Digital Logic
PhD Qualifying Exam
Review Session

December, 2016

Boolean Algebra

- **Algebra with 0's and 1's**
  - X + 0 = X
  - X + 1 = 1
  - X * 1 = X
  - X * 0 = 0

- **Idempotent Laws**
  - X + X = X
  - X * X = X

- **Complement Laws**
  - X + /X = 1
  - X * /X = 0

Boolean Algebra

- **Dual**:
  - 1 → 0
  - + → -

X + 0 = X → X * 1 = X

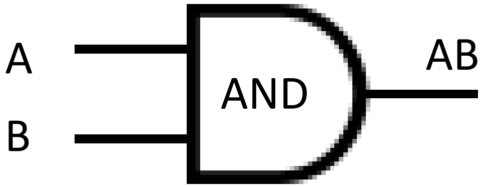
X + 1 = 1 → X * 0 = 0

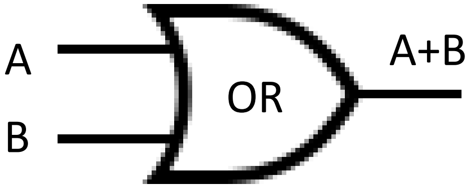
X + /X = 1 → X * /X = 0

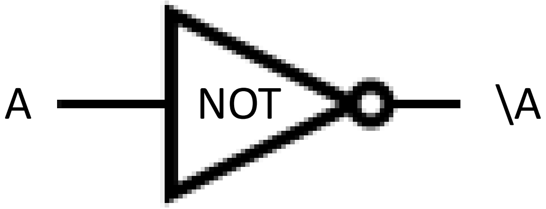
- **DeMorgan’s Laws**
  - \(X + Y + Z) = \X * \Y * \Z
  - \(X * Y * Z) = \X + \Y + \Z
- **Consensus Theorem**
  - XY + **YZ** + \XZ = XY + \XZ

Proof - example

Logic Gates







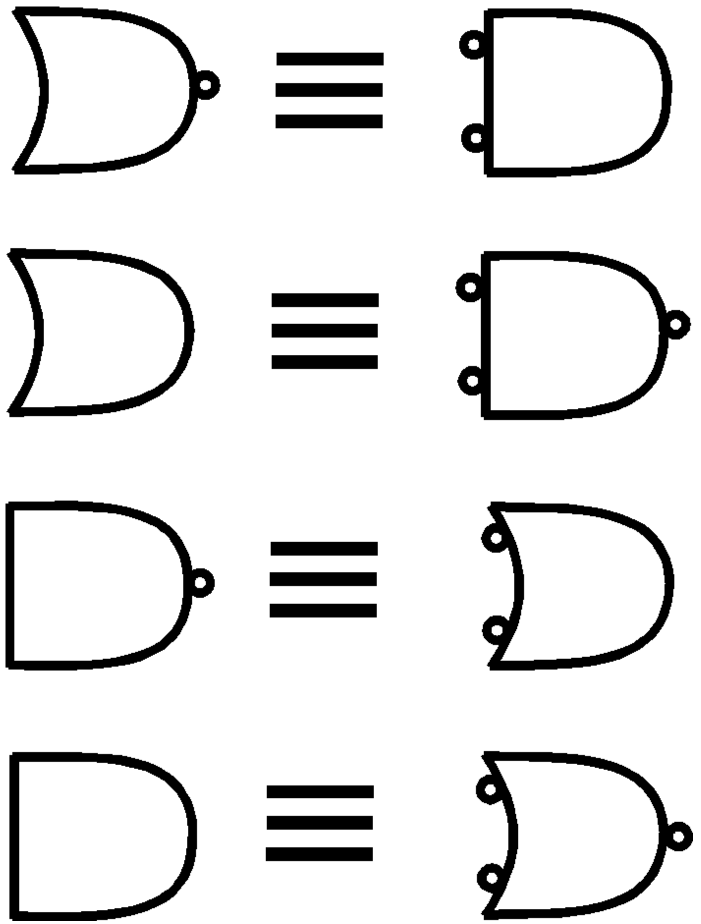
INPUT		**OUTPUT**		
A	B	A **AND** B	A **OR** B	**NOT** A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

More Logic Gates

- Active High
  - High Voltage (1) is TRUE

- Active Low
  - Low Voltage (0) is TRUE

- Active Low to bubble = match
- Active High to bubble = mismatch
- Bubbles do not change logic



Number Systems

- **Base 10**: 541 = **5***10^**2** + **4***10^**1** + **1***10^**0**

- **Base 2:** 0101 = **0***2^**3** + **1***2^**2** + **0***2^**1** + **1***2^**0**

Conversion between systems

- **Decimal to Binary**
  1. Brute Force -- “Count it out”
  2. Divide by 2 – remainder becomes binary (least to most significant)

- **Decimal to Hex**
  1. Convert to Binary then groups of 4 bits
  2. Divide by 16 -- remainder becomes hex (least to most significant)

- **Decimal to Octal**
  1. Convert to Binary then groups of 3 bits
  2. Divide by 8 -- remainder becomes octal (least to most significant)

Signed Number Representations

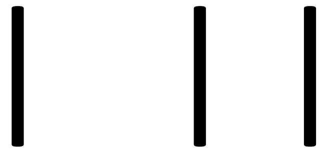
- Signed Magnitude:
  - MSB gives sign
  - 1000 0101 = -5
- 1’s Complement:
  - if MSB is 1 – flip bits and apply minus sign
  - If MSB is 0 – do nothing, positive
  - 1111 1100 (flipped = 0000 0011) = -3
- 2’s Complement
  - if MSB is 1 – flip bits, add 1, and apply minus sign
  - 1111 1101 (flipped + 1 = 0000 0011) = -3

Alternate 2's Complement Solution

- 2's Complement:
  - 1011

1. Flip Bits: 0100
2. Add 1: 0101
3. Interpret: -5

- 2's Complement:
  - 1011



-8 + 2 + 1 = -5

Arithmetic

- Think back to basic arithmetic in base 10

- Let’s just do some problems

SOP, POS, and K-Maps

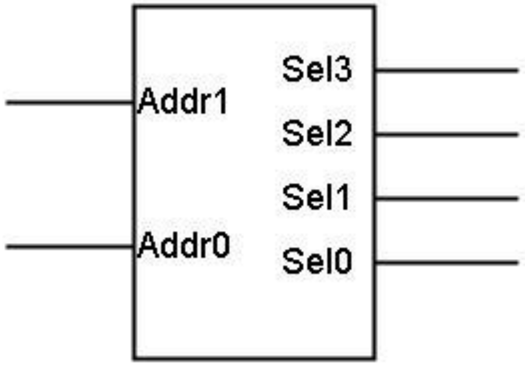
- SOP
  - XY + YZ

- POS
  - (X+Y)(Y+Z)

- K-Map
  - Truth table to K-Map form
  - Equation to K-Map

Decoder

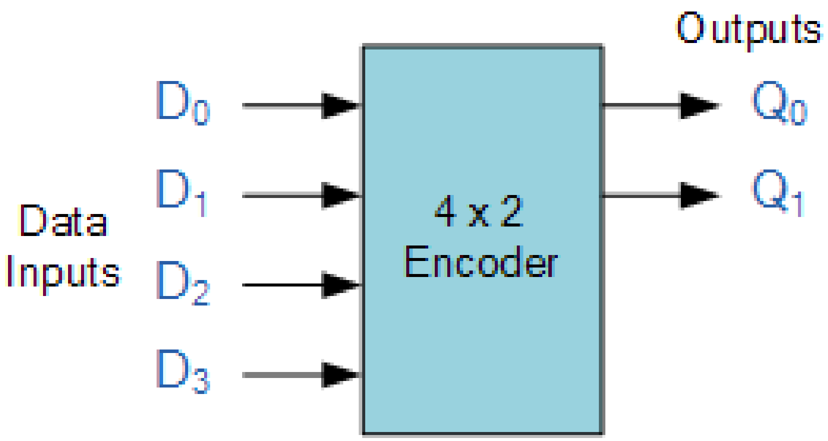
- Maps binary input to decimal output
  - /A1 * /A0 → Sel0
  - /A1 * A0 → Sel1
  - A1 * /A0 → Sel2
  - A1 * A0 → Sel3

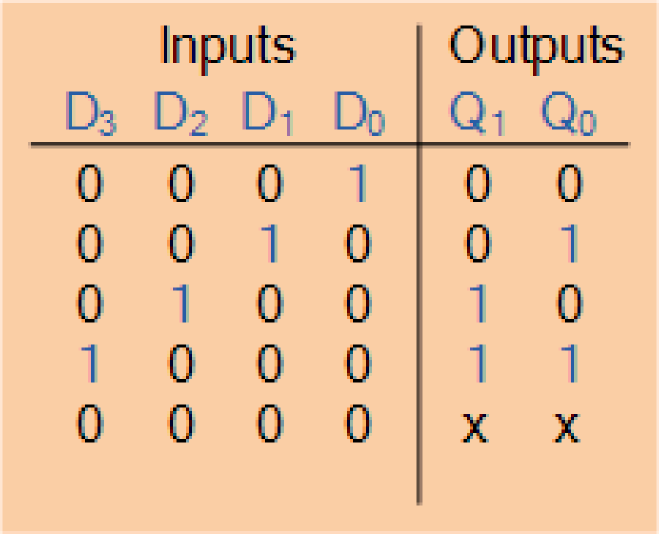


2 to 4 Decoder

Encoder

- Maps decimal input to binary output

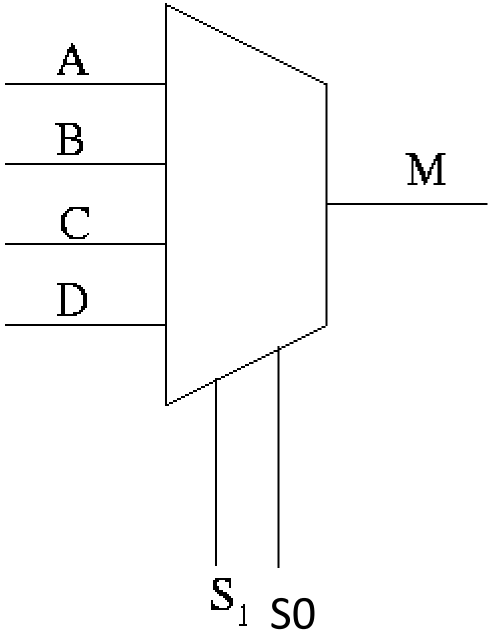




Inputs				Outputs	
D_3	D_2	D_1	D_0	Q_1	Q_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1
0	0	0	0	x	x

Mux

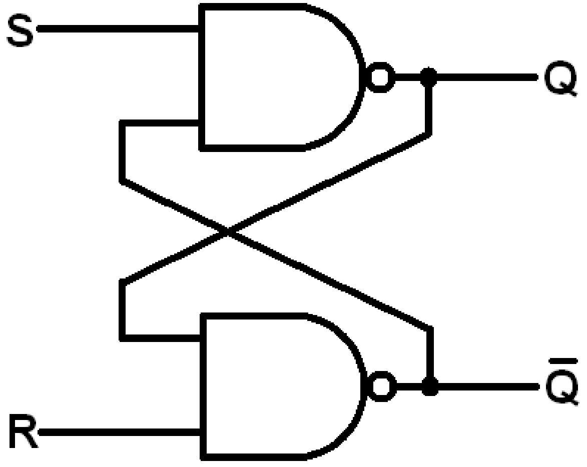
- Multiple input, single output selection device
  - M = \S1 * \S0 * A
    + \S1 * S0 * B
    + S1 * \S0 * C
    + S1 * S0 * D

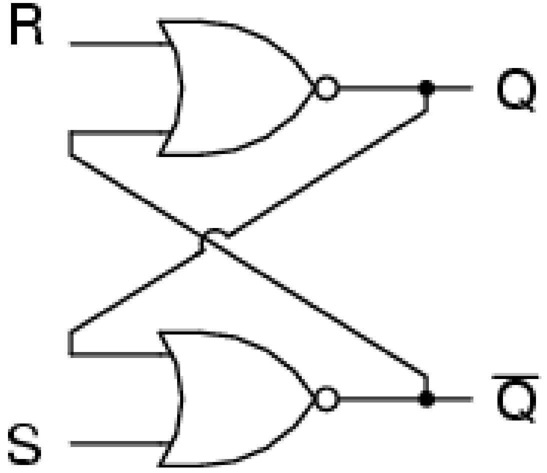


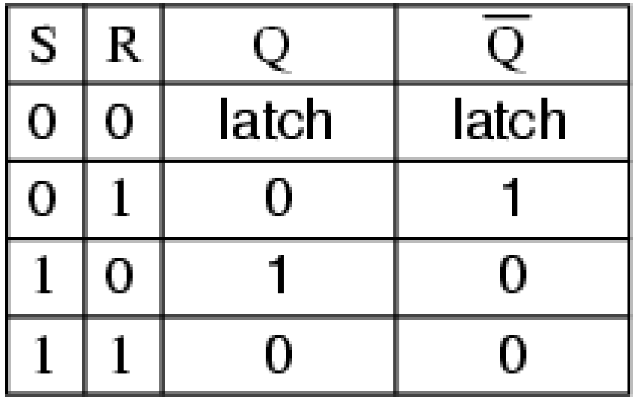
We can implement logic equations using multiplexers - example

SR Latch

- NOR and NAND configurations

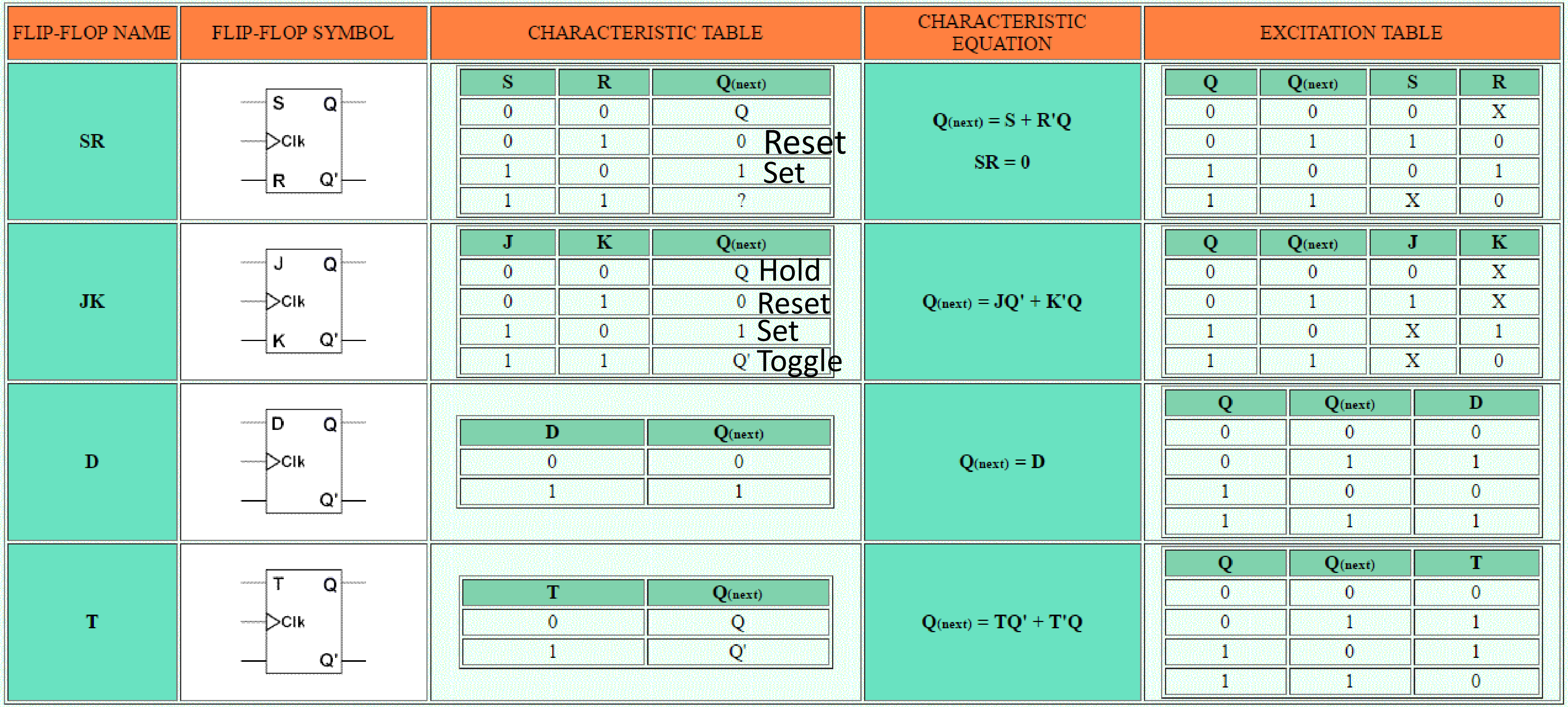






S	R	Q	$\overline{Q}$
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

Flip-Flops



<table>
<tr><th></th><th>FLIP-FLOP NAME</th><th>FLIP-FLOP SYMBOL</th><th>CHARACTERISTIC TABLE</th><th>CHARACTERISTIC EQUATION</th><th>EXCITATION TABLE</th></tr>
<tr><td></td><td>SR</td><td>S Q
Clk
R Q'</td><td><table><tr><th>S</th><th>R</th><th>$Q_{(next)}$</th><th></th></tr><tr><td>0</td><td>0</td><td>Q</td><td></td></tr><tr><td>0</td><td>1</td><td>0</td><td>Reset</td></tr><tr><td>1</td><td>0</td><td>1</td><td>Set</td></tr><tr><td>1</td><td>1</td><td>?</td><td></td></tr></table></td><td>$Q_{(next)} = S + R'Q$
$SR = 0$</td><td><table><tr><th>Q</th><th>$Q_{(next)}$</th><th>S</th><th>R</th></tr><tr><td>0</td><td>0</td><td>0</td><td>X</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>X</td><td>0</td></tr></table></td></tr>
<tr><td></td><td>JK</td><td>J Q
Clk
K Q'</td><td><table><tr><th>J</th><th>K</th><th>$Q_{(next)}$</th><th></th></tr><tr><td>0</td><td>0</td><td>Q</td><td>Hold</td></tr><tr><td>0</td><td>1</td><td>0</td><td>Reset</td></tr><tr><td>1</td><td>0</td><td>1</td><td>Set</td></tr><tr><td>1</td><td>1</td><td>Q'</td><td>Toggle</td></tr></table></td><td>$Q_{(next)} = JQ' + K'Q$</td><td><table><tr><th>Q</th><th>$Q_{(next)}$</th><th>J</th><th>K</th></tr><tr><td>0</td><td>0</td><td>0</td><td>X</td></tr><tr><td>0</td><td>1</td><td>1</td><td>X</td></tr><tr><td>1</td><td>0</td><td>X</td><td>1</td></tr><tr><td>1</td><td>1</td><td>X</td><td>0</td></tr></table></td></tr>
<tr><td>Delay</td><td>D</td><td>D Q
Clk
Q'</td><td><table><tr><th>D</th><th>$Q_{(next)}$</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table></td><td>$Q_{(next)} = D$</td><td><table><tr><th>Q</th><th>$Q_{(next)}$</th><th>D</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td></tr>
<tr><td>Toggle</td><td>T</td><td>T Q
Clk
Q'</td><td><table><tr><th>T</th><th>$Q_{(next)}$</th></tr><tr><td>0</td><td>Q</td></tr><tr><td>1</td><td>Q'</td></tr></table></td><td>$Q_{(next)} = TQ' + T'Q$</td><td><table><tr><th>Q</th><th>$Q_{(next)}$</th><th>T</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td></tr>
</table>

Design a Counter Example

- Count 7, 0, 3 using J-K Flip Flop and T-FF

State Machines

- Moore Machines
  - Output only depends on current state

- Mealy Machine
  - Output depends on current state and input
  - Asynchronous

- Example: Detect sequence 010*1

ASM Diagrams

- Rectangle:
  - State box
  - Outputs go inside
- Diamond
  - Decision box, follows state box
  - Conditional Branch
  - Inputs go inside
- Oval:
  - Conditional (Mealy) outputs
  - Outputs go inside, asynchronous

- Examples:
  - Detect sequence 010*1
  - Washing Machine

Registers

- Just a collection of Flip-Flops

- Example:
  - Design 4-bit shift register with the following specs
    - Global enable
    - Asynchronous reset to 1010
    - We can give 4-bit input
    - Shift left and wrap
    - Shift right and sign extend