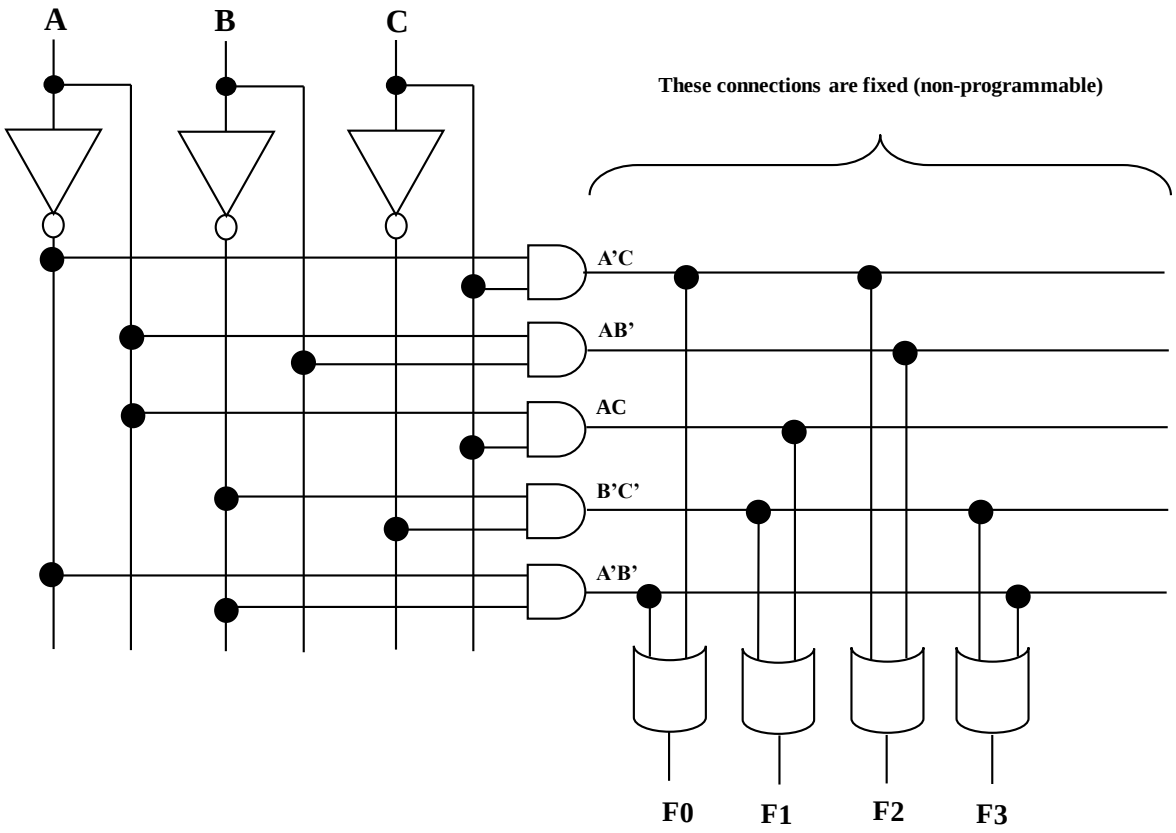
F2 = A'C + AB

F3 = B'C' + A'B'

Programmable Array Logic

- The PAL is very similar (in structure) to the PLA
- It has AND-OR arrays
- There are a couple of subtle differences
  - The OR array in the PAL is fixed
  - Specific AND gates accept input from specific OR gates

The PAL Unwrapped



F0 = A'B' + A'C

F1 = B'C' + AC

F2 = A'C + AB

F3 = B'C' + A'B'

Programming PLDs

- PLDs are programmed with low-level languages
- Examples of these languages include PALASM, ABEL and CUPL
- These programs written in these languages consist mostly of equations that are compiled into a “fuse map” file to configure the PLD

Basic Sequential Components

Flip-flops

Sequential Logic

- Sequential logic is different that the combinational logic circuits that we have been studying
- The operation of sequential logic elements are driven by a clock signal
- Each pulse of a clock the sequential circuit may change its state or remain in its current state. That is dependent on the function of the circuit.

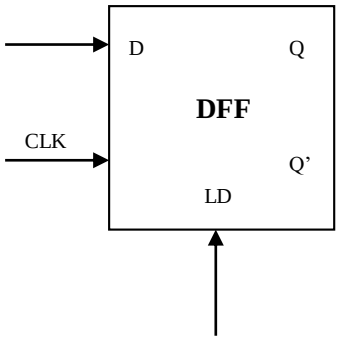
The Flip-Flop

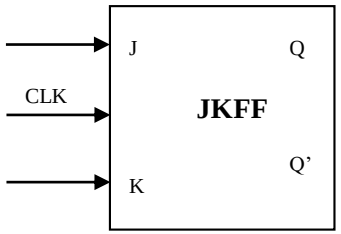
- The flip-flop (FF) is the basic sequential component.
- The FF stores a single bit of data. It makes that data available to other components
- The simplest type of FF is the D flip-flop. It is sometimes referred to as the D-latch.

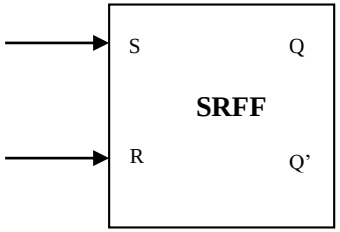
The Four Basic Flip-Flop Types

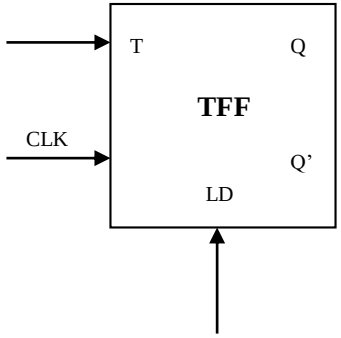
- D flip-flop
- Set-Reset (SR) flip-flop
- J-K flip-flop
- Toggle (T) flip-flop

Flip-Flops









Finite State Machines

Finite State Machine (FSM)

- A finite state machine models the behavior of a sequential system
- In this class session, we will learn all of the steps involved in creating a FSM.
  - We will use flip-flops as the major component to implement our design
- There are several steps involved in constructing a FSM

What is a FSM?

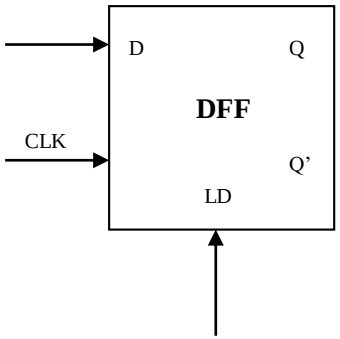
- A FSM consists of several states.
- The states are viewed in two forms
  - The present state and the next state
- The FSM is driven by a set of inputs
  - The inputs determine if the FSM stays in its present state or moves to its next state
- The FSM may produce output values when it transitions to specific states

Making a FSM

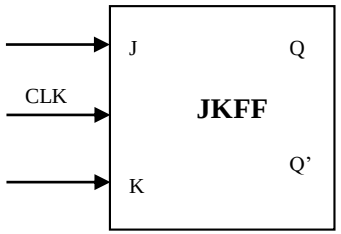
- Now that we know the basic functional characteristics of a FSM, we may begin defining the process by which they are constructed.

Flip-Flop Characteristic Equations

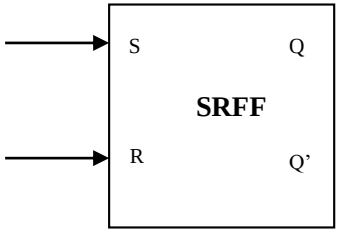
$Q^+ = D$



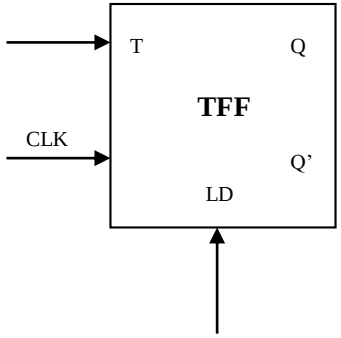
$Q^+ = JQ' + K'Q$



$Q^+ = S + R'Q$



$Q^+ = T \oplus Q$



Take the time to learn these 4 characteristic equations. They are an essential element in the design and realization of sequential logic.

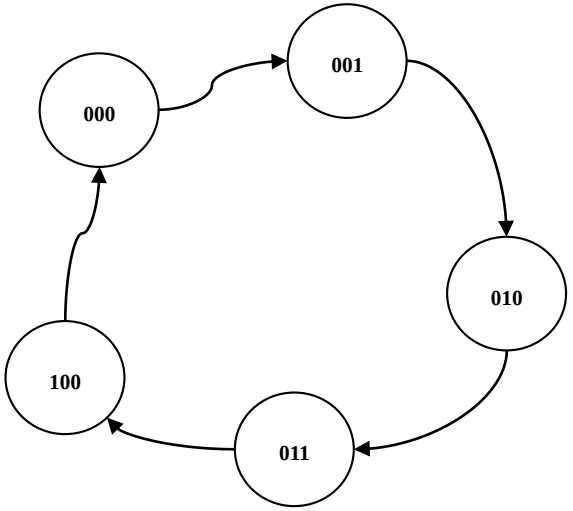
An Example Sequential Design

- Over the next few minutes, we will use the knowledge of Boolean algebra, truth tables, combinational logic layout, K-maps and flip-flops to create a finite state machine (FSM)
- As we go through the design, we will learn the process of constructing a FSM composed of sequential and combinational logic elements

The State Graph

- Each time we want to implement a FSM, it is important to illustrate its planned behavior
- The behavior of the FSM may be drawn as a graph
- The graph depicts the various states of the FSM and details the conditions when the FSM changes states and produces outputs

An Example of a State Graph



The state graph is a directed graph consisting of two major components. These components are the arc and the vertex. The arc represents the transitioning from one vertex to the next. The vertex indicates the state of the machine at any specific instance. This state graph indicates a counting sequence. That means, it is showing the behavior of a counter!

The State Table

- Once we have completed the state graph, we move to the next step toward creating the logic circuit
- That step is the construction of the state table
- The state table translates the state graph into a tabular form to determine a number of design elements
  - Number of variables required
  - Listing of unspecified states

State Table Example

Present State	Next State
A B C	**A^+ B^+ C^+**
000	**001**
001	**010**
010	**011**
011	**100**
100	**000**

Notice the sequence of states. The variable ABC indicate each of the states detailed in the state graph. Given what we know about the number of variables (i.e. 3) used to represent the inputs, we "could" represent 8 possible states. In this particular case, however, only 5 states are specified. In addition, look at the next state column of the state table, it follows the behavior of the state graph. If the present state is 000 the next state is 001. This means that when the FSM leaves state 000 it must transition to state 001.

Implementing our Counter

- In order to realize your logic design for this counter, we must select our sequential components
- Currently, we have 4 from which to select
  - DFF, SRFF, JKFF and the TFF
- Let's choose the TFF to implement our counter design

Recording the State Transitions

Present State	Next State	TA Input	TB Input	TC Input
ABC	A+B+ C+	TA	TB	TC
000	001	0	0	1
001	010	0	1	1
010	011	0	0	1
011	100	1	1	1
100	000	1	0	0

In order to determine the input values for the TFF's (TA, TB and TC), we use the characteristic equations for the TFF. Examine the present state values of A,B &C against the next state values of TA, TB, &TC, respectively. Since Q+ = T XOR Q, whenever a present state value differs from its next state value, the corresponding T input receives a 1. Otherwise, it receives a 0.

Deriving the Logic

- Now that we have the transitions recorded in the state table, we must now derive the input logic for the 3 TFFs
- This is done using our K-map skills

Deriving the TFF Inputs

Present State	Next State	TA Input	TB Input	TC Input
ABC	A+B+ C+	TA	TB	TC
000	001	0	0	1
001	010	0	1	1
010	011	0	0	1
011	100	1	1	1
100	000	1	0	0

T_{A}-Map

BC \ A	0	1
00	0	1
01	0	X
11	1	X
10	0	X

$T_A = A + BC$

T_{B}-Map

BC \ A	0	1
00	0	0
01	1	X
11	1	X
10	0	X

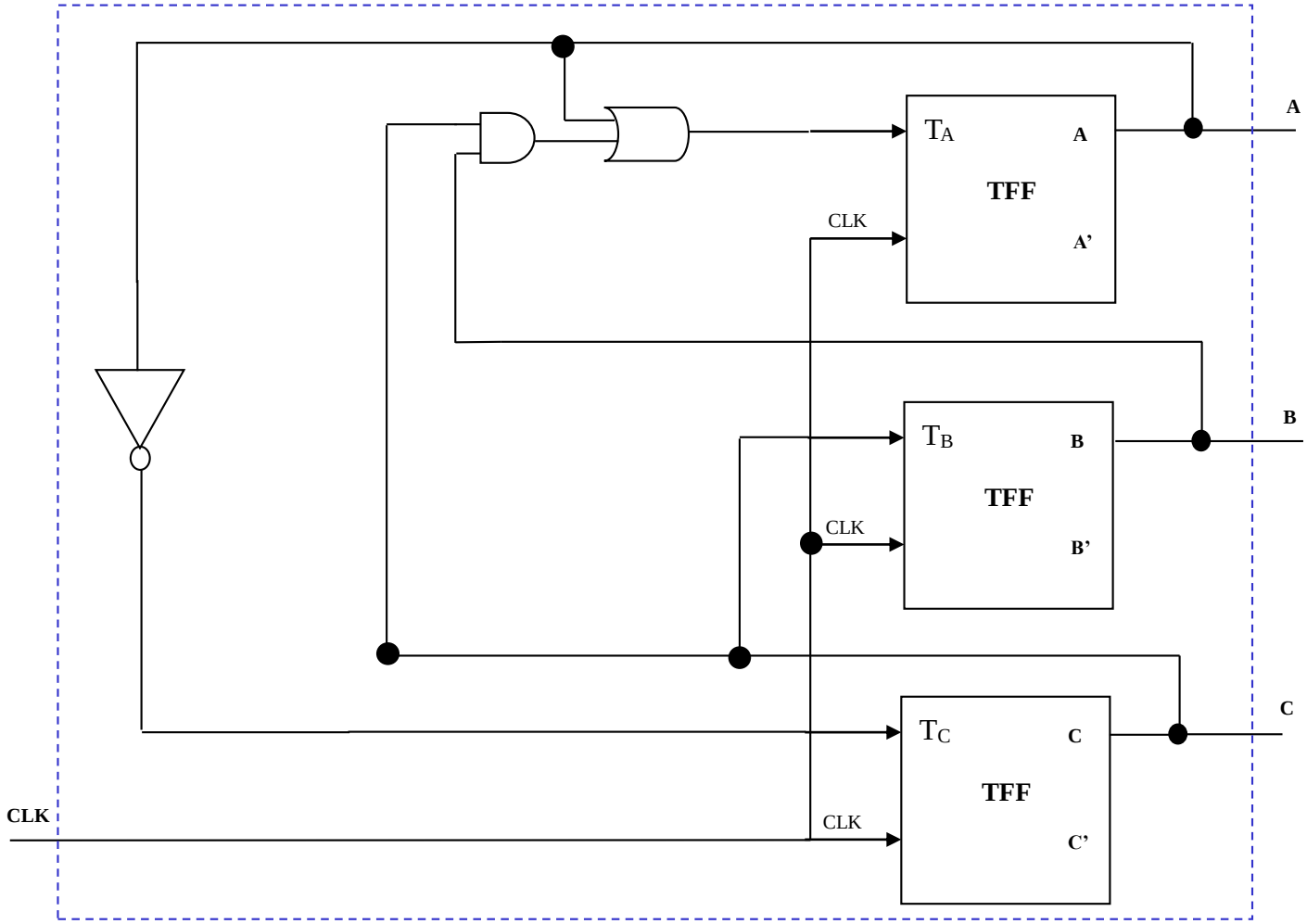
$T_B = C$

T_{C}-Map

BC \ A	0	1
00	1	0
01	1	X
11	1	X
10	1	X

$T_C = A'$

Realizing the Design



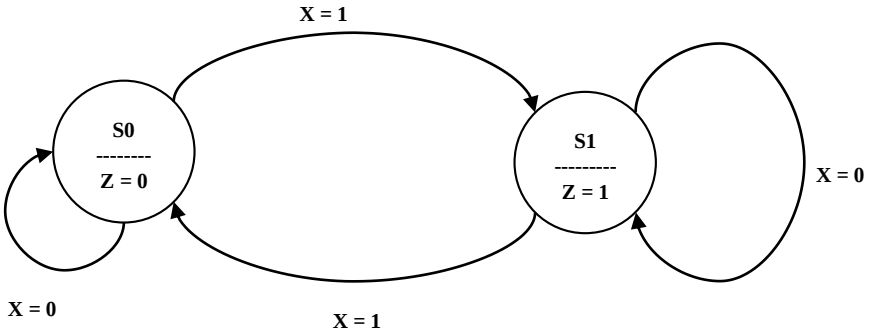
State Machine Types

Moore and Mealy Machines

The Moore Machine

- The Moore machine was named after Edward Moore
- It has the characteristic of associating its outputs with the states
  - The outputs are represented within the vertex or in close proximity to the vertex

Moore Machine State Graph and State Table



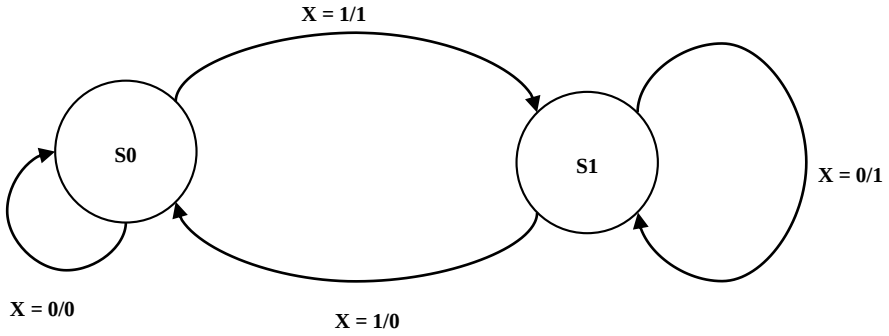
Present State	Next State		Z
	X = 0	X = 1	
S0	S0	S1	0
S1	S1	S0	1

Notice the vertices. The state name is shown along with the output value Z. The Moore machine state graph is always represented in this fashion. Notice that the output Z is not dependent on the input X.

The Mealy Machine

- The Mealy machine was named after George Mealy
- It has the characteristic of associating its outputs with the arcs
  - The outputs are represented within the arcs or in close proximity to the arc

Mealy Machine State Graph and State Table



Present State	Next State		Z	
	X = 0	X = 1	X = 0	X = 1
S0	S0	S1	0	1
S1	S1	S0	1	0

FSM Design Concepts

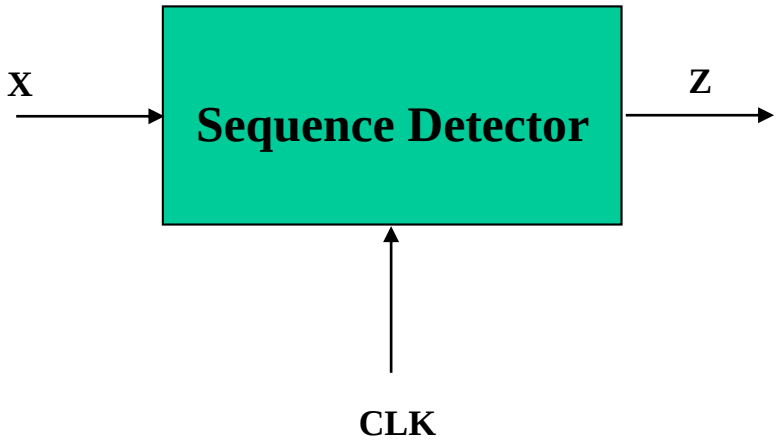
A Sequence Detector

Mealy and Moore Examples

What's a Sequence Detector, Yul?

- A sequence detector is a FSM that produces a logic 1 when a specific binary sequence has been observed
- When the FSM does not observe the sequence in the observed binary sequence, it emits a logic 0
- Let's get started!

Macro View of the Sequence Detector



This sequence detector will be designed to recognize the pattern "101". The behavior of the machine calls for the Z output to equal 1 whenever the programmed pattern is observed in the input bit stream X.

Example:

X = 0011011001010100

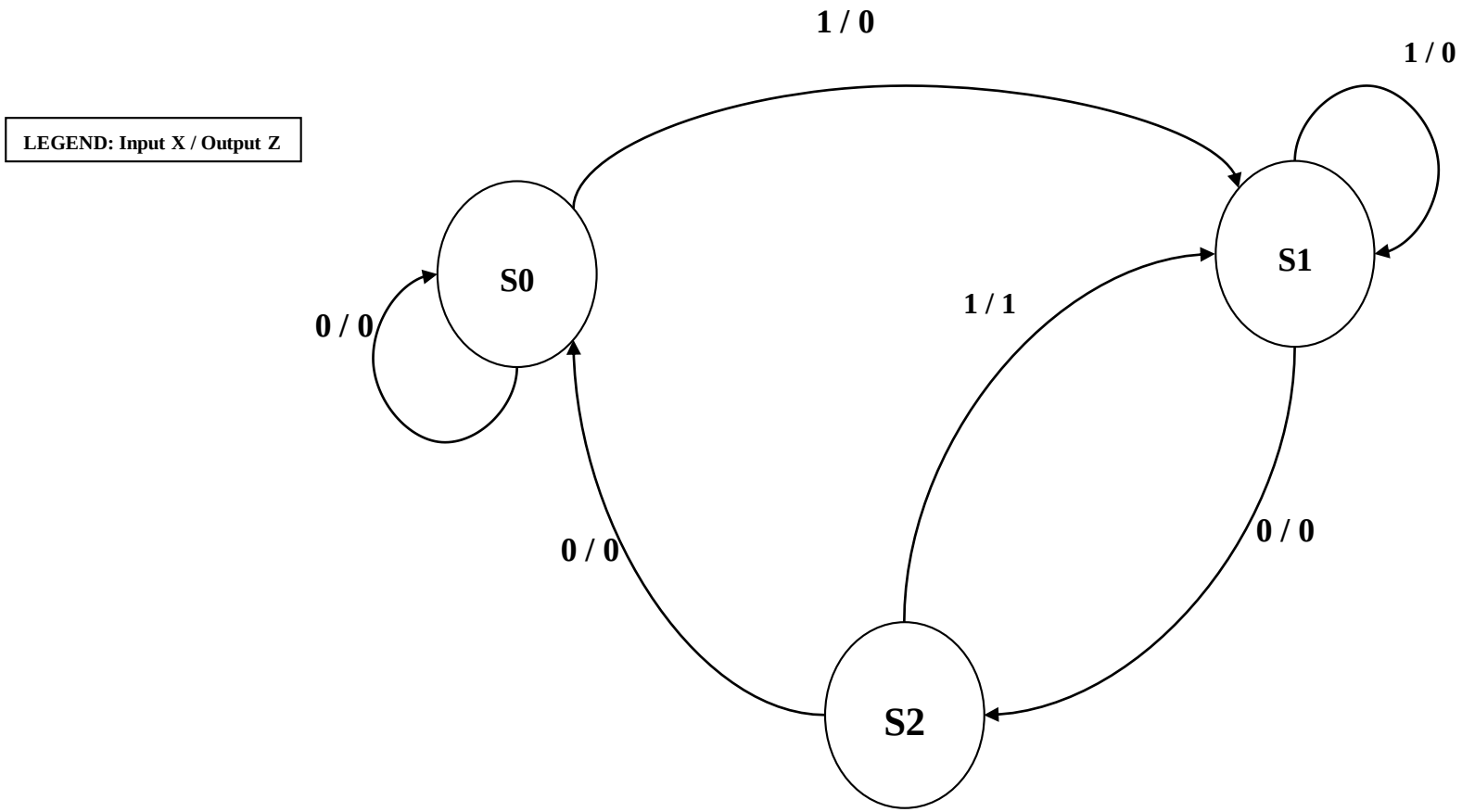
Z = 0000010000010100

Source: Fundamentals of Logic Design by Charles H. Roth

Design Strategy

- For the design of the sequence detector, we will select the Mealy machine model
- For this design, we will use the following process:
    1. Generate the state graph
    2. Create the state table
    3. Create the state transition table
    4. Generate the input expressions for the JKFF
    5. Realize the final logic design

Generate the State Graph



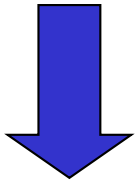
Create the state table

Present State	Next State		Z	
	X = 0	X = 1	X = 0	X = 1
S0	S0	S1	0	0
S1	S2	S1	0	0
S2	S0	S1	0	1

Create the State Transition Table

State Table

Present State	Next State		Z	
	X = 0	X = 1	X = 0	X = 1
S0	S0	S1	0	0
S1	S2	S1	0	0
S2	S0	S1	0	1



Let S0 = 00

S1 = 01

S2 = 10

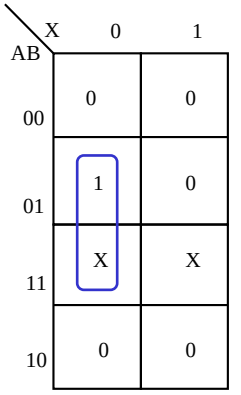
State Transition Table

Present State	Next State		Z	
	X = 0	X = 1		
AB	A+B+	A+B+	X = 0	X = 1
00	00	01	0	0
01	10	01	0	0
10	00	01	0	1

Generate the Input Expressions for the JKFF

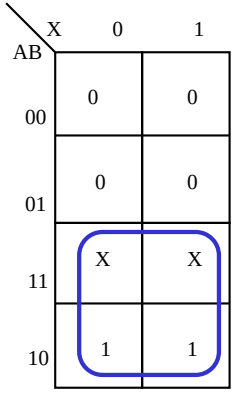
Present State	Next State		Z	
	X = 0	X = 1		
AB	A+B+	A+B+	X = 0	X = 1
00	00	01	0	0
01	10	01	0	0
10	00	01	0	1

J_{A}-Map



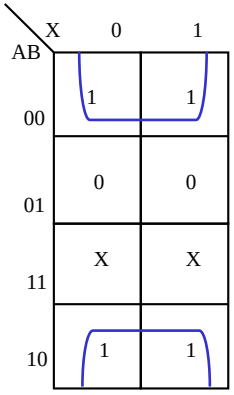
JA = X'B

K_{A}-Map



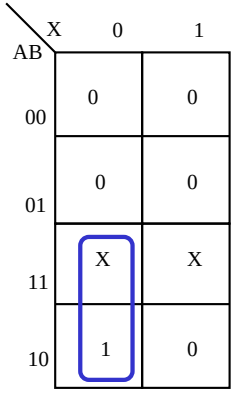
KA = A

J_{B}-Map



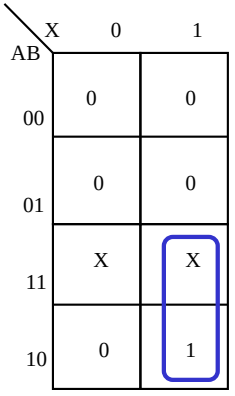
JB = B'

K_{B}-Map



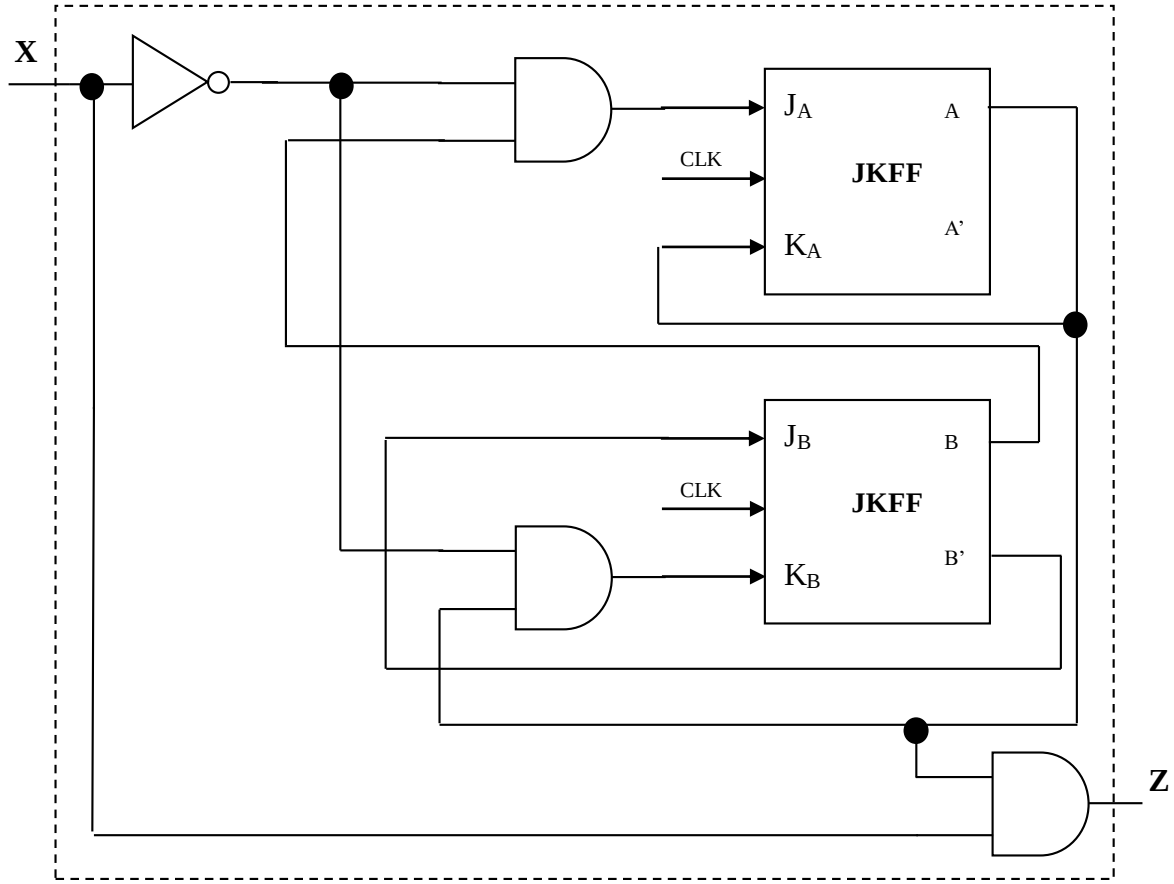
KB = X'A

Z-Map



Z = XA

Realize the final logic design



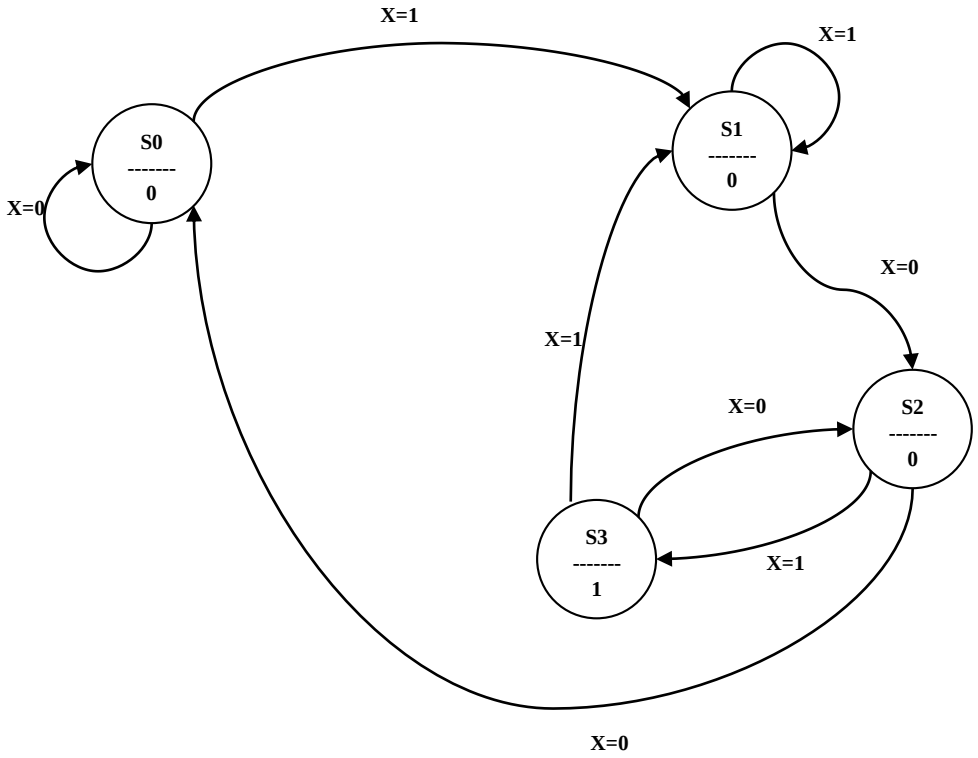
Sequence Detector based on the Mealy Machine model

Design Strategy

- For the design of the sequence detector, we will select the Moore machine model
- For this design, we will use the SAME process as before:
  - Don’t change the strategy
  - Put the pencil on the table and …step away from the strategy!!!
- Let’s move on…

Generate the State Graph

LEGEND:	State
	Output Z



Note: The Moore machine graph has one more state than the Mealy machine implementation. Since the output is represented within a state, an additional state must be created because it is the only state that can show an output of '1' for this FSM.

Create the state table

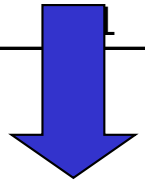
Present State	Next State		Z
	X = 0	X = 1	
S0	S0	S1	0
S1	S2	S1	0
S2	S0	S3	0
S3	S2	S1	1

Notice that the Moore machine model has an output value Z that is not dependent on the value of X.

Create the State Transition Table

State Table

Present State	Next State		Z
	X = 0	X = 1	
S0	S0	S1	0
S1	S2	S1	0
S2	S0	S3	0
S3	S2	1	1



Let S0 = 00
S1 = 01
S2 = 10
S3 = 11

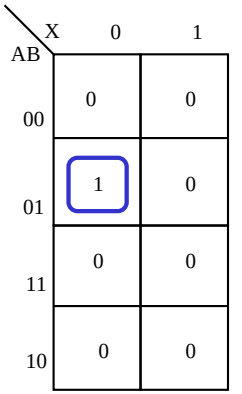
State Transition Table

Present State	Next State		Z
AB	X = 0	X = 1	
00	00	01	0
01	10	01	0
10	00	11	0
11	10	01	1

Generate the Input Expressions for the JKFF

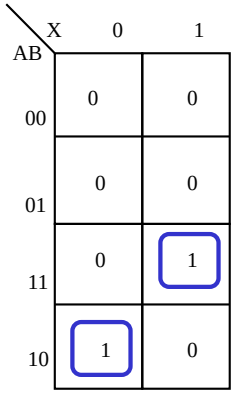
Present State	Next State		Z
AB	X = 0	X = 1	
00	00	01	0
01	10	01	0
10	00	11	0
11	10	01	1

J_{A}-Map



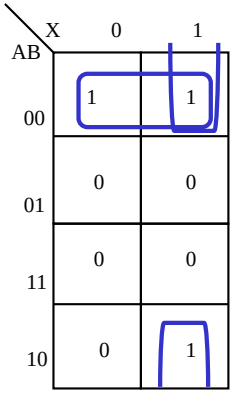
$J_A = X'A'B$

K_{A}-Map



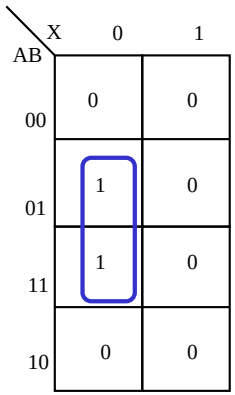
$K_A = X'AB'+XAB$

J_{B}-Map



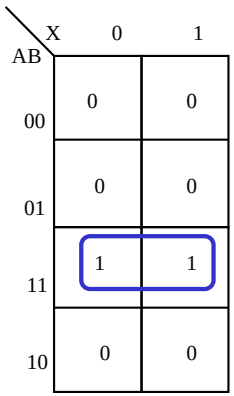
$J_B = A'B' + XB'$

K_{B}-Map



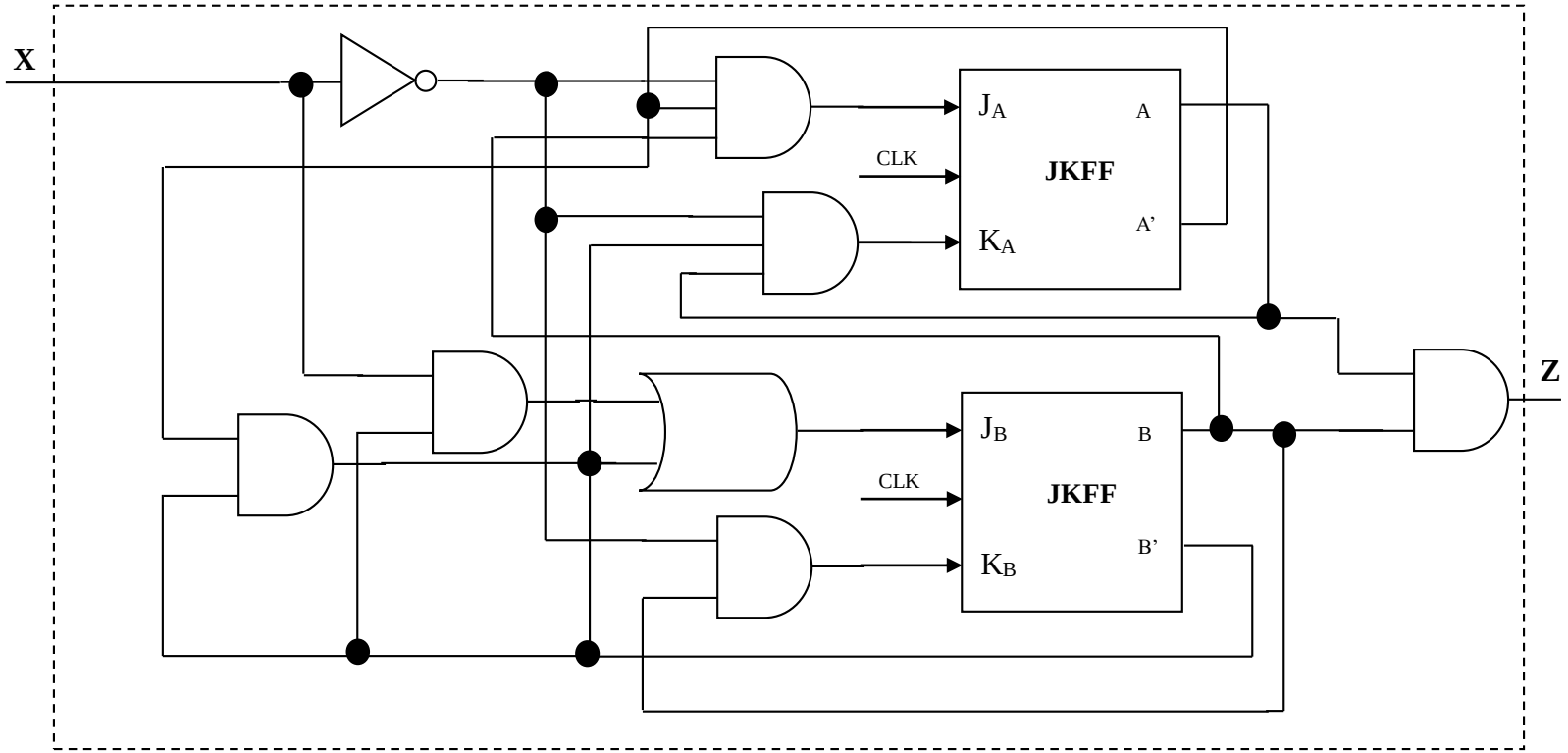
$K_B = X'B$

Z-Map



$Z = AB$

Realize the final logic design



Sequence Detector based on the Moore Machine model

Practice Questions

1. Design a PLA circuit that can produce the following outputs:

F0 = AB'C' + A'C

F1 = BC + A'C'

F2 = A + B'C

F3 = AB + A'B'C'

2. Draw the Moore and Mealy state graphs for the sequence generator that recognizes the sequence 1011.

3. Name two programming languages for customizing PLDs.