# Synchronous Sequential Logic

Chapter 5

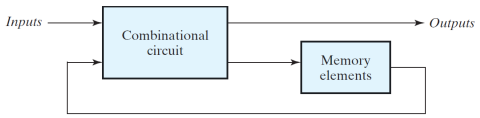
Steve Oldridge
Dr. Sidney Fels

## Topics

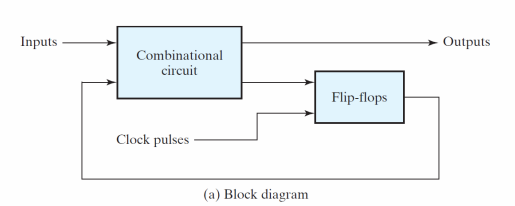
- Sequential Circuits
  - What happens when we add memory?
- Latches
- Flip-Flops
- Clocks
- State Tables
  - Reduction of States



## Sequential Circuits



## Synchronous Sequential Circuits




(a) Block diagram

(b) Timing diagram of clock pulses

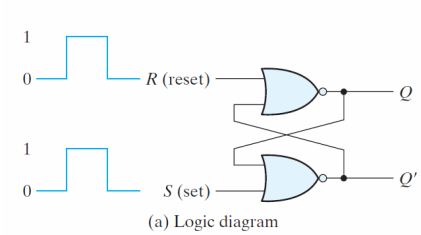


## Latches

- Maintains a State
  - Output is the same as long as there is power
  - Memory!
- SR Latch
  - NOR
  - NAND
- D Latch

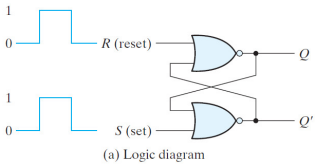


## *SR* Latch




(a) Logic diagram

# *SR* Latch Truth Table



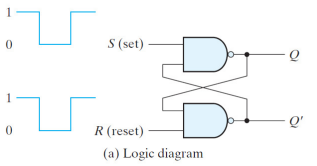

(a) Logic diagram

| *S* | *R* | *Q* | *Q′* | |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | |
| 0 | 0 | 1 | 0 | (after $S = 1, R = 0$) |
| 0 | 1 | 0 | 1 | |
| 0 | 0 | 0 | 1 | (after $S = 0, R = 1$) |
| 1 | 1 | 0 | 0 | (forbidden) |

(b) Function table



# *SR* Latch Truth Table (NAND edition)




(a) Logic diagram

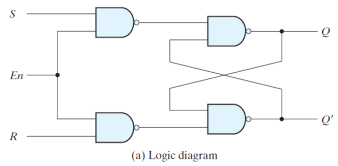
| *S* | *R* | *Q* | *Q′* | |
|---|---|---|---|---|
| 1 | 0 | 0 | 1 | |
| 1 | 1 | 0 | 1 | (after $S = 1, R = 0$) |
| 0 | 1 | 1 | 0 | |
| 1 | 1 | 1 | 0 | (after $S = 0, R = 1$) |
| 0 | 0 | 1 | 1 | (forbidden) |

(b) Function table



# Controlling *SR* Latches

- We don't always want the circuit to change immediately




(a) Logic diagram

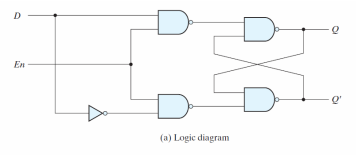
| *En* | *S* | *R* | Next state of *Q* |
|---|---|---|---|
| 0 | X | X | No change |
| 1 | 0 | 0 | No change |
| 1 | 0 | 1 | $Q = 0$; reset state |
| 1 | 1 | 0 | $Q = 1$; set state |
| 1 | 1 | 1 | Indeterminate |

(b) Function table

- What was the problem with latches?
  - If set and reset are both active…
  - Q = Q` which is impossible
- How can we overcome that?
  - R = S`



# D Latch (Transparent Latch)



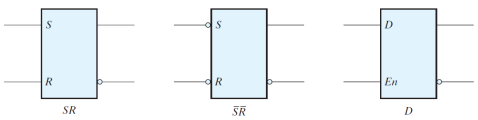

(a) Logic diagram

| En | D | Next state of Q |
|---|---|---|
| 0 | X | No change |
| 1 | 0 | Q = 0; reset state |
| 1 | 1 | Q = 1; set state |

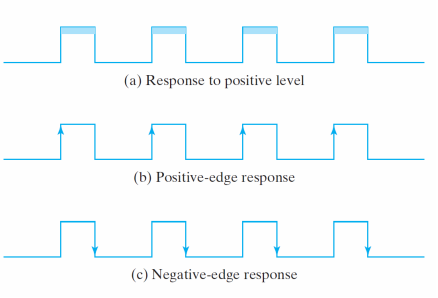
(b) Function table



# Symbols for Latches



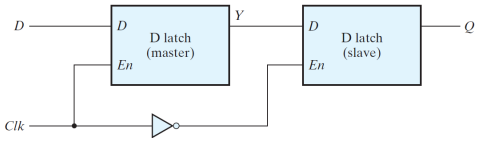
# Clocks & Edge Triggers




- Latch
- FlipFlop

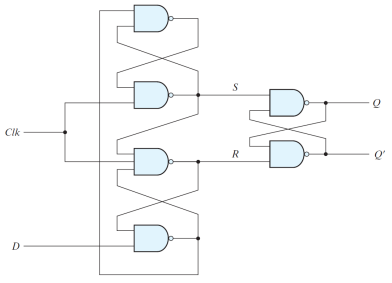


# Edge triggered D Flip-Flop

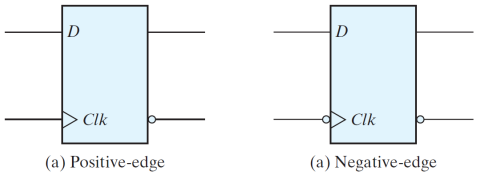





# Positive Edge Trigger



# D Flip-Flop




(a) Positive-edge (a) Negative-edge

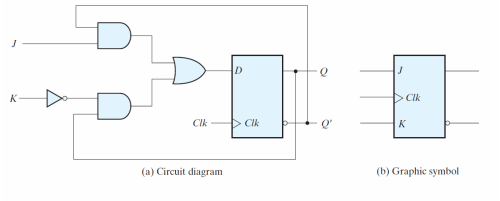


# Other Flip-Flops?

- What is the main function of a flip-flop?
  - Store & output a value
  - Change only on an edge
- D flip-flop is most economical (least # of gates)
- Additional functionality can be useful



# JK Flip Flop



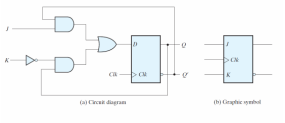

(a) Circuit diagram (b) Graphic symbol

# JK Flip-Flop

*Flip-Flop Characteristic Tables*

**JK Flip-Flop**

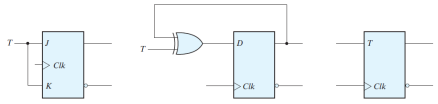
| J | K | Q(t + 1) | |
|---|---|---|---|
| 0 | 0 | $Q(t)$ | No change |
| 0 | 1 | 0 | Reset |
| 1 | 0 | 1 | Set |
| 1 | 1 | $Q'(t)$ | Complement |



(a) Circuit diagram (b) Graphic symbol



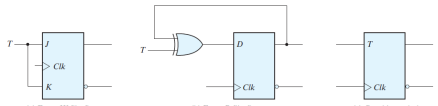
# T Flip-Flop



(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol



# T Flip-Flop



(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol

**T Flip-Flop**

| T | Q(t + 1) | |
|---|---|---|
| 0 | $Q(t)$ | No change |
| 1 | $Q'(t)$ | Complement |

# Characteristic Tables

**JK Flip-Flop**

| J | K | Q(t + 1) | |
|---|---|---|---|
| 0 | 0 | $Q(t)$ | No change |
| 0 | 1 | 0 | Reset |
| 1 | 0 | 1 | Set |
| 1 | 1 | $Q'(t)$ | Complement |

**D Flip-Flop**

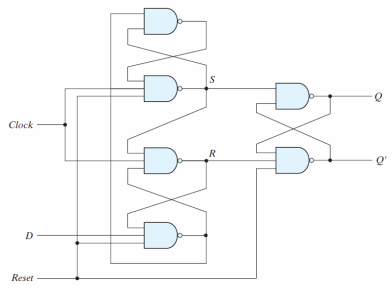
| D | Q(t + 1) | |
|---|---|---|
| 0 | 0 | Reset |
| 1 | 1 | Set |

**T Flip-Flop**

| T | Q(t + 1) | |
|---|---|---|
| 0 | $Q(t)$ | No change |
| 1 | $Q'(t)$ | Complement |

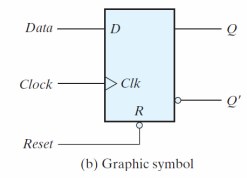


# Adding an Asynchronous Reset






# Adding an Asynchronous Reset

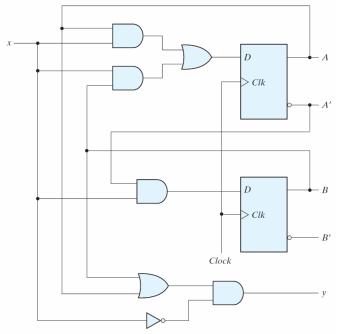



(b) Graphic symbol

| R | Clk | D | Q | Q' |
|---|---|---|---|---|
| 0 | X | X | 0 | 1 |
| 1 | ↑ | 0 | 0 | 1 |
| 1 | ↑ | 1 | 1 | 0 |

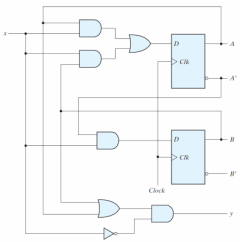
(b) Function table

## Analysis of Sequential Circuits






## Analysis of Sequential Circuits



- A(t+1) = A(t)x(t) + B(t)x(t)
- B(t+1) = A'(t)x(t)
- Y = (A+B)x'



## State Table

| Present State | | Input | Next State | | Output |
|---|---|---|---|---|---|
| *A* | *B* | *x* | *A* | *B* | *y* |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 |

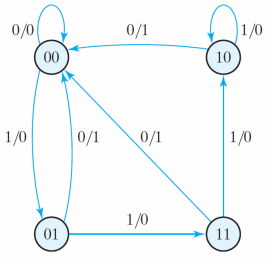
# State Table

*Second Form of the State Table*

| Present State | | Next State | | | | Output | |
|---|---|---|---|---|---|---|---|
| | | $x = 0$ | | $x = 1$ | | $x = 0$ | $x = 1$ |
| $A$ | $B$ | $A$ | $B$ | $A$ | $B$ | $y$ | $y$ |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |



# State Diagram

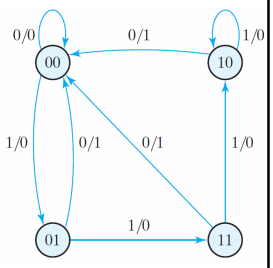




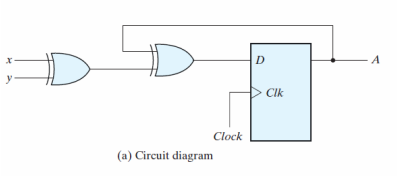

# State Diagram

*Second Form of the State Table*

| Present State | | Next State | | | | Output | |
|---|---|---|---|---|---|---|---|
| | | $x = 0$ | | $x = 1$ | | $x = 0$ | $x = 1$ |
| $A$ | $B$ | $A$ | $B$ | $A$ | $B$ | $y$ | $y$ |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |



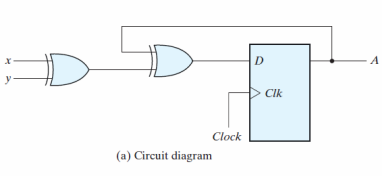
# Sequential Circuit Example




(a) Circuit diagram



# Sequential Circuit Example



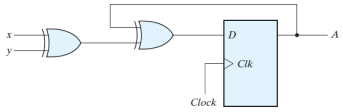

(a) Circuit diagram

| Present state | Inputs | | Next state |
|---|---|---|---|
| *A* | *x* | *y* | *A* |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

(b) State table



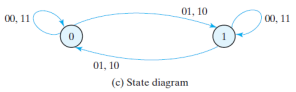
# Sequential Circuit Example




(a) Circuit diagram

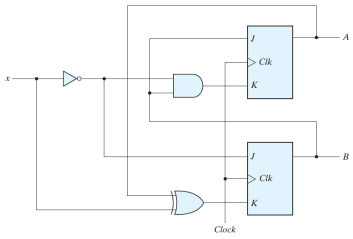
| Present state | Inputs | | Next state |
|---|---|---|---|
| *A* | *x* | *y* | *A* |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

(b) State table




(c) State diagram

# Analysis with JK FFs



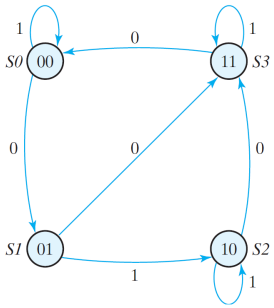



# Analysis with JK FFs

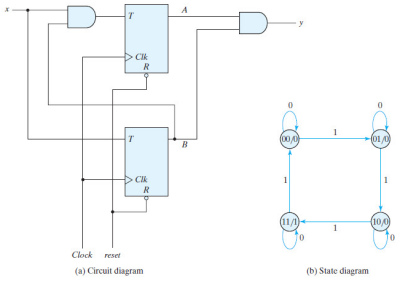
| Present State | | Input | Next State | | Flip-Flop Inputs | | | |
|---|---|---|---|---|---|---|---|---|
| *A* | *B* | *x* | *A* | *B* | $J_A$ | $K_A$ | $J_B$ | $K_B$ |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |



# Analysis with JK FFs



# Analysis with T flip flops




(a) Circuit diagram (b) State diagram

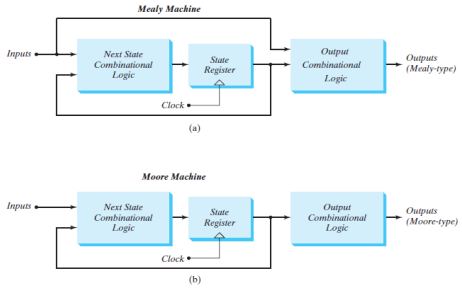


# Analysis with T flip flops

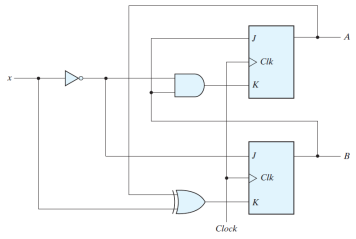
| Present State | | Input | Next State | | Output |
|---|---|---|---|---|---|
| *A* | *B* | *x* | *A* | *B* | *y* |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 |



# Mealy Machines and Moore Machines

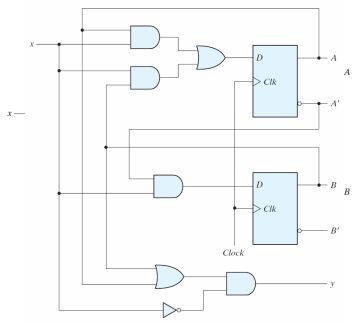


## Mealy or Moore?






## Mealy or Moore?






## How do hardware developers create circuits?

- Visual Layout
  - Draw the circuits by connecting components
  - Not practical for large scale or VLSI
- Verilog / VHDL
  - Describe the circuit's functionality in code
  - Run it through a 'compiler'
  - Output is a circuit
  - Language details are in your text

## How do hardware developers create circuits?

- Verilog / VHDL
  - Describe the circuit's functionality in code
  - Run it through a 'compiler'
  - Output is a circuit compiled / synthesized onto / into hardware
    - FPGA
    - Custom ASIC
  - Language details are in your text

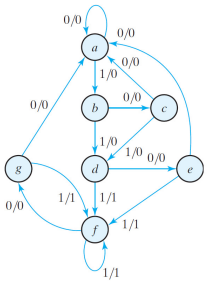


## Verilog

- Behavioral Modeling
  - What is the circuit supposed to do?
  - Describe Inputs / Outputs / States
- Structural Modeling
  - What blocks make up the circuit?
  - How are they connected?



## State Reduction



## Examine the State Table

| Present State | Next State $x = 0$ | Next State $x = 1$ | Output $x = 0$ | Output $x = 1$ |
|---|---|---|---|---|
| $a$ | $a$ | $b$ | 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 |



## Reducing

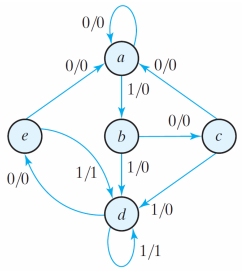
| Present State | Next State $x = 0$ | Next State $x = 1$ | Output $x = 0$ | Output $x = 1$ |
|---|---|---|---|---|
| $a$ | $a$ | $b$ | 0 | 0 |
| $b$ | $c$ | $d$ | 0 | 0 |
| $c$ | $a$ | $d$ | 0 | 0 |
| $d$ | $e$ | $f$ | 0 | 1 |
| $e$ | $a$ | $f$ | 0 | 1 |
| $f$ | $e$ | $f$ | 0 | 1 |



## Reducing Again

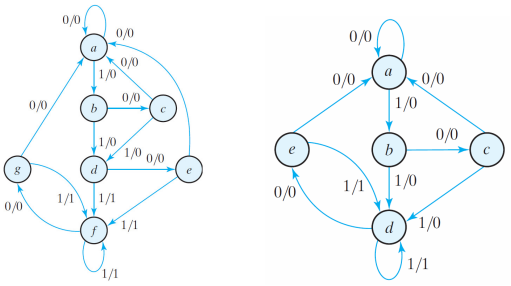
| Present State | Next State $x = 0$ | Next State $x = 1$ | Output $x = 0$ | Output $x = 1$ |
|---|---|---|---|---|
| $a$ | $a$ | $b$ | 0 | 0 |
| $b$ | $c$ | $d$ | 0 | 0 |
| $c$ | $a$ | $d$ | 0 | 0 |
| $d$ | $e$ | $d$ | 0 | 1 |
| $e$ | $a$ | $d$ | 0 | 1 |

## Our new state diagram






## Comparing…






## State Assignment

| State | Assignment 1, Binary | Assignment 2, Gray Code | Assignment 3, One-Hot |
|---|---|---|---|
| *a* | 000 | 000 | 00001 |
| *b* | 001 | 001 | 00010 |
| *c* | 010 | 011 | 00100 |
| *d* | 011 | 010 | 01000 |
| *e* | 100 | 110 | 10000 |

## Applying Binary Assignment to our Example

*Reduced State Table with Binary Assignment 1*

| | Next State | | Output | |
|---|---|---|---|---|
| **Present State** | $x = 0$ | $x = 1$ | $x = 0$ | $x = 1$ |
| 000 | 000 | 001 | 0 | 0 |
| 001 | 010 | 011 | 0 | 0 |
| 010 | 000 | 011 | 0 | 0 |
| 011 | 100 | 011 | 0 | 1 |
| 100 | 000 | 011 | 0 | 1 |

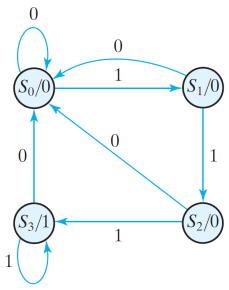


## Design Procedure

- Derive a state diagram
- Reduce the number of states
- Assign binary values to states
  - Integrate this into your state table
- Choose the type of flip-flops
- Derive the flip-flop input and output equations
- Draw (or code) the logic diagram



## State Diagram ->

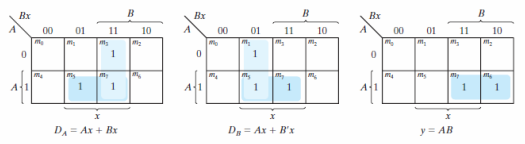


## Reduced Binary State Table ->

| Present State | | Input | Next State | | Output |
|---|---|---|---|---|---|
| **A** | **B** | **x** | **A** | **B** | **y** |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

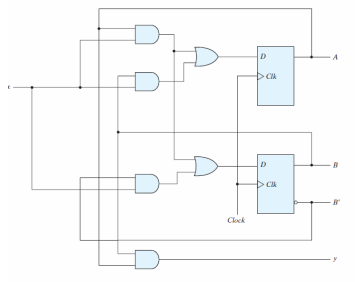


## Boolean Maps ->






## Circuit!



# Excitation Tables

*Flip-Flop Excitation Tables*

| Q(t) | Q(t + 1) | J | K |
|---|---|---|---|
| 0 | 0 | 0 | X |
| 0 | 1 | 1 | X |
| 1 | 0 | X | 1 |
| 1 | 1 | X | 0 |

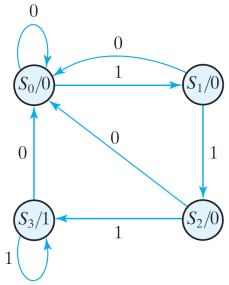
(a) *JK*

| Q(t) | Q(t + 1) | T |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

(b) *T*



# State Diagram ->

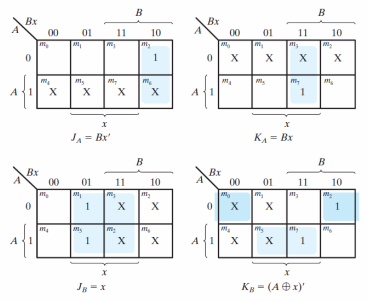





# JK FF Excitation Tables ->

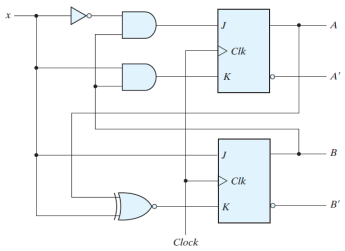
| Present State | | Input | Next State | | Flip-Flop Inputs | | | |
|---|---|---|---|---|---|---|---|---|
| A | B | x | A | B | $J_A$ | $K_A$ | $J_B$ | $K_B$ |
| 0 | 0 | 0 | 0 | 0 | 0 | X | 0 | X |
| 0 | 0 | 1 | 0 | 1 | 0 | X | 1 | X |
| 0 | 1 | 0 | 1 | 0 | 1 | X | X | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | X | X | 0 |
| 1 | 0 | 0 | 1 | 0 | X | 0 | 0 | X |
| 1 | 0 | 1 | 1 | 1 | X | 0 | 1 | X |
| 1 | 1 | 0 | 1 | 1 | X | 0 | X | 0 |
| 1 | 1 | 1 | 0 | 0 | X | 1 | X | 1 |

## Boolean Maps ->



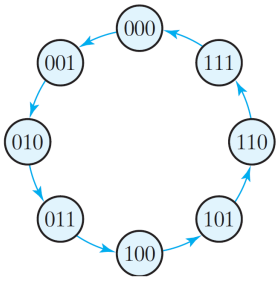



## Circuit






## How do we implement a binary counter?

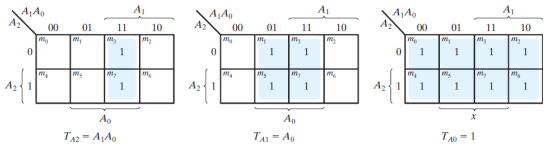


## State / Excitation Table ->

| Present State | | | Next State | | | Flip-Flop Inputs | | |
|---|---|---|---|---|---|---|---|---|
| $A_2$ | $A_1$ | $A_0$ | $A_2$ | $A_1$ | $A_0$ | $T_{A2}$ | $T_{A1}$ | $T_{A0}$ |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |

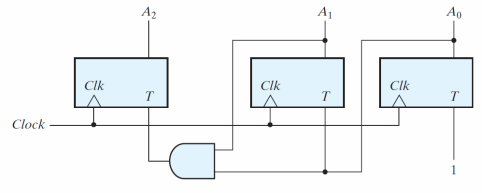


## Boolean Maps ->






## Circuit



## Sequential Circuit Design Process Review

- Derive a state diagram
- Reduce the number of states
- Assign binary values to states
  - Integrate this into your state table
- Choose the type of flip-flops
- Derive the flip-flop excitation table
- Create the Boolean Maps
- Draw (or code) the logic diagram



## Review

- Sequential Circuits
- Latches
- Flip-Flops
- Design of Sequential Circuits
  - State Diagrams
  - State Tables
    - Reduction of States
  - Excitation tables
  - Boolean Maps