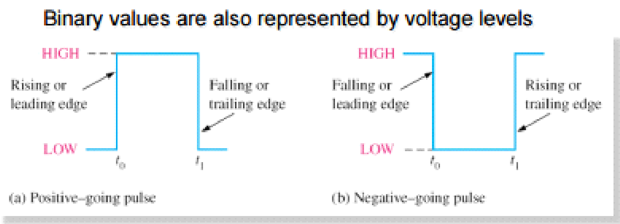
Overview - Digital Concepts

13 October 2016 10:27

Exam in Summer Term

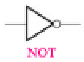
Tutorial Questions every lecture - non assessed

15 total lectures



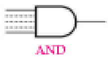
Only three basic logic operations:

- NOT



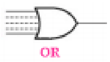
H -- L
L-- H

- AND

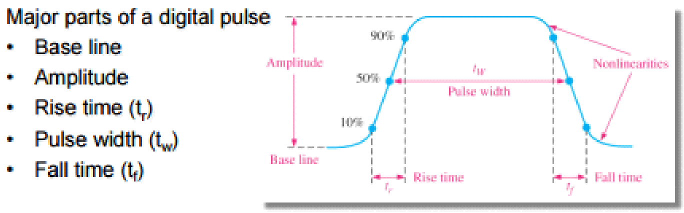


Only H -- H
Any other combo -- L

- OR



Only needs one H -- H
Only L -- L



Rise time - 10% to 90%
Fall time - 90% to 10%
Pulse width - rise 50% to fall 50%

$$\textbf{Duty cycle} = \left(\frac{t_w}{T}\right)100\%$$

Basically describes how long wave is at HIGH value
t_w = pulse width

Fixed Function IC's

1. Dual in-line package (DIP):
   - cheapest
2. Small-outline IC (SOIC):
   - Flatter pins
   - Thinner
   - Easier to fit
3. Flat pack (FP)
   - Pins can be bent/cut
   - Used in aerospace
   - Withstand higher temps
4. Plastic-leaded chip carrier (PLCC)
   - J connections pins
   - Compact
   - More pin out for a given area
5. Leaded-ceramic chip carrier (LCCC)
   - Occupies less space
6. Ball Grid Array (BGA)
   - Identical pattern on board
   - Heat solder balls - melt and connect

Data Representation

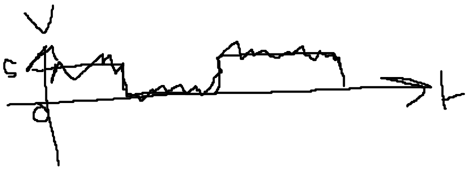
16 October 2016 18:44

- Information can be represented by a voltage level
- The simplest information is TRUE/FALSE
  - This can be represented by two voltage levels:
    - 5 Volts for TRUE
    - 0 Volts for FALSE
- A voltage signal which has only two possibilities is a BIT
  - Bit stands for Binary Digit
- Binary means: only 2 possible values

FALSE	TRUE
(0)	(1)

- Advantages of using binary representation
  - simple to implement in electronic hardware (switch)
  - good tolerance to noise

Digital - Great Noise Immunity



Q# Format

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
1	0	0	1	1	0	1	1

Binary Point

$(19.375)_{10}$

This 8-bit number is in Q3 format
- 3 bits after the binary point

Conversions:

- Decimal to Binary:

$$50_{10} = 32 + 18$$
$$= 32 + 16 + 2$$
$$= 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^1$$
$$50_{10} = 110010_2$$

- Binary to Decimal:
  - eg: Convert $(1101)_2$ into decimal
  - $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (13)_{10}$

Binary Addition - Right to Left

- 0 + 0 = 0 carry-out 0
- 0 + 1 = 1 carry-out 0
- 1 + 0 = 1 carry-out 0
- 1 + 1 = 0 carry-out 1
- 1 + 1 + carry-in = 1 carry-out 1

Hexadecimal Numbers conversions

Binary-to-hexadecimal conversion

1. Break the binary number into 4-bit groups
2. Replace each group with the hexadecimal equivalent

Hexadecimal-to-decimal conversion

1. Convert the hexadecimal to groups of 4-bit binary
2. Convert the binary to decimal
3. Or, convert directly using powers of 16

Decimal-to-hexadecimal conversion

- Repeated division by 16

Binary Coded Decimal (BCD)

- Use 4-bit binary to represent one decimal digit
- Easy conversion
- Wasting bits (4-bits can represent 16 different values, but only 10 values are used)
- Used extensively in financial applications

Grey Code: Only 1 bit changes in increasing count sequence

ASCII - 7-bit

Basic Logic Gates & Boolean Expressions

13 November 2016 18:05

● = AND
+ = OR

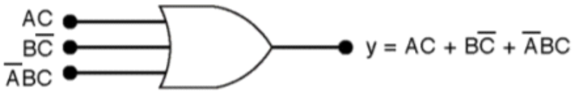
<u>Order of Precedence:</u>

- Single term inversions
- parenthesis
- **AND before OR**
- Expression Inversions

To build gates

- Block1
  - Two transistors in series
- Block2
  - Two transistors in //
- NMOS - Active HIGH
- PMOS - Active LOW

Start from *logic-circuit diagram*



Boolean Algebra

13 November 2016 18:06

Laws

Commutative
- A + B = B + A
- A ● B = B ● A

Associative
- A + (B + C) = A + B + C
- A ● (B ● C) = A ● B ● C

Distributive
- A(B+C) = AB + AC

Rules of Boolean Algebra

- A + 0 = A
- A + 1 = 1
- A · 0 = 0
- A · 1 = A

- A + A = A
- A + $\bar{A}$ = A
- A · A = A
- A · $\bar{A}$ = 0

- $\bar{\bar{A}} = A$

- **A + AB = A**
- **A + $\bar{A}B$ = A + B**
- **(A + B)(A + C) = A + BC**

<u>DeMorgan's theorem:</u>

Switch to invert of single variables
Add or remove Bar
Change operator

- $\overline{(X+Y)} = \bar{X} \cdot \bar{Y}$
  - $X + Y = \overline{(\bar{X} \cdot \bar{Y})}$
- $\overline{(X \cdot Y)} = \bar{X} + \bar{Y}$
  - $X \cdot Y = \overline{(\bar{X} + \bar{Y})}$

Universality

GATE	NAND	NOR
INV	All inputs same	All inputs same
AND	Output of first- Both input of 2nd	Invert inputs Feed into NOR
OR	Invert inputs Feed into NAND	Output of first- Both input of 2nd

Alternate Gate Representation Steps
1. Invert inputs and output of gate
2. Change Boolean Operator
   a. Except for inverter

Aim to match input and outputs
- Bubble to bubble
- Non-bubble to non-bubble

Logic Simplification & Karnaugh Map

13 November 2016 18:06

Sum of Products - AND terms into an OR gate

Product of Sums - OR terms into an AND gate

Canonical Form - every variable appears in every term

SOP to Canonical

- Term missing variable X
    - AND with $(X + \bar{X})$

Can use $f[\square] = \sum()$ to express Canonical

Sum of PRODUCTS

Number inside ()

Represents row number from truth table

Variables inside []

ORDER matters

Logic Simplification

Method 1: Algebra

Things to try

- Grouping
    - Factorisation
- Multiplication by redundant variables
    - Multiply term without X by:
        - $X + \bar{X}$
        - $1 + X$
    - Doesn't alter logic
- Apply DeMorgan's theorem
    - Useful for expressions with lots of inversions

Method 2: Karnaugh Maps

Normally works by filling in table with a SOP expression

Grey code ordering

Avoid with more than 4 variables

- With 5, use two maps, and group over 3D

1. Find pairs
2. Group - use minimum number of groups
    a. X - don't care - treat as 0 or 1 for min groups
3. New SOP formed

Can be done to POS expressions as well

1. Group 0's to produce a SOP
2. Make a POS with same terms from above
3. Apply DeMorgan's theorem to each term

More Gates and their Applications

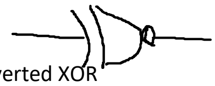
13 November 2016 18:07

XOR gate - HIGH when only 1 input is HIGH

XOR used for selectable inversion

When B HIGH, A inverted

When B LOW, A normal



XNOR gate - HIGH when inputs are same - inverted XOR

XNOR used in PARITY generator and checker

Other Representation

Input on left

Output on the right

Common inputs on top control block

If numbered

- Only perform action when asserted
- With corresponding input/output in lower blocks

Label	Name
EN	Enable
G	AND
V	OR
N	Invert
S	Set
R	Reset

Multiplexer

- Select inputs select one of the Data inputs for output
  - **# Data inputs = 2 ^ # select lines**
- Each input is **AND'd with all select lines**
- **Output = OR of all the above terms**

Implementing Logic in a MUX

- Select lines = Variables
- Data inputs connected to GND or +Vcc depending on Output of combination of variables
  - GND if output should be 0
  - +Vcc if output should be 1
  - Check from Truth Table

Demultiplexer

- Only one Data input
- Select lines choose which output(s) to send input to
- # Outputs = 2 ^ # Select lines
- Implement:
  - Data line is AND'd with all combinations of select lines
  - Each output comes from each AND gate

Signed Numbers & Arithmetic Circuits

13 November 2016 18:08

<u>2's Compliment</u>

- Invert and add 1
- MSB has weighting of -1
- Can use adder circuits to perform subtraction
  - Can use unsigned Adder HW

<u>Sign Extension</u>

- Duplicate Sign-bit into new MSB's

Add 2 n-bit Signed numbers and avoid overflow

- Sign extend to n+1 bits
- Use n+1 bit adder

Multiply a signed number by **-1**

- Invert and +1
- Doesn't work for smallest number

LSR	Divide by 2
LSL	Multiply by 2
ASR	Signed Divide by 2

<u>Half-Adder</u>

- Sum = A XOR B
- Carry = AB

<u>Full-Adder</u>

- Sum = A XOR M
  - M = B XOR Cin
- C = OR of:
  - AB
  - ACin
  - BCin

- From 2 Half-Adders
  - First produces HSum and HCarry
  - Second:
    - Sum = HSum + Cin
    - Carry = HCarry OR SecondCarry

<u>Parallel Adder</u>

- One Full-Adder per bit
- Carry out from bit n = Carry in for bit n+1

Propagation delay = n x delay for one adder

<u>To make this Subtract</u>

- Set **first Carry in = 1**
- **Invert bits** for Number to be subtracted

<u>Comparators</u>

- Output HIGH when **corresponding bits equal**
- Use a **XNOR** gate

To **extend to N-bits**

- Use XNOR gates for each pair of pits
- AND the XNOR outputs

<u>Decoders</u>

- Binary Decoder

  Output for Decimal value is only Asserted when inputs match binary representation of the decimal value

- Also Have:
  - BCD - Decimal
  - BCD - 7-Segment

ROM & Programmable Logic Devices

31 March 2017 02:48

Memory Cell - stores 1-bit

To Address ROM - A x B
- A = number of cells
- B = word length
- $2^N = A$
    - N = number of control lines
    - Half for row
    - Half for columns

ROM cell
- MOS transistor for switch
- Row control line set to as signal for Transistor
- When row line asserted, all row cells release stored value to respective columns

PLD's

- Implement canonical SOP expressions
- Reduces chip count

Implementing Logic
- N # of Inputs
- B # of Outputs
- From Truth Table, most right variable = LSB input

Static hazards

0 hazard: 0 -1 -0
1 hazard: 1-0-1

In K-Map, groups that do not overlap are potential HAZARDS
- To avoid, add group that overlaps
- Redundant logic, but prevents hazard

Flip-flops

31 March 2017 02:48

Three main types

- Reset - Set (RS / SC)
- JK
- D

Clock Edge

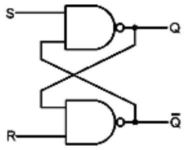
- If **Bubble** on CLK input
  - Negative-going transition

- Hold input stable
  - On clock edge
  - **Setup time** Before edge
  - **Hold Time** After edge
  - Allows Sequential FF

Asynchronous Inputs

- Asserted **CLEAR - Q = 0**
- Asserted **PRESET - Q = 1**
- **Both Asserted - normal operation**

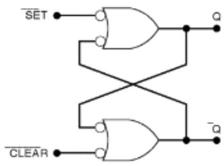
NAND Gate Latch: **Set - Reset**



ACTIVE LOW

Set	**Reset**	**Output**
1	1	**No Change**
0	1	**Q = 1**
1	**0**	**Q = 0**
0	0	**INVALID**

NOR Gate Latch: **Set - Reset**



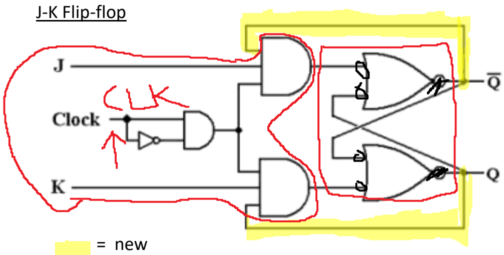
ACTIVE HIGH

Set	**Reset**	**Output**
0	0	**No Change**
1	0	**Q = 1**
0	**1**	**Q = 0**
1	1	**INVALID**

To make FF clocked,

- Use **edge detector** circuit -> CLK*
- **AND** CLK* with S/R
  - Use these as inputs for S R in circuit

J-K Flip-flop



= new

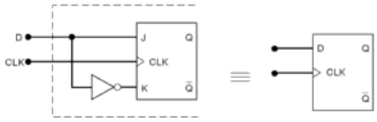
J(S)	K(R)	CLK	Q
0	0	|	No Change
1	0	|	1
0	1	|	0
1	**1**	|	**Toggle**

D Flip-Flop

- Same as JK but only two possible inputs

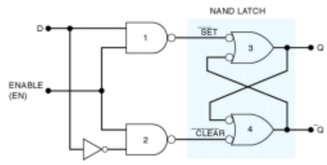
D	**Q**
0	0
1	1

Clocked D FF



D Latch - Enable signal

EN = 0: Q is latched to its value
EN = 1: Q = D - **D passes through**



EN	D	Q
0	X	Q_0
1	0	0
1	1	1

State Diagrams

31 March 2017 02:49

<u>Moore Model</u>

<u>State Diagram</u>

- Circle for each state
  - Inside circle:
    - State Number - starting from 1
    - Associated Output
    - e.g. 1/0 - State #1, output = 0

- Arrows for each transition between states
  - Label with required inputs
  - e.g. 11 - A = 1, B = 1

<u>State Table</u>

- One Row for each State
- Input Columns in Grey code order

<u>Assigned State Table</u>

- Output associated with that state
- Replaces the state in the table
- Same contents as a K-MAP
  - Row is Present Output
  - Next Output = Boolean Expression

<u>Mealy Model</u>

<u>State Diagram</u>

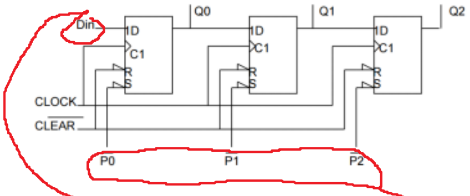
- More useful when output depends on inputs, not previous state
- **Circles - only labelled with state number**
- Arrow Labels
  - Inputs / Next Output
  - e.g. 10/1 - input of 1 and 0 gives output of 1

Circuits Using Flip-flops

04 April 2017 13:35

Register
- Parallel D FF's
- Global RESET line

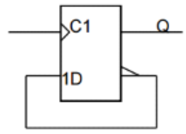
Shift Register



Serial FF
Q connected to next D input

- Can input **Serial data at D0**
  - Wait N cycles to shift data N-bits
- **Parallel port** to load // data
  - 1 cycle to shift by 1-bit

Divide **CLK** by 2
Connect $\bar{Q}$ to D



Asynchronous Counter

- String together Divide by 2 circuits
- Previous $\bar{Q}$ is next FF CLK
- Take Q as bit output

Low Max Clock Freq
- Can be used to divide CLK by a power of 2

Synchronous Counter

- FF store value
- Use separate Adder to Increment / Decrement

Finite State Machines

04 April 2017 13:35

FSM By Combinatorial Logic

1. Start with Transition Table of FF used
2. Construct State Transition Table
3. K-Map for each Output of CL block, from its inputs
4. Extract Boolean Expressions form K-Maps
5. Can use X for don't cares or map to a base state

FSM By ROM

1. Cells Contain State Outputs
2. Use current State as Address
   a. Cell pointed to contains next State

Using Shift Register for a DELAY

- D Input Synchronous
  - Q0 delayed by 1 cycle
  - QN delayed by 1 + N cycles
  -
- D Input Asynchronous
  - Q0 delayed by UP TO 1 cycle = T_0
  - QN delayed by T_0 + N cycles

Two States are considered Equivalent if

- Have same next state
- Have same current output

IF removing a state from diagram, ensure no undefined State
- Make it redundant and point to Base state no matter input