Chapter 6 Digital

6.1 Introduction

Digital systems are the language of computers and microprocessor hardware. A basic understanding of the principals associated with digital systems are presented.

6.2 Binary (Digital) Systems

6.2.1 Number Systems

Decimal numbers

$$7392_{10} = 7\times10^3 + 3\times10^2 + 9\times10^1 + 2\times10^0$$

When using numbers, we usually write only the coefficient and let the position indication the power of 10. The coefficient range = 0-9 for base 10

Binary numbers

The coefficient range = 0-1 for base 2

$$1010_2 = 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0 = 10_{10}$$

Octal numbers

Octals are simply binary numbers combined into groups of 3 (base 8=2^3)

$$8 = 2^3 = 1|001 = 12_8$$

The coefficient range for octal numbers is 0-7

Hexadecimal numbers

Hexadecimal numbers are binary numbers that are combined into groups of 4 (base 16 = 2^4)

$$16 = 2^4 = 0|1010 = 0A_{16}$$

The range for hexadecimal numbers is 0-F.

Hex #	Decimal Equiv	Hex #	Decimal Equiv
0	1	8	9
1	2	9	10
2	3	A	11
3	4	B	12
4	5	C	13
5	6	D	14
6	7	E	15
7	8	F	16

Arithmetic

Arithmetic rules for all bases have the same basic set of rules.

Addition
$$\begin{array}{r}111\\ \underline{101}\\ 1100\end{array}$$

Subtraction
$$\begin{array}{r}111\\ \underline{101}\\ 010\end{array}$$

Multiplication
$$\begin{array}{r}111\\ \underline{101}\\ 111\\ 000\\ \underline{111}\\ 100011\end{array}$$

Decimal to Binary Equivalence

$0 = 0000$	$5 = 0101$
$1 = 0001$	$6 = 01101$
$2 = 0010$	$7 = 0111$
$3 = 0011$	$8 = 1000$
$4 = 0100$	$9 = 1001$

6.2.2 Binary System Wiring

Equivalent Names

5V = 1 = high = True = on = closed

0V = 0 = low = False = off = open

Resistors

Pull-up – from power source –
2.2KΩ

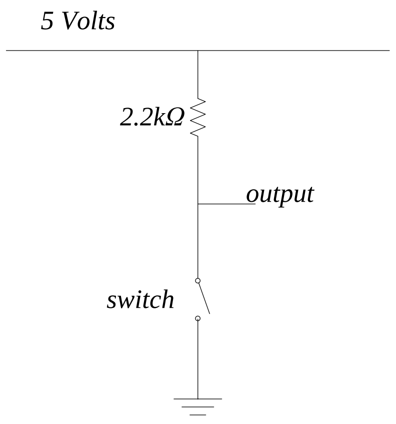
Current limiting – in series with load
330Ω

Switch

When switch = open, output = 1 (True)

When switch = closed, output = 0 (False)

Called an inverting switch

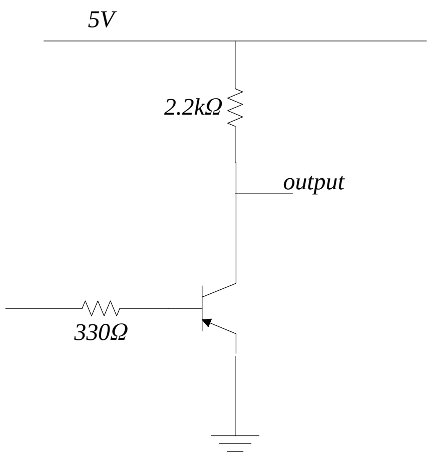


Transistor

Simply a digital switch with an electrical input (inverter)

When base switch = off, output = 1 (True)

When base switch = on, output = 0 (False)



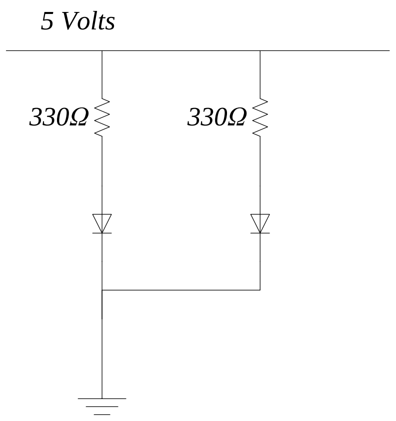
Output – LED

Generally connected to ground

Use current limiting resistor

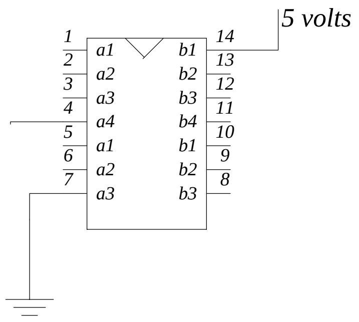
LED Long leg is the ground

This is a common cathode arrangement. A common anode would have inverted diodes



TTL

- Standard Arrangement
- 5V Power, VCC, on upper right pin
- Ground on lower left pin
- Notch at top



Problem:

Wire a switch circuit with an LED connected to the output pin.

6.2.3 The Huntington Postulates

Required entities to define an algebra:

A defined group of coefficients (R, N, Q, Z [0,1]), Real Natural, Complex Binary

A defined group of operators and a table which defines how each operator works.

A defined group of axioms or postulates (unproven theorems) from which new theorems, lemmas, corollaries, and propositions may be constructed.

[0,1] = *B* (coefficients)

OR

+	0	1
0	0	1
1	1	1

AND

*	0	1
0	0	0
1	0	1

NOT

~	
0	1
1	0

For all *x.y* elements of group *B*: for each operator

P1 (a) $x + y \in B$ (b) $x * y \in B$ (closure) only 1 or 0

P2 (a) $x + 0 = x$ (add Identity, I=0) (b) $x * 1 = x$ (multiply Identity, I=1)

P3 (a) $x + y = y + x$ (b) $x * y = y * x$ (commutative)

P4 (a) $x * (y + z) = (x * y) + (x * z)$ (b) $x + (y * z) = (x + y) * (x + z)$ (distributive)

P5 (a) $x + x' = 1$ (b) $x * x' = 0$ (complement, unique)

P6 (a) *B* contains at least 2 distinct elements

Useful theorems: listed in pairs that have correspondence – duality – every expression is valid if operator and identity elements are changed. To find the dual, exchange + to * and 1 to 0.

T1 (a) $x + x = x$ (b) $x * x = x$

T2 (a) $x + 1 = 1$ (b) $x * 0 = 0$

T3 (a) $(x')' = x$ (involution)

T4 (a) $x + (y + z) = (x + y) + z$ (b) $x(yz) = (xy)z$ (associative)

DeMorgan's Theorem – complement function by interchanging AND & OR operators and complementing each literal

T5 (a) $(x + y)' = x'y'$ (b) $(x * y)' = x' + y'$

T6 (a) $x + xy = x$ (b) $x(x + y) = x$ (absorption)

T7 (a) $x + x'y = x + y$ (b) $x(x' + y) = xy$

Operator Precedence

Parentheses ()

Not

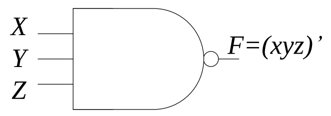
AND

OR

Complement – take dual and complement each literal

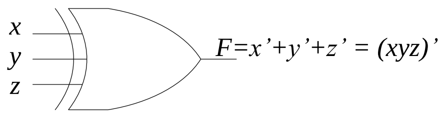
6.2.4 Basic Digital Gates

NAND



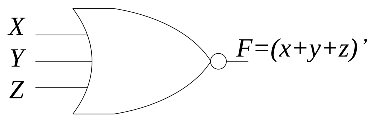
And Invert

XOR



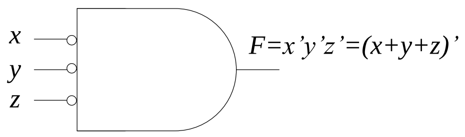
Invert Or

NOR



Or Invert

XAND



Invert And

A one input gate acts like an inverter

To use NAND requires Sum of Products (SOP) form

To use NOR requires Product of Sums (POS) form

6.3 Karnaugh Maps

Definitions:

Literal	a variable of the problem
Don't Care	Output values for which no value is necessary (situations that never occur, or if they do, can be ignored
SOP	Abbreviation for Sum of Product
POS	Abbreviation for Product of Sum
Reflected Code	A binary code which has the property that the codes following every 2^{n-1} codes are the reflection of the first 2^{n-1} codes in all bits except the most significant. The MSB of the first 2^{n-1} is a zero, and the MSB of the next 2^{n-1} codes is a 1.

Example

Binary Code	Reflected Code
00	00
01	01
10	11
11	10

PURPOSE:

A method of realizing a function in either of the two standard forms such that the number and complexity of terms in the function is minimal.

Construction:

- Draw a square or rectangular figure allowing 2^n squares (for a problem with n variables).
- The table should be drawn so that there are 2^x rows and 2^y columns where $x + y = n$.
- Label the rows and columns with reflected code from left to right and top to bottom.
- Fill in each square of the map with its corresponding truth table function value.

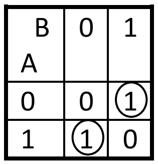
Simplification:

- Circle the largest contiguous binary (2^w, $w \le n$) group of 1's for SOP (0's for POS) which is rectangular or square
- Consider all the edges of the map to be physically adjacent.
- Each circled block of 1's (0's) corresponds to one SOP (POS) term. The term is extracted by observing which literals do not change for the block. These can be complemented to create the POS terms using 0's. The literals are then AND'ed (OR'ed) together to realize the function.
- Finally, each term, which corresponds to each rectangular block, is OR'ed (AND'ed) together to realize the function
- Each 1 (0) in the map must be circled at least once to realize the function. (Sometimes there is more than one way to do it).
- Additional Rule for Don't Cares: It is not required that don't cares be circled, but treat them as 1 (0) if it will help with minimization.

6.3.1 Construction / Simplification of Karnaugh Maps:

Example 1:

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

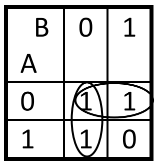


B / A	0	1
0	0	1
1	1	0

$$= AB' + A'B$$
$$= (A + B)(A' + B')$$

Example 2:

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

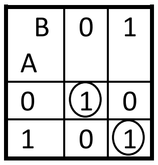


B / A	0	1
0	1	1
1	1	0

$$= A' + B'$$
$$= A' + B'$$

Example 3:

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

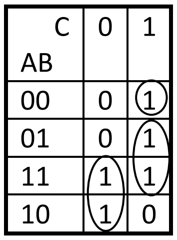


B / A	0	1
0	1	0
1	0	1

$$= A'B' + AB$$
$$= (A + B')(A' + B)$$

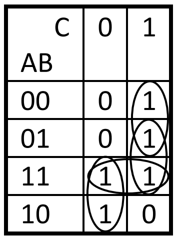
Example 4:

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



C / AB	0	1
00	0	1
01	0	1
11	1	1
10	1	0

$$= A'B'C + BC + AC'$$
$$= (A + C)(A'BC')$$



C / AB	0	1
00	0	1
01	0	1
11	1	1
10	1	0

Eliminate Races - Overlap

$$F = A'C + BC + AB + AC' \qquad \text{(SOP)}$$

Example 5:

$F = \Sigma(0,1,3,4,6,7) + \Pi(2,5)$

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

$F = \Sigma(0,1,3,4,6,7)$

C AB	0	1
00	1	1
01	0	1
11	1	1
10	1	0

$F = \Pi(2,5)$

C AB	0	1
00	1	1
01	0	1
11	1	1
10	1	0

C AB	0	1
00	1	1
01	0	1
11	1	1
10	1	0

$F = (A + B' + C)(A + B + C')\ (POS)$

$F = A'B' + BC + AC' + A'C$ (SOP)

C AB	0	1
00	1	1
01	0	1
11	1	1
10	1	0

$F = B'C' + A'C + AB$ (SOP)

C AB	0	1
00	1	1
01	0	1
11	1	1
10	1	0

Eliminate Races

$F = A'B' + A'C + AB + BC + AC'$ (SOP)

6.4 Design w/ Multiplexer

Multiplexers are a mixed bag

Designers try to minimize gates	good
Do clever things	good
Results are not logical	bad
Hard to troubleshoot	bad

Use multiplexer to show sequence of states

3 level pattern

#1 – multiplexers that determine the next state of registers

#2 – Register that holds present binary state

#3 – Decoder that provides a separate output for each control state

Put "Select" line across top of Karnaugh map

With "C" Select

C / AB	0	1
00	0	0
01	1	1
11	1	0
10	1	0

$$F = A'B + AC' + BC'$$
$$F' = A'B' + AC + B'C$$

$$Select = C$$
$$I_0 = AB + AB + AB'$$
$$I_1 = A'B$$

With "A" Select

A / BC	0	1
00	0	1
01	0	0
11	1	0
10	1	1

$$F = A'B + B'C + AC'$$
$$F' = A'B' + B'C + AC$$

$$Select = A$$
$$I_0 = B$$
$$I_1 = C'$$

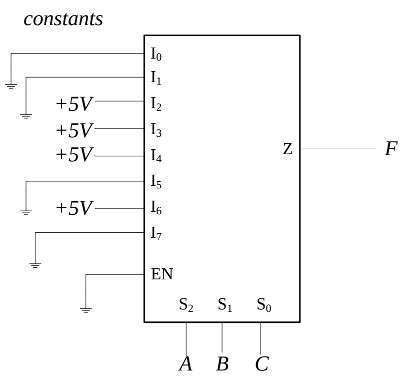
<u>Three different implementations of the same function.</u>

A valve header requires the following logic to control its actions:

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

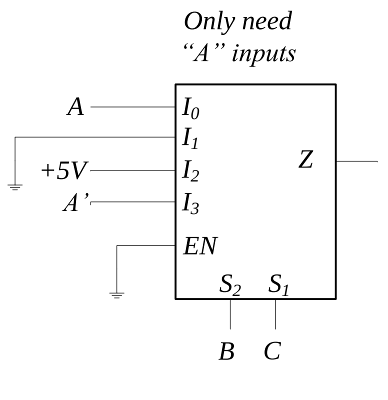
$$F = \Sigma(2,3,4,6)$$
$$= \Pi(0,1,5,7)$$

<u>For 8:1 Multiplexer</u>



BC / A	00	01	11	10
0	0^{0}	0^{1}	1^{3}	1^{2}
1	1^{4}	0^{5}	0^{7}	1^{6}

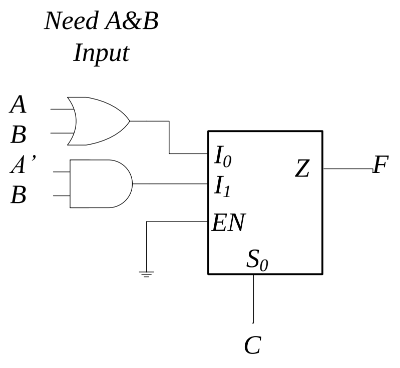
<u>For 4:1 Multiplexer</u>



BC / A	00	01	11	10
0	0^{0}	0^{1}	1^{3}	1^{2}
1	1^{4}	0^{5}	0^{7}	1^{6}
E	0	1	3	2

B & C are select
∴ Do not cross areas of constant BC

For 2:1 Multiplexer



BC A	00	01	11	10
0	0^{0}	0^{1}	1^{3}	1^{2}
1	1^{4}	0^{5}	0^{7}	1^{6}

C=0

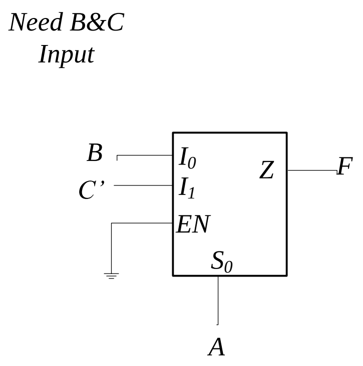
B A	0	1
0	0	1
1	1	1

C=1

B A	0	1
0	0	1
1	0	0

C is select
∴Do not cross areas of constant C
Use I_0 for C = 0 Area
Use I_1 for C = 1 Area

For 2:1 Multiplexer – different select



BC A	00	01	11	10
0	0^{0}	0^{1}	1^{3}	1^{2}
1	1^{4}	0^{5}	0^{7}	1^{6}

A=0

C B	0	1
0	0	0
1	1	1

A=1

C B	0	1
0	1	0
1	1	0

A is select
∴Do not cross areas of constant A
Use I_0 for A = 0 Area
Use I_1 for A = 1 Area

Choice of inputs to select lines determines how to partition K-Map

- Select line – ABC, then each value is fixed, nothing changes
- Select line = BC, then A value changes
- Select line = C, then AB values change

6.5 Decoder

Function

SOP = Σ(minterms) (OR of minterms)

POS = Π(maxterms) (AND of maxterms)

Each output of decoder is a minterm

For function in SOP

Use Active Hi output for OR gate

If use Active Lo output – must use invert-OR = NAND

For function in POS

Active Lo output – use AND gate

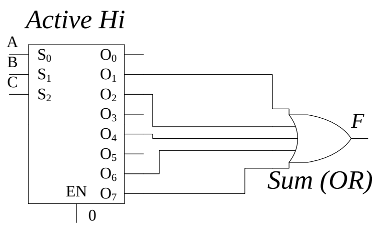
If use Active Hi output – must use invert-AND = NOR

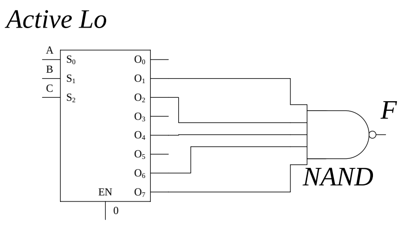
Decoder Example

SOP Form

SOP uses 1's

F= Σ(1,2,4,6,7)

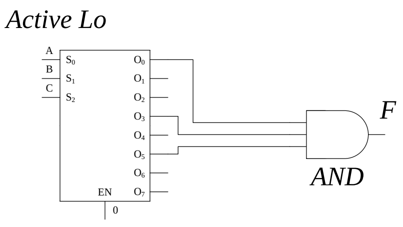


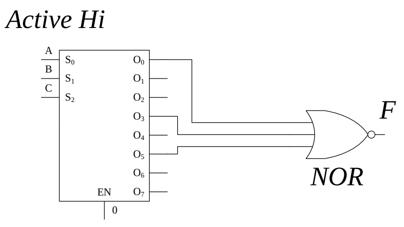


POS Form

POS Uses 0's

F= Π(0,3,5)





Note that the Functions are equivalent - F= Σ(1,2,4,6,7) = Π(0,3,5)

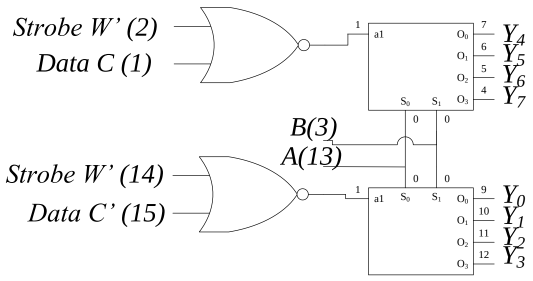
6.5.1 74156 DECODER

The 74156 is a dual 2:4 decoder

The 2 input lines can be decoded into 4 output lines

An Active low device – when the input is selected low, the decoded output line will be low

Two separate 2:4 decoders may be individual, or connected as a 3:8



The select lines are common

WCBA is sequence for complete address

W' have write to output, this is enable line

C-C' tie together – data

BA Address

RAM

Write – write line=0, then data (1 or 0) on data line is clocked into decoder

A_1A_0 – Same address connected to both decoders

DATA (C-C')– Inverted on decoder b, direct on Decoder a, therefore output of

both each side of a FF will force the FF to change state

6.6 Flip Flops / Latch

S	R	Q	Q_{t+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	indeterminate
1	1	1	indeterminate

When *$S=0$, $R=1$, $Q_{t+1} = 0$*
When *$S=1$, $R=0$, $T_{t+1} = 1$*

Q / SR	0	1
00	0	1
01	0	0
11	IND	IND
10	1	1

QR Flip Flop Function – $Q_{t+1} = S + R'Q$

6.6.1 General – Flip Flop Types

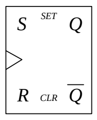
Characteristic Table **Excitation Table** **Characteristic Equation** **Symbol** **Circuit**

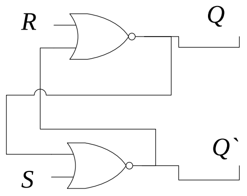
Asynchronous (RS Flip Flop)

S	R	Q_{t+dt}	Q'_{t+dt}
0	0	Q_t	Q'_t
0	1	1	0
1	0	0	1
1	1	0	0

Q_t	Q_{t+dt}	S
0	0	0
0	1	1
1	0	0
1	1	-

$$Q_{t+dt} = S + R'Q_t$$
$$SR = 0$$

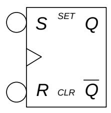


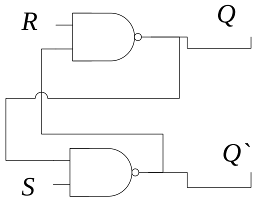


S	R	Q_{t+dt}	Q'_{t+dt}
0	0	1	1
0	1	1	0
1	0	0	1
1	1	Q_t	Q'_t

Q_t	Q_{t+dt}	S
0	0	1
0	1	0
1	0	1
1	1	-

$$Q_{t+dt} = S' + RQ_t$$
$$S + R = 1$$



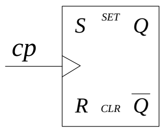


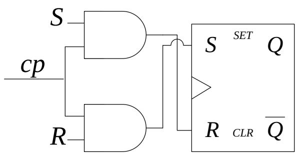
Synchronous

S	R	Q_{t+1}	Q'_{t+1}
0	0		
0	1	1	0
1	0	0	1
1	1	0	0

Q_t	Q_{t+1}	S
0	0	0
0	1	1
1	0	0
1	1	-

$$Q_{t+1} = S + R'Q_t$$
$$SR = 0$$

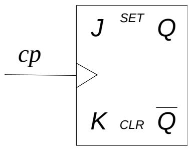


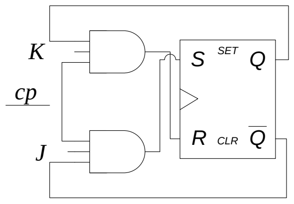


J	K	Q_{t+1}	Q'_{t+1}
0	0	Q_t	Q'_t
0	1	0	1
1	0	1	0
1	1	Q'_t	Q_t

Q_t	Q_{t+1}	J
0	0	0
0	1	1
1	0	-
1	1	-

$$Q_{t+1} = JQ'_t + K'($$

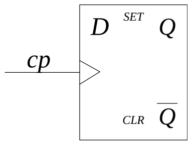


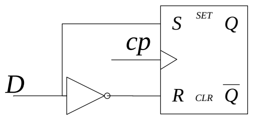


D	Q_{t+1}	Q'_{t+1}
0	0	1
1	1	0

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

$$Q_{t+1} = D$$

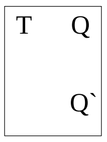


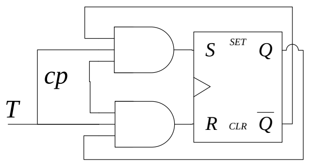


T	Q_{t+1}	Q'_{t+1}
0	Q_t	Q'_t
1	Q'_t	Q_t

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

$$Q_{t+1} = TQ'_t + T'($$





6.6.2 Counter

Sequential circuit that goes through a prescribed sequence of states on application of input pulse

Binary counter – counter that follows binary sequence\

n-bit counter has n flip flops

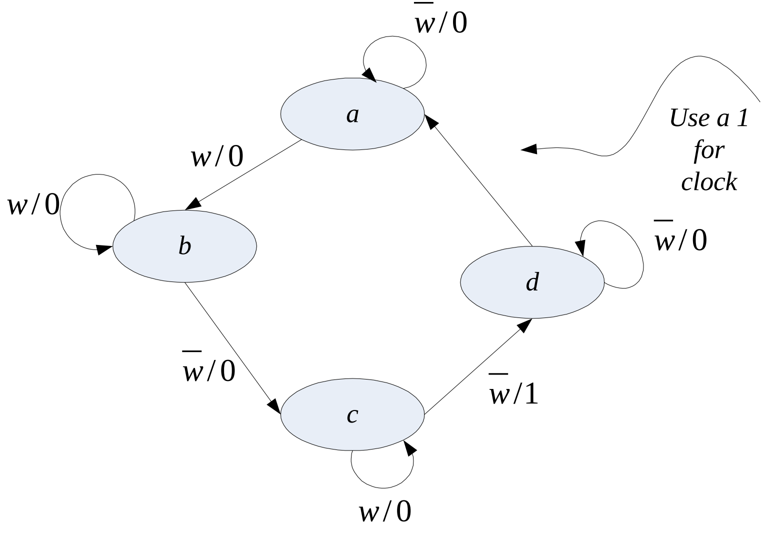
only input is count pulse, clock is implied

Output = <u>present</u> state of FF

Counter completely specified by a list of the count sequence – is the sequence of binary states it undergoes

Counter Design

- Description – Specifications
- Draw state diagram
  a. Used to show progressive states of sequence events
  b. Same as state table
  c. Present state in circle
  d. Vector shows next state
  e. Vector information
     - First character indicates input that causes change of state. Use 1 for clock
     - 2nd number(s) show output for present state. Only happens when arriving at next state, not during transition



- Make State Table
  a. This contains the same information as the state diagram, and is interchangeable.

PS	NS w=0	NS w=1	Out w=0	Out w=1
a	a	b	0	0
b	c	b	0	0
c	d	c	0	0
d	d	a	0	0

- Assign state variables
  a. Assign each desired output to a FF if have no conditional outputs (no logic)
  b. Assign each state adjacent so that you do not change more than one state variable for a single change in input – mirror code
  c. n=# state variables → $2^n = \# states$
     i. Example 4 states = 2^n → n = 2 state variables, which is the number of FFs
  d. Identify 2 FFs as Q_1Q_2, Q_AQ_B, AB, PQ, depending on author
  e. Sometimes do not use abc, but numbers

PS	Assignment	
a	00	$\overline{A}\overline{B}$
b	01	$\overline{A}B$
c	11	AB
d	10	$A\overline{B}$

- Make transition table
  a. This is simply present state – next state with state assignments
  b. Output = present state is use FF for output
- Determine FF type & assign letters to each
  a. Make column for input to each FF
  b. From excitation table for the FF, what is the input to the FF required to make transition from PS to NS. i.e. if PS Q_1=1, NS Q_1=0
  c. FF input T_1 must be 1 to cause toggle
     D FF = Next State

Inputs	Present State	Next State	Outputs	FF Input
w	Q_1Q_2	Q_1Q_2	Z_1	T_1T_2
0	$00_{(a)}$	00	0	00
1	$00_{(a)}$	01	0	01
0	$01_{(b)}$	11	0	10
1	$01_{(b)}$	01	0	00
0	$11_{(c)}$	10	0	01
1	$11_{(c)}$	11	1	00
0	$10_{(d)}$	10	0	00
1	$10_{(d)}$	00	0	10

- Draw K-map for FF input @ this present state. Where is the T FF input required from excitation table to get next state.
    d. Use PS & Input to yield FF input

Q_1Q_2 W	00	01	11	10
0	0	1	0	0
1	0	0	0	1

Q_1Q_2 W	00	01	11	10
0	0	0	1	0
1	1	0	0	0

$T_1 = \overline{w}\overline{Q_1}Q_2 + wQ_1\overline{Q_2}$ $\quad T_2 = w\overline{Q_1}\overline{Q_2} + \overline{w}Q_1Q_2$ $\quad z = wQ_1Q_2$

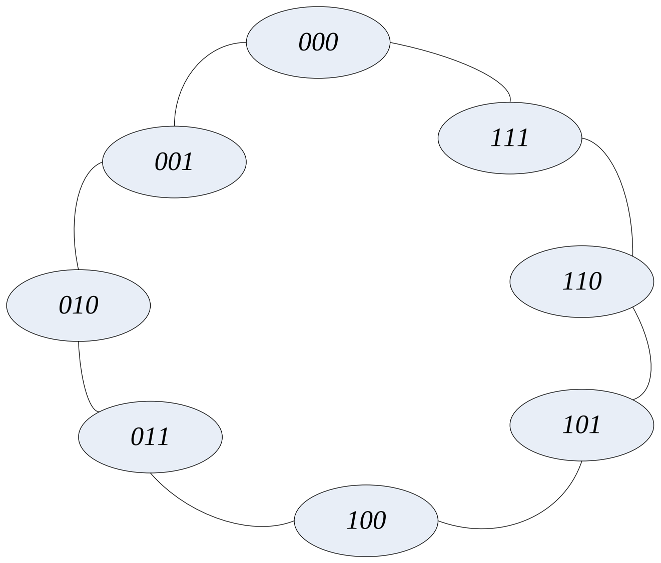
Start w/ specification – end w/ Boolean expression. Can draw diagram and wire circuit from this.

Example: 3 bit binary counter

- State diagram
- state Table (T Flip Flops)

Output = Present State = $Q_2Q_1Q_0$

PS	NS		T_2	T_1	T_0
000	001		0	0	1
001	010		0	1	1
010	011		0	0	1
011	100		1	1	1
100	101		0	0	1
101	110		0	1	1
110	111		0	0	1
111	000		1	1	1
000					



- Karnaugh Maps

T_2

Q_1Q_0 / Q_2	00	01	11	10
0	0	0	1	0
1	0	0	1	0

$T_2 = Q_1Q_0$

T_1

Q_1Q_0 / Q_2	00	01	11	10
0	0	1	1	0
1	0	1	1	0

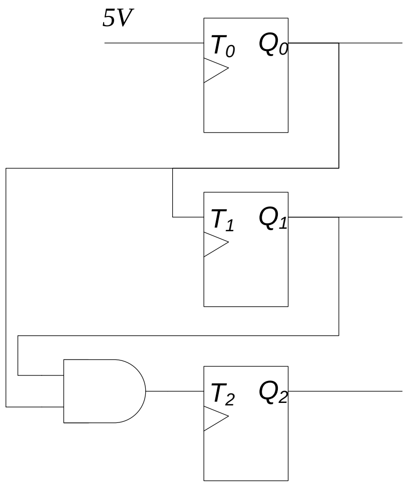
$T_1 = Q_0$

T_0

Q_1Q_0 / Q_2	00	01	11	10
0	1	1	1	1
1	1	1	1	1

$T_0 = 1$

- Draw Circuit


5V
T_0
Q_0
T_1
Q_1
T_2
Q_2

6.6.3 Sequence Detector (Random)

- Sequence consists of n bits
- number of states: Mealy = n, Moore = n+1
- Make transition table with added columns of current sequence and desired sequence. Table columns are current sequence, desired sequence, input, present state, next state, output, FF input
- Label current state & desired state
- Write desired sequence under column for all rows
- Place sequence up to this point under correct sequence
- Draw line through as many as useable of current sequence to next state in desired sequence.
- Same number of states must be crossed in both
- Output = 1 when all states are crossed for Mealy. For Moore, add one more state when all states are crossed
- Else, go to 7.

Note: it is easier to make input