Revision Guide

30.110 Digital Systems Lab, Term 6 2020

Wei Min Cher

23 Oct 2020

Contents

1 W1: Introduction

1.1 Analog vs Digital Signals

Analog signals	Digital signals
- Primitives from physical	- Primitives are Boolean functions
- Noise from thermal fluctuations	- Noise from roundoff errors
- Error accumulates	- Error does not accumulate through stages

1.2 Fixed-length Encodings

- To represent information in binary, we use fixed-length encodings
- e.g. 4-bit binary coded decimal (BCD)
  - Total of 10 decimal digits represented by 4 bits $\log_2(10) = 3.322 < 4$ bits
  - 10 decimal digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- 7-bit ASCII (American Standard Code for Information Interchange)
  - Total of 86 characters represented by 7 bits $\log_2(86) = 6.426 < 7$ bits
  - A-Z (26 chars), a-z (26 chars), 0-9 (10 chars), punctuation (11 chars), math (9 chars), financial (4 chars)

1.3 Base Conversion

- Decimal numbers are written normally or with subscript $_{10}$
- Octal numbers are prefixed with the letter O or with subscript $_8$
- Hexadecimal numbers are prefixed with 0x or with subscript $_{16}$

1.3.1 Converting to Decimal

$$\text{Total Decimal Value} = \sum_{i=\mathrm{p}_{min}}^{\mathrm{p}_{max}} d_i \cdot (\text{radix})^i$$

- In some cases, level of accuracy is specified to reduce number of fractional digits

1.3.2 Converting from Decimal

1. Whole number portion
   - Divide decimal number by base of system
   - Remainder recorded as least significant numeral
   - Process repeated until quotient of 0 is achieved

2. Fractional number portion
   - Multiply fractional component by base of system
   - Whole number recorded as most significant numeral
   - Process repeated until fractional component equal to zero
3. Rounding
   - Binary: If next bit is 1_2, round the LSB up.
   - Octal: If next bit is 4_8 or greater, round up.
   - Hexadecimal: if next bit is 8_{16} or greater, round up.

Decimal	**Binary**	**Octal**	**Hex**
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Table 1: Number system equivalency

1.3.3 Converting between 2^n Bases

1.3.3.1 Binary to Octal

1. Form groups of 3 bit representing octal symbols. $(010)(111).(010)_2$
2. Perform a direct substitution of the bit groupings with the equivalent octal symbol.

$$(010)(111).(010)_2 = 27.2_8$$

1.3.3.2 Binary to Hexadecimal

1. Form groups of 4 bit representing hexadecimal symbols. $(0011)(1011).(1111)(1000)_2$

2. Perform a direct substitution of the bit groupings with the equivalent hexadecimal symbol.

$$(0011)(1011).(1111)(1000)_2 = 3B.F8_{16}$$

1.3.3.3 Octal to Binary

1. Each of the octal symbols is replaced with its 3 bit binary equivalent.

$$347.12_8 = (011)(100)(111).(001)(010)_2 = 11100111.00101_2$$

1.3.3.4 Hexadecimal to Binary

1. Each of the hexadecimal symbols is replaced with its 4 bit binary equivalent.

$$1B.A_{16} = (0001)(1011).(1010)_2 = 11011.101_2$$

1.3.3.5 Octal to Hexadecimal

1. Convert the octal number into binary. $71.5_8 = (111)(001).(101)_2 = 111001.101_2$

2. Convert the binary number into hexadecimal. $(0011)(1001).(1010)_2 = 39.A_{16}$

1.3.3.6 Hexadecimal to Octal

1. Convert the hexadecimal number into binary. $AB.C_{16} = (1010)(1011).(1100)_2 = 10101011.11_2$

2. Convert the binary number into octal. $(010)(101)(011).(110)_2 = 253.6_8$

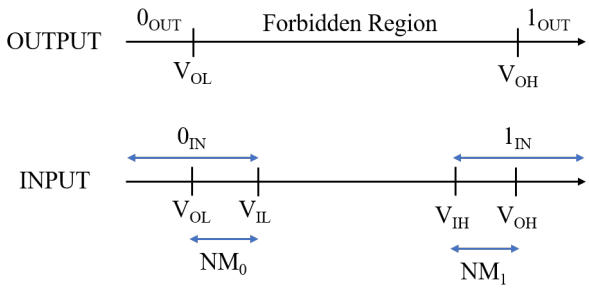
1.4 Two's Complement

Decimal	4-bit Two's Complement
-8	**1**000
-7	**1**001
-6	**1**010
-5	**1**011
-4	**1**100
-3	**1**101
-2	**1**110
-1	**1**111
0	**0**000
1	**0**001
2	**0**010
3	**0**011
4	**0**100
5	**0**101
6	**0**110
7	**0**111

- Range of a n-bit two's complement number, $N_{2\text{'s comp}}$: $-(2^{n-1}) \leq N_{2\text{'s comp}} \leq 2^{n-1} - 1$
- Convert two's complement number into decimal number:
  1. Sign bit represents -(2^{n-1}).
  2. Apply $\sum_{i=\mathrm{p}_{min}}^{\mathrm{p}_{max}} d_i \cdot (\text{radix})^i$ to find the corresponding decimal number.
- Taking the two's complement of a number:
  1. Perform complement on binary number.
  2. Add 1, ignore carry out if any.

1.4.1 Immunity Representation

- Analogy: strict at sender, tolerant at receiver



- Sending logical 0 (0_{OUT}): sender produces output voltage $\leq V_{OL}$
- Receiving logical 0 (0_{IN}): receiver receives input voltage $\leq V_{IL}$
- Sending logical 1 (1_{OUT}): sender produces output voltage $\geq V_{OH}$
- Receiving logical 1 (1_{IN}): receiver receives input voltage $\geq V_{IH}$

- Noise margin: absolute value of difference between input and output voltages for given logic value

$$NM_0 = V_{IL} - V_{OL} \qquad NM_1 = V_{OH} - V_{IH}$$

  - For reasonable noise margin, $V_{OL} \leq V_{IL},\ V_{OH} \geq V_{IH}$.
  - When $NM_0 = NM_1$, noise margins are symmetric.

- Forbidden region: range of voltage levels at which the logic value is undefined

1.5 Timing Specifications

- Propagation delay (t_{PD}): upper bound on delay from valid inputs to valid outputs
  - Maximum cumulative delay over all paths from inputs to outputs
  - Design goal: minimize this
  - Also known as $t_{PD,MAX}$
- Contamination delay (t_{CD}): lower bound delay from invalid inputs to invalid outputs
  - Minimum cumulative delay over all paths from inputs to outputs
  - If not specified, can be assumed to be 0.
  - Important when designing circuits with registers.
  - Also known as $t_{PD,MIN}$

1.6 Combination vs Sequential Logic

Combinational logic	Sequential logic
- e.g. calendar, 7-segment LED	- e.g. lock, ATM, traffic light
- No state	- State
- No memory	- Memory
- Truth table	- State diagram

2 W2: Boolean Algebra

2.1 Basic Definitions

- Set of elements, S: collection of objects with common property
- Elements, e.g. $x,\ y \in S$: objects that are in the set of elements
- Binary operator, e.g. $*,\ +$: rule that assigns to each pair of elements from S a unique element from S

2.2 Postulates of Algebraic Systems

1. Closure: A set is closed with respect to a binary operator if it describes a rule for obtaining a unique element of S for every pair of elements in S.
2. Associative law: A binary operator * on set S is associative if

$$(x * y) * z = x * (y * z) \text{ for all } x,\ y,\ z \in S.$$

3. Commutative law: A binary operator * on set S is commutative if

$$x * y = y * x \text{ for all } x,\ y \in S.$$

4. Identity element: A set S has an identity element e with respect to a binary operation * if

$$e * x = x * e = x \text{ for every } x \in S.$$

5. Inverse: A set S with an identity element e has an inverse y when

$$x * y = e \text{ every } x \in S.$$

6. Distributive law: If * and $\cdot$ are two binary operators on the set S, * is distributive over $\cdot$ if

$$x * (y \cdot z) = (x * y) \cdot (x * z).$$

2.3 Huntington Postulates

1. Closure:
   a) Structure is closed with respect to operator $+$.
   b) Structure is closed with respect to operator $\cdot$.
2. Identity element:
   a) 0 is the identity element with respect to $+$.
   b) 1 is the identity element with respect to $\cdot$.

3. Commutative law:

   a) Structure is commutative with respect to $+$.

   b) Structure is commutative with respect to $\cdot$.

4. Distributive law:

   a) Operator $\cdot$ is distributive over $+$. $\quad x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

   b) Operator $+$ is distributive over $\cdot$. $\quad x + (y \cdot z) = (x + y) \cdot (x + z)$

5. For every $x \in B$, there exists a complement $x' \in B$ such that

   (a) $x + x' = 1$, and

   (b) $x \cdot x' = 0$.

6. There are 2 elements $x,\ y \in B$ such that $x \neq y$.

2.4 Boolean Algebra vs Ordinary Algebra

	Boolean Algebra	**Ordinary Algebra**
Associative law	✓	✓
Distributive law of $+$ over $\cdot$ $x + (y \cdot z) = (x + y) \cdot (x + z)$	✓	×
Additive or multiplicative inverses	× (No subtraction or division)	✓ (Subtraction and division)
Complement	✓	×
Elements	0, 1	Real numbers

2.5 Two-Valued Boolean Algebra

- A set of two elements: 0 and 1
- Two binary operators equivalent to AND and OR operators
- Complement operator equivalent to NOT operator

2.6 Basic Theorems and Properties

- Duality
  - Every Boolean expression remains valid if the operators and identity elements are interchanged.
- Basic Theorems

	a)	**b)**
Postulate 2, identity element	$x + 0 = x$	$x \cdot 1 = x$
Postulate 5, complement	$x + x' = 1$	$x \cdot x' = 0$
Theorem 1	$x + x = x$	$x \cdot x = x$
Theorem 2	$x + 1 = 1$	$x \cdot 0 = 0$
Theorem 3, involution	$(x')' = x$	
Postulate 3, commutative	$x + y = y + x$	$xy = yx$
Theorem 4, associative	$x + (y + z) = (x + y) + z$	$x(yz) = (xy)z$
Postulate 4, distributive	$x(y + z) = xy + xz$	$x + yz = (x + y)(x + z)$
Theorem 5, De Morgan's Law	$(x + y)' = x'y'$	$(xy)' = x' + y'$
Theorem 6, absorption	$x + xy = x$	$x(x + y) = x$

  - To prove these theorems, either make use of the postulates or complete the truth table.
    - e.g. Theorems 1b and 2b are dual of theorems 1a and 2a.
    - Each step of proof in Theorems 1b and 2b is the dual of its counterpart in Theorems 1a and 2a.
- By manipulating the Boolean expression, it is possible to obtain a simpler expression.

2.7 Minterms and Maxterms

- Minterms: logical AND of a set of variables
  - Primed if 0, unprimed if 1
- Maxterms: logical OR of a set of variables
  - Primed if 1, unprimed if 0

			Minterms		**Maxterms**	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

- Any Boolean function can be expressed algebraically by taking sum of minterms which produce a 1.
- Any Boolean function can be expressed algebraically by taking product of maxterms which produce a 0.

2.8 Conversion between Minterms and Maxterms

- Complement of sum of minterms = Sum of minterms missing in original function
  - e.g. F(A, B, C) = $\sum(1, 4, 7) = m_1 + m_4 + m_7$
    $\Rightarrow$ F'(A, B, C) = $\sum(0, 2, 3, 5, 6) = m_0 + m_2 + m_3 + m_5 + m_6$
- Using De Morgan's Theorem on F', we can express F as a product of maxterms:

$$\begin{aligned} \text{F(A, B, C)} = \text{(F'(A, B, C))'} &= (m_0 + m_2 + m_3 + m_5 + m_6)' \\ &= m_0' m_2' m_3' m_5' m_6' \\ &= M_0 M_2 M_3 M_5 M_6 = \Pi(0, 2, 3, 5, 6) \end{aligned}$$

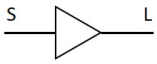
- Maxterm M_i is complement of minterm m_i. $\quad m_i' = M_i$

2.9 Two and Three-Level Implementation

- Two-level implementation preferred: produces least amount of delay
- However, number of inputs to logic gates may not practical.
- In those cases, one could use a three-level implementation instead.

2.10 Digital Logic Gates

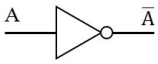
Buffer



S	L
0	0
1	1

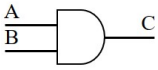
- Strengthens
- Delays

NOT gate



A	$\overline{A}$
0	1
1	0

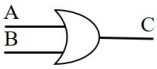
AND gate



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

$C = A \cdot B$

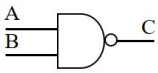
OR gate



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

$C = A + B$

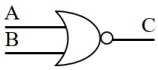
NAND gate



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

$C = \overline{A \cdot B}$

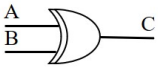
NOR gate



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

$C = \overline{A + B}$

XOR gate

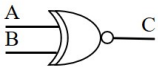


A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

$C = A \oplus B$

Output is 1 if only 1 input is 1

XNOR gate



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

$C = \overline{A \oplus B}$

Output is 0 if only 1 input is 1

2.11 Signal Logic and Logic Polarity

- Positive logic system: choosing logical high to represent logic 1
- Negative logic system: choosing logical low to represent logic 1
- Wedges are added at inputs and outputs to signify a negative logic gate.

3 W2: Gate-Level Minimization

3.1 K-Map

	0	1
0	m_0	m_1
1	m_2	m_3

	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

3.1.1 General Rules

- Priority should go to drawing the largest squares possible, grouping either 2, 4 or 8 minterms together.
- The larger the squares, the simpler the expression.
- All minterms with 1's should be covered when we combine squares.

3.1.2 Product of Sums Simplification

- Combine squares with 0's, then apply De Morgan's theorem to obtain the product of sums expression.

3.2 Don't Care Conditions

- Function is not specified for some combinations or we don't care
- X is used for don't care conditions
- Can be assumed to be either 0 or 1

3.3 NAND and NOR Implementation

- Any logic circuit can be implemented using NAND and NOR gates
- Using De Morgan's Theorem: $\overline{A \cdot B} = \overline{A} + \overline{B}$ $\quad \overline{A + B} = \overline{A} \cdot \overline{B}$

4 W2: Time Response

4.1 Gate Delays

- Signal propagation is not instantaneous
- Glitches: unwanted transient output changes
  - Occurs when different pathways have different delays
- Hazard: logic circuit with potential for glitches

4.2 Types of Hazards

- Static hazards
  - Brief glitch to other logic state
- Dynamic hazards
  - Multiple transitions instead of clean transition

4.3 Eliminating Static Hazards

- Use clock signals
- Lengthen waiting interval
- Add redundant K-map encirclements
  - To eliminate static 1-hazards: use SOP form
  - To eliminate static 0-hazards: use POS form
  - Only works for 2-level logic

5 W3: Combinational Logic

5.1 Combinational Circuit

- Circuit with n inputs, m outputs
- No feedback paths or memory elements

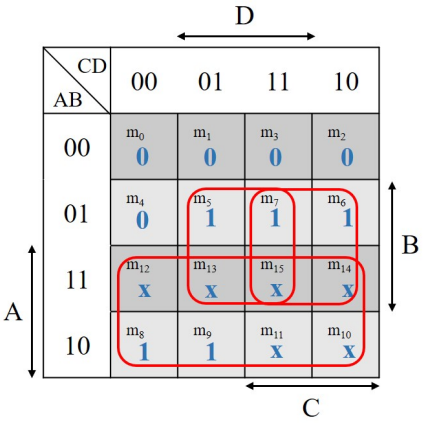
5.2 Design Process

1. Understand the problem.
   - e.g. *Build a circuit that converts from binary-coded decimal (BCD) to excess-3 binary code.*
2. From the circuit specifications, find the required number of inputs and outputs, assigning a symbol to each.
   - e.g. *For the circuit, there will be 4 inputs A, B, C and D and 4 outputs w, x, y and z.*
3. Derive a truth table that defines the required relationship between inputs and outputs.

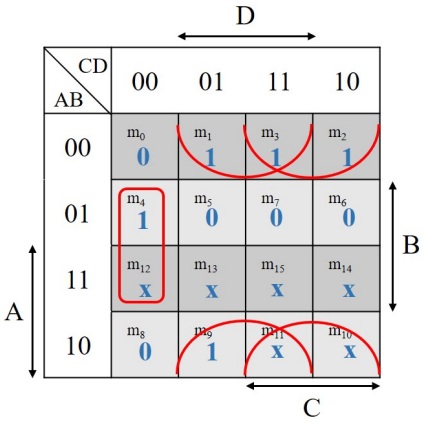
	Inputs (BCD)				**Outputs (Excess-3)**			
	A	**B**	**C**	**D**	**w**	**x**	**y**	**z**
	0	0	0	0	0	0	1	1
	0	0	0	1	0	1	0	0
	0	0	1	0	0	1	0	1
	0	0	1	1	0	1	1	0
	0	1	0	0	0	1	1	1
	0	1	0	1	1	0	0	0
	0	1	1	0	1	0	0	1
	0	1	1	1	1	0	1	0
	1	0	0	0	1	0	1	1
	1	0	0	1	1	1	0	0
Don't Care	1	0	1	0	x	x	x	x
Conditions	1	0	1	1	x	x	x	x
	1	1	0	0	x	x	x	x
	1	1	0	1	x	x	x	x
	1	1	1	0	x	x	x	x
	1	1	1	1	x	x	x	x

4. Obtain simplified Boolean functions for each output as a function of input variables using K-maps.

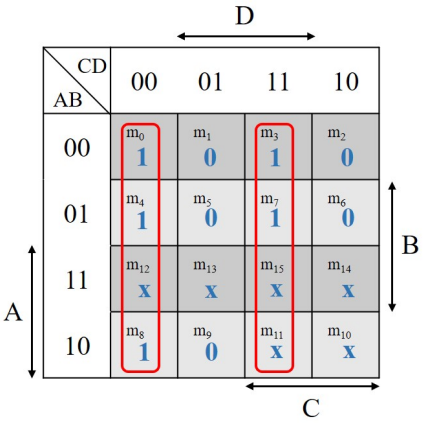
   Check for wrap-arounds!



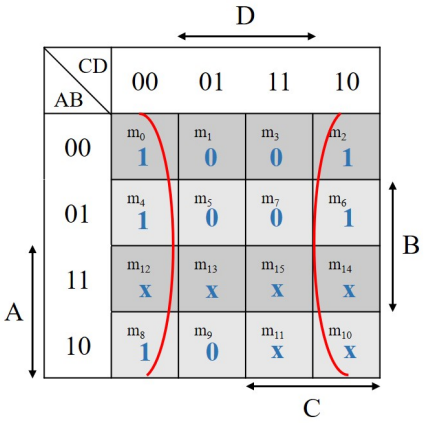
w = A + BC + BD



x = B'D + B'C + BC'D'



y = C'D' + CD



z = D'

5. Verify the correctness of the design either manually or by simulation.

6 W4: Sequential Circuits

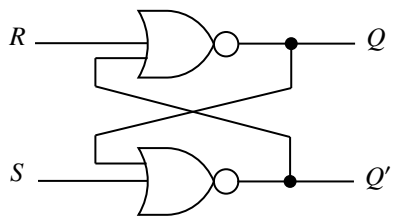
6.1 Overview

- Sequential circuit: present output depends on present input(s) and output(s)
  1. Synchronous sequential circuit: synchronised by clock signal
     - Reacts to changes slower e.g. flip-flops
  2. Asynchronous sequential circuit: not synchronised by clock signal
     - Reacts to changes quicker e.g. latches
- Combinational circuit: present output depends on present input(s)

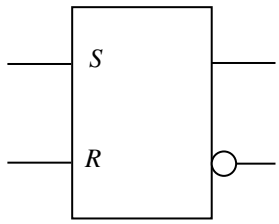
6.2 Latches

- Basic storage element
- Stores either 0 or 1.

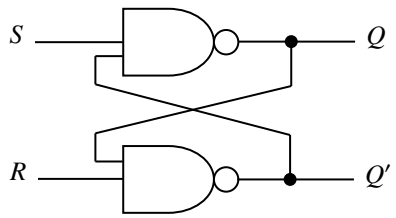
6.2.1 SR Latch



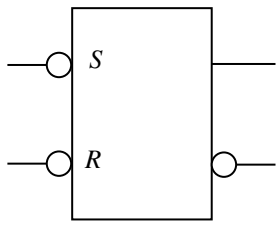
S	R	Q	Q’
0	0	Memory	
0	1	0	1
1	0	1	0
1	1	Forbidden	



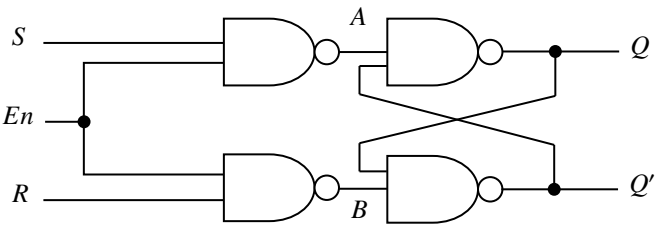
6.2.2 S’R’ Latch



S	R	Q	Q’
0	0	Forbidden	
0	1	1	0
1	0	0	1
1	1	Memory	



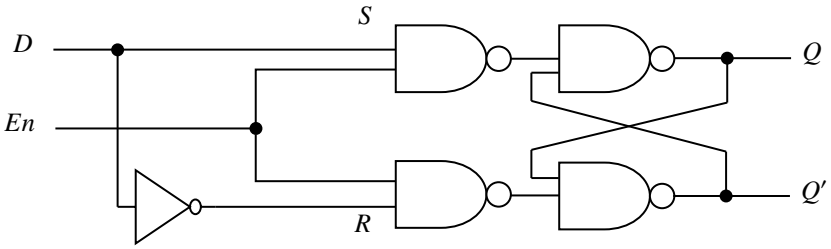
6.2.3 SR Latch with Control Input



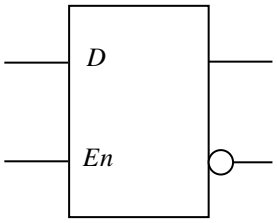
En	S	R	Q	Q’
0	x	x	Memory	
1	0	0	Memory	
1	0	1	0	1
1	1	0	1	0
1	1	1	Forbidden	

- Outputs stay as long as enable signal is 0.
- When enable input is 1, info from inputs can affect latch.

6.2.4 D Latch



En	D	$Q(t+1)$
0	x	$Q(t)$
1	0	0
1	1	1



- Eliminates condition when S and R are both 1.
- Holds data in internal storage.
- Output follows inputs as long as enable input is 1.

6.3 Latch vs Flip-flop

- Latch: operate with signal levels
  - e.g. level sensitive devices
- Flip-flops: controlled by clock transitions (positive/negative-edge responses)
  - e.g. edge sensitive devices

6.4 Flip-flops

- Storage elements that are controlled by clock transitions.
- Stores either 0 or 1.

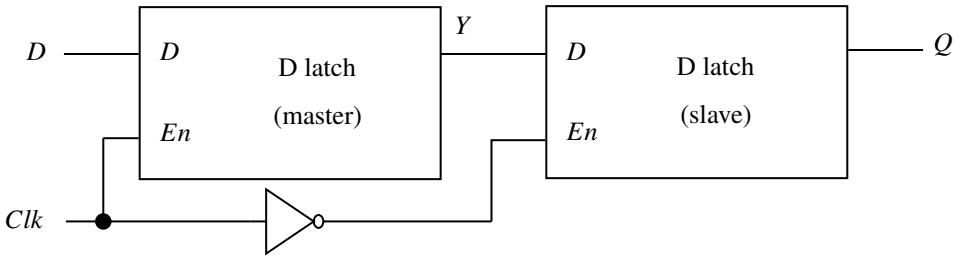
6.4.1 D Flip-flop

- Implementation of D latch using clock transitions instead of level transitions
- Value of D transferred to Q upon clock transition
- Characteristic equation: $Q(t+1) = D$

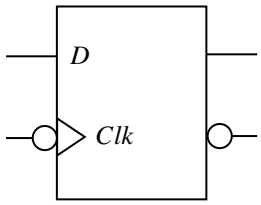
D	$Q(t+1)$	
0	0	Reset
1	1	Set

6.4.1.1 Negative-edge-triggered

- Also known as master-slave D flip-flop
- Value of D transferred to Q upon negative clock transition from 1 to 0

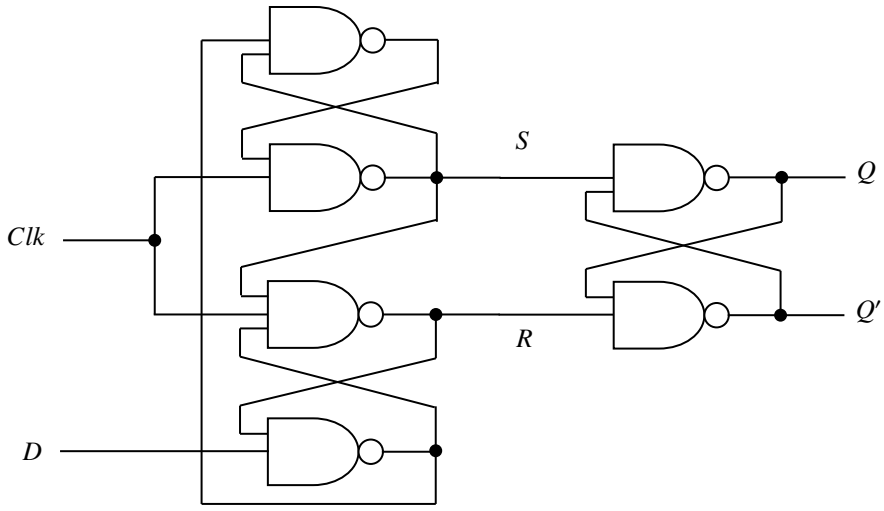


Clk	D	Q	Q'
↓	0	0	1
↓	1	1	0

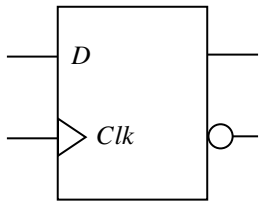


6.4.1.2 Positive-edge-triggered

- Value of D transferred to Q upon positive clock transition from 0 to 1

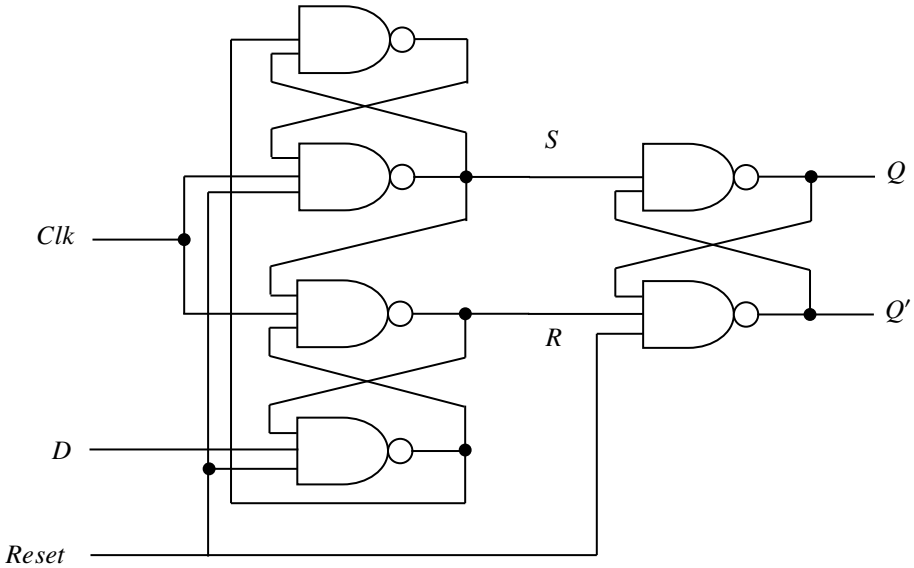


Clk	D	Q	Q’
↑	0	0	1
↑	1	1	0

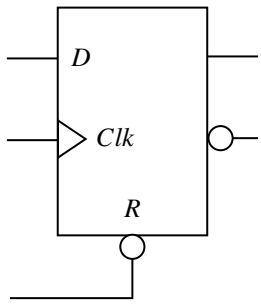


6.4.1.3 Positive-edge-triggered with Asynchronous Reset

- When $R = 0$, output Q is reset to 0 regardless of D or Clk.
- Value of D transferred to Q with every positive-edge clock signal, provided that $R = 1$.

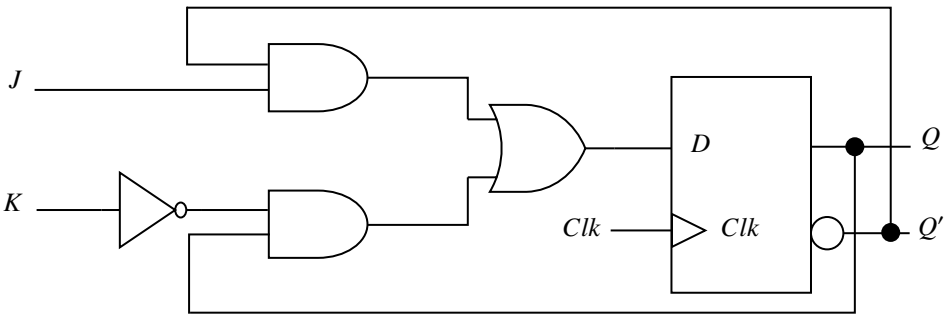


R	Clk	D	Q	Q’
0	x	x	0	1
1	↑	0	0	1
1	↑	1	1	0

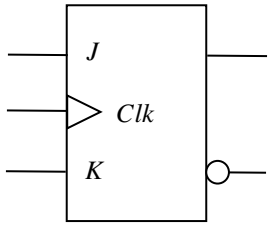


6.4.2 JK Flip-Flop

- Characteristic equation: $Q(t + 1) = JQ' + K'Q$

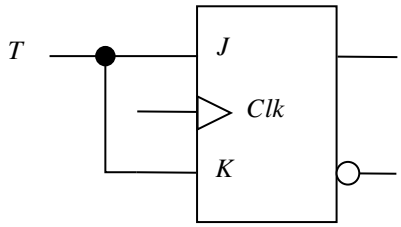


J	K	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

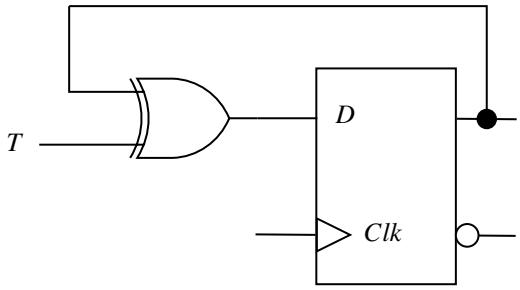


6.4.3 T Flip-Flop

- Characteristic equation: $Q(t+1) = T \bigoplus Q = TQ' + T'Q$

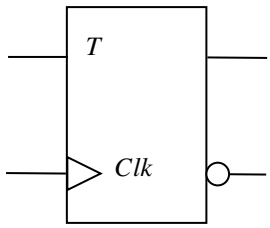


Derived from JK flip-flop



Derived from D flip-flop

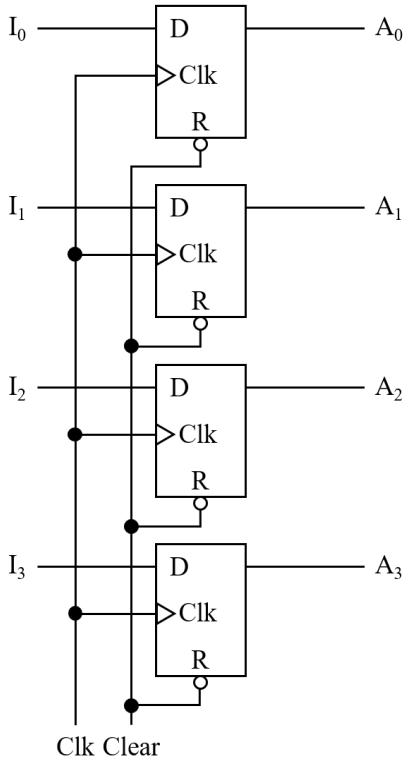
T	Q(t+1)	
0	Q(t)	No change
1	Q'(t)	Reset



6.5 Registers

- Group of flip-flops with a common clock
- Each flip-flop is capable of storing 1 bit of information

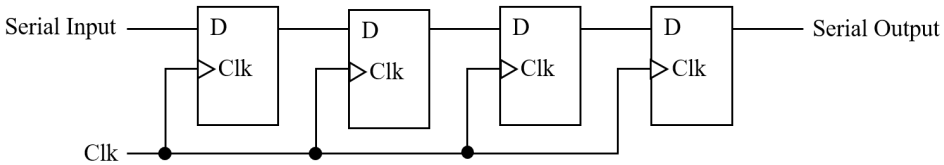
6.5.1 4-bit Register



- Triggered by positive edge of clock
- `Clear` must be maintained at logic 1 during clock operation
  - Allows for asynchronous reset of output values
- Two ways for outputs to remain constant:
  1. Maintain constant input values
  2. Stop the clock

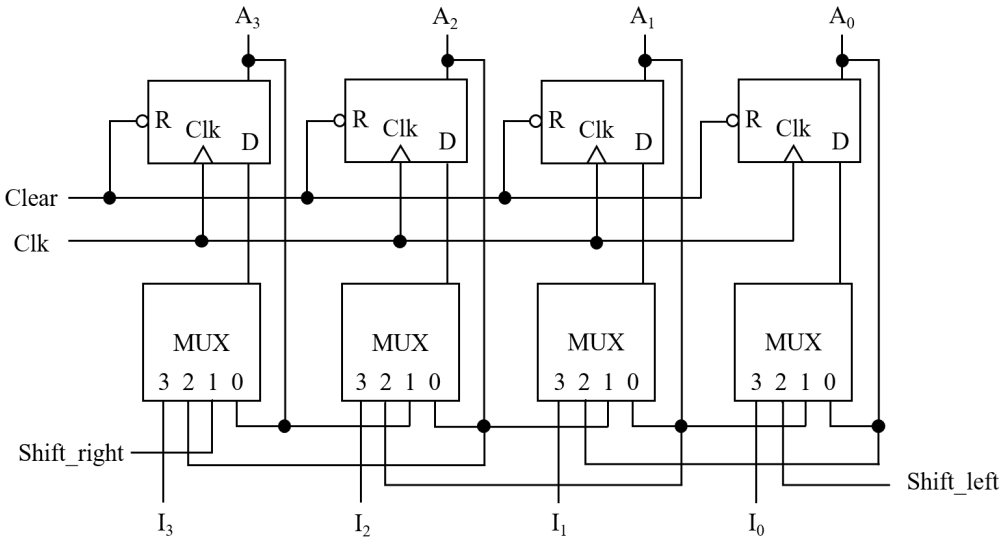
6.5.2 Shift Register

- Shifts binary information in each flip-flop in one direction
- Each clock pulse shifts value of flip-flop one position to the right



6.5.3 Universal Shift Register

- Register that performs bidirectional bit-shift and parallel transfer operations
- Takes two inputs s_0 and s_1



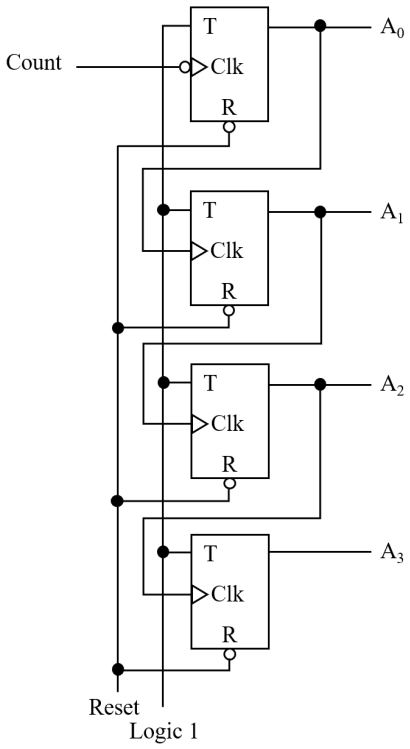
Inputs

s_1	s_0	**Register operation**
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel transfer

6.6 Counters

- A register that cycles through a predetermined sequence of binary states

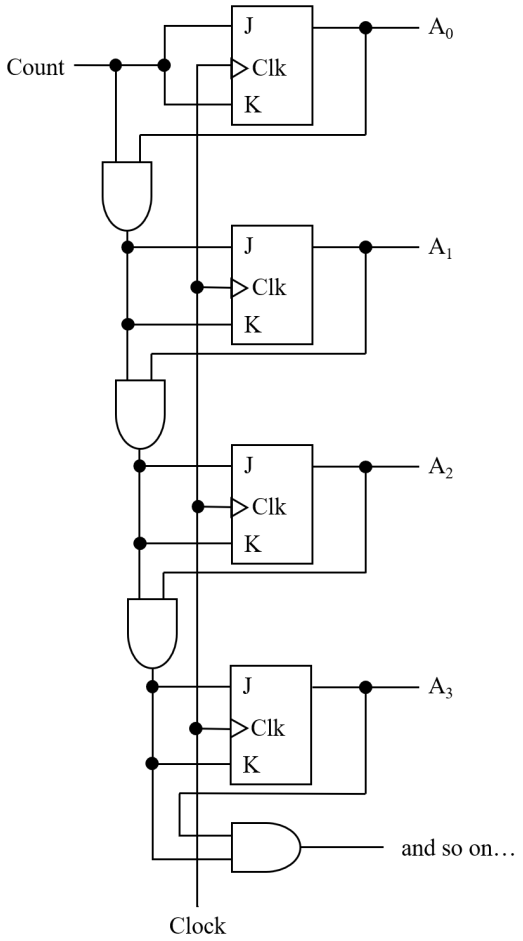
6.6.1 Binary Ripple Counter



- Negative edge triggered with asynchronous reset

- When A_0 goes from 1 to 0, it triggers and complements A_1.
- When A_1 goes from 1 to 0, it triggers and complements A_2.
- When A_2 goes from 1 to 0, it triggers and complements A_3.

6.6.2 Binary Synchronous Counter



- Outputs depend on `count` and/or outputs

- When `count` = 1, A_0 is complemented.
- When `count` = 1, $A_0 = 1$, A_1 is complemented.
- When `count` = 1, $A_0 = 1$, $A_1 = 1$, A_2 is complemented.
- When `count` = 1, $A_0 = 1$, $A_1 = 1$, $A_2 = 1$, A_3 is complemented.

7 W5: Analysis of Sequential Circuits

7.1 Overview

- Analysis of sequential circuits:
  - Circuit diagram → State equation → State table → State diagram
- Design of sequential circuits:
  - System specifications → State diagram → State table → Circuit/HDL

7.2 Analysis of Sequential Circuits

- A sequential circuit can be represented by:
  1. State equations
  2. State table
  3. State diagram

7.2.1 State Equation

- State equation: specifies next state as function of present state and inputs
  - Left hand side: next state of flip-flop input/output
  - Right hand side: Boolean expression specifying present state of flip-flop inputs and inputs
- Deriving state equations
  1. Determine input(s), output(s) and state variables of sequential circuit.

     e.g. Input: x, Output: y, States: A, B
  2. If there are JK or T flip-flops, determine flip-flop inputs in terms of present state variables and input(s).

     e.g. $\mathrm{D}_A = Ax + Bx,\ \mathrm{D}_B = A'x$
  3. Express next state of state variables/output(s) in terms of flip-flop inputs.

     $$\mathrm{A}(t+1) = \mathrm{D}_A = Ax + Bx,\ \mathrm{B}(t+1) = \mathrm{D}_B = A'x,\ y(t+1) = x'(A+B)$$
  4. Rewrite next state of state variables/output(s) in terms of present state variables and input(s) only.

7.2.2 State Table

- Enumerated time sequence of inputs, outputs and flip-flop states
- Sequential circuit with m flip-flops, n inputs needs 2^{m+n} rows in 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

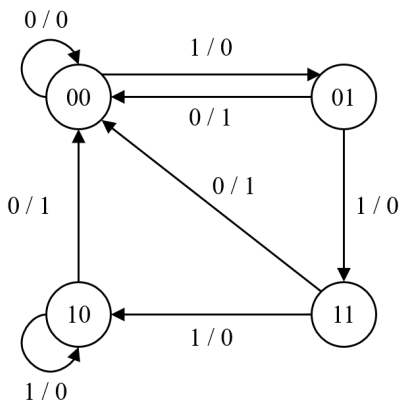
e.g. $\mathrm{A}(t+1) = \mathrm{D}_A = Ax + Bx$

$\mathrm{B}(t+1) = \mathrm{D}_B = A'x$

$y(t+1) = x'(A+B)$

7.2.3 State Diagram

- Graphical representation of information in state table
- Function of sequential circuit can be deciphered from it

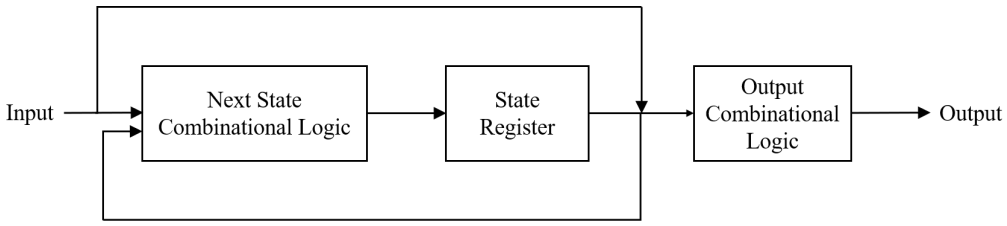


e.g. Detecting a ”0” in bit stream of data

- Input of 0 after a series of 1’s gives an output of 1
- State returns to initial state

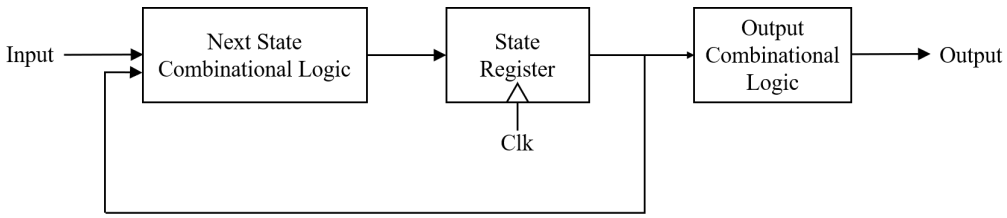
7.3 Types of State Machines

7.3.1 Mealy Machine



- Output function of both present state and input
  - Outputs may change if inputs change during clock cycle
- State symbols in state diagram: (00), (01)

7.3.2 Moore Machine



- Output function of only present state
  - Outputs synchronized with clock
- State symbols in state diagram: (00/0), (01/1)

7.4 State Reduction

- Reduces no. of states in state table, while keeping inputs and outputs unchanged
  - Reduces no. of flip-flops needed
- Two states are equivalent if they give the same outputs and subsequent states for the same input
- e.g.

	Next state		**Output**	
Present state	$x = 0$	$x = 1$	$x = 0$	$x = 1$
e	a	f	0	1
g	a	f	0	1

- e and g are equivalent states
- g can be removed
- All next states to g replaced with e

7.5 State Assignment

- Assign a unique binary value to each state
- e.g.

State	Binary encoding	Gray encoding	One-hot encoding
a	000	000	00001
b	001	001	00010
c	010	011	00100
d	011	010	01000
e	100	110	10000

8 W6: Design of Sequential Circuits

8.1 Overview

- Design of sequential circuits:
  - System specifications → State diagram → State table → Circuit/HDL

8.2 Excitation Table

- Lists required inputs for a given change of state

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

JK Flip-flop

$Q(t)$	$Q(t+1)$	T
0	0	0
0	1	1
1	0	1
1	1	0

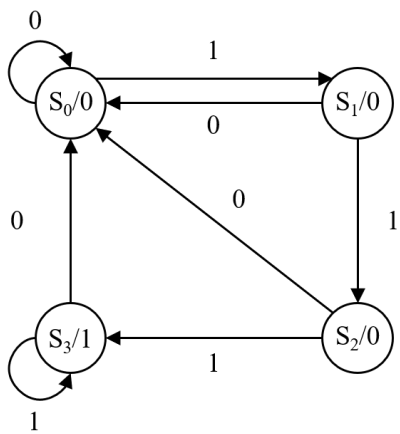
T Flip-flop

- Derived from characteristic tables

8.3 Steps of Sequential Logic Design

Designer Focus

1. Understand the problem.
   - e.g. *Design a circuit that detects a sequence of three or more consecutive 1's.*
     Output 1 after three or more consecutive 1's, 0 otherwise.
2. Obtain abstract representation of finite state machine e.g. state diagram.



3. Perform state assignment.

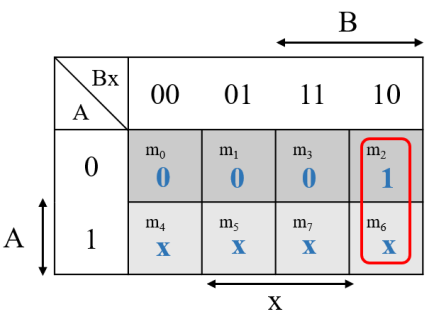
State	Binary value
S_0	00
S_1	01
S_2	10
S_3	11

Performed by Synthesis Tool

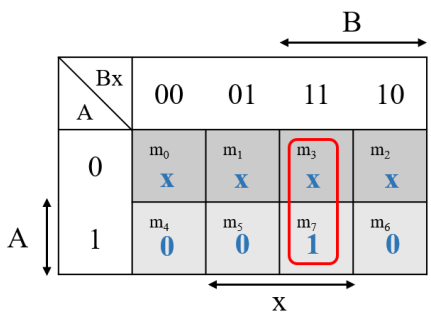
4. Obtain state table.

Present state		Input	Next state		Output	Flip-flop inputs			
A	B	x	A	B	y	J_A	J_B	K_A	K_B
0	0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	0	x	1	x
0	1	0	1	0	0	1	x	x	1
0	1	1	0	1	0	0	x	x	0
1	0	0	0	0	0	x	0	0	x
1	0	1	1	1	0	x	0	1	x
1	1	0	0	0	1	x	0	x	0
1	1	1	1	1	1	x	1	x	1

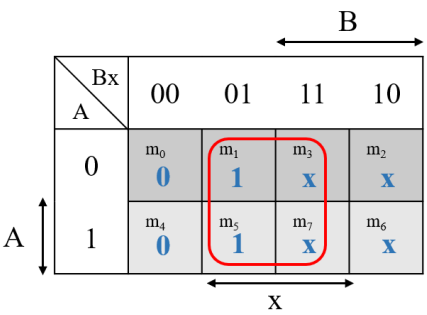
5. Perform state minimization.



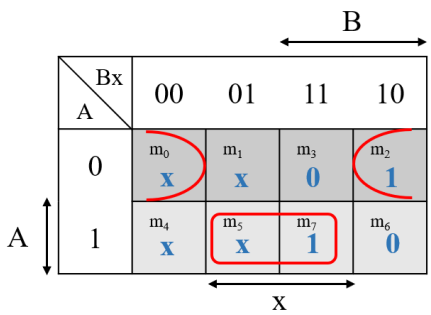
$J_A = Bx'$



$K_A = Bx$

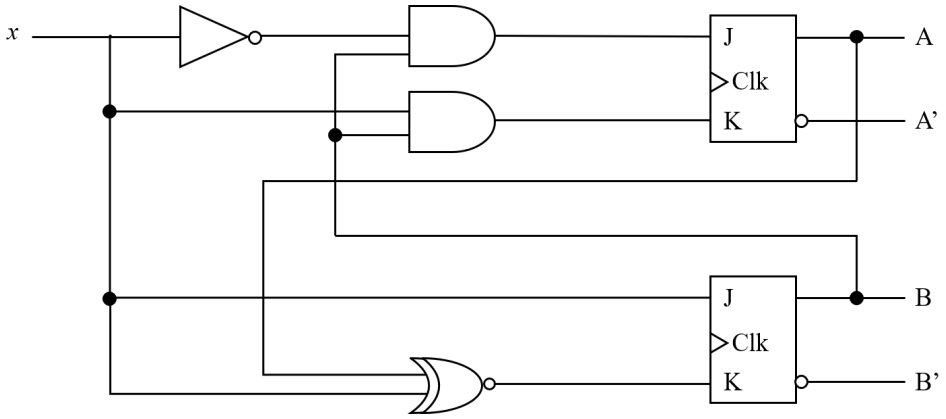


$J_B = x$



$K_B = Ax + A'x'$

6. Implement finite state machine.



9 Miscellaneous Definitions

9.1 Comparison between Tables

- A **truth table** describes a combinational circuit.
- A **state table** describes a sequential circuit.
- A **characteristic table** describes the operation of a flip-flop.
- A **excitation table** gives the values of flip-flop inputs for a given state transition.

9.2 Comparison between Equations

- A **Boolean equation** is an algebraic expression of a truth table.
- A **state equation** is an algebraic expression of a state table.
- A **characteristic equation** is an algebraic expression of a characteristic table.
- A **flip-flop input equation** is an algebraic expression of an excitation table.