

Kuwait University
College of Engineering and Petroleum



جامعة الكويت
KUWAIT UNIVERSITY

ME319 MECHATRONICS

PART I: THE BRAINS – MICROCONTROLLERS, SOFTWARE AND DIGITAL LOGIC

LECTURE 3: DIGITAL LOGIC

Spring 2021

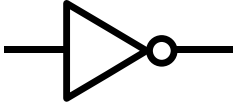

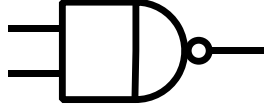




Ali ALSaibie

Lecture Plan

- Objectives:
 - Review the basics of Logic Gates
 - Review the basics of bitwise logical operations in C/C++



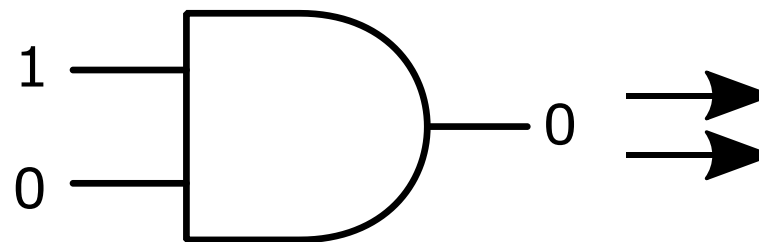
Logic Gates Truth Table

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR																																																																																																
Alg. Expr.	\bar{A}	AB	\overline{AB}	$A + B$	$\overline{A + B}$	$A \oplus B$	$\overline{A \oplus B}$																																																																																																
Symbol																																																																																																							
True Table	<table border="1"> <thead> <tr> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> </tr> </tbody> </table>	A	X	0	1	1	0	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	B	A	X	0	0	0	0	1	0	1	0	0	1	1	1	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</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> </tbody> </table>	B	A	X	0	0	1	0	1	1	1	0	1	1	1	0	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <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>1</td> </tr> </tbody> </table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	1	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	0	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <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> </tbody> </table>	B	A	X	0	0	0	0	1	1	1	0	1	1	1	0	<table border="1"> <thead> <tr> <th>B</th> <th>A</th> <th>X</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> </tr> </tbody> </table>	B	A	X	0	0	1	0	1	0	1	0	0	1	1	1
A	X																																																																																																						
0	1																																																																																																						
1	0																																																																																																						
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	1																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	0																																																																																																					
0	1	1																																																																																																					
1	0	1																																																																																																					
1	1	0																																																																																																					
B	A	X																																																																																																					
0	0	1																																																																																																					
0	1	0																																																																																																					
1	0	0																																																																																																					
1	1	1																																																																																																					

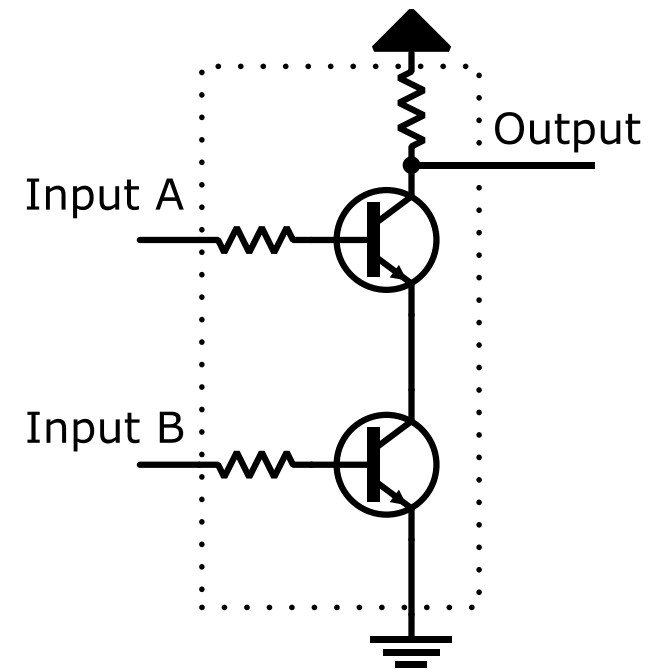


Digital Logic

- In its simplest form, a computer is composed of
 - Transistors: states (1 or 0), and
 - Logic gates: operations on 1s and 0s
- Logic gates are built from electrical circuit



AND Gate Symbol



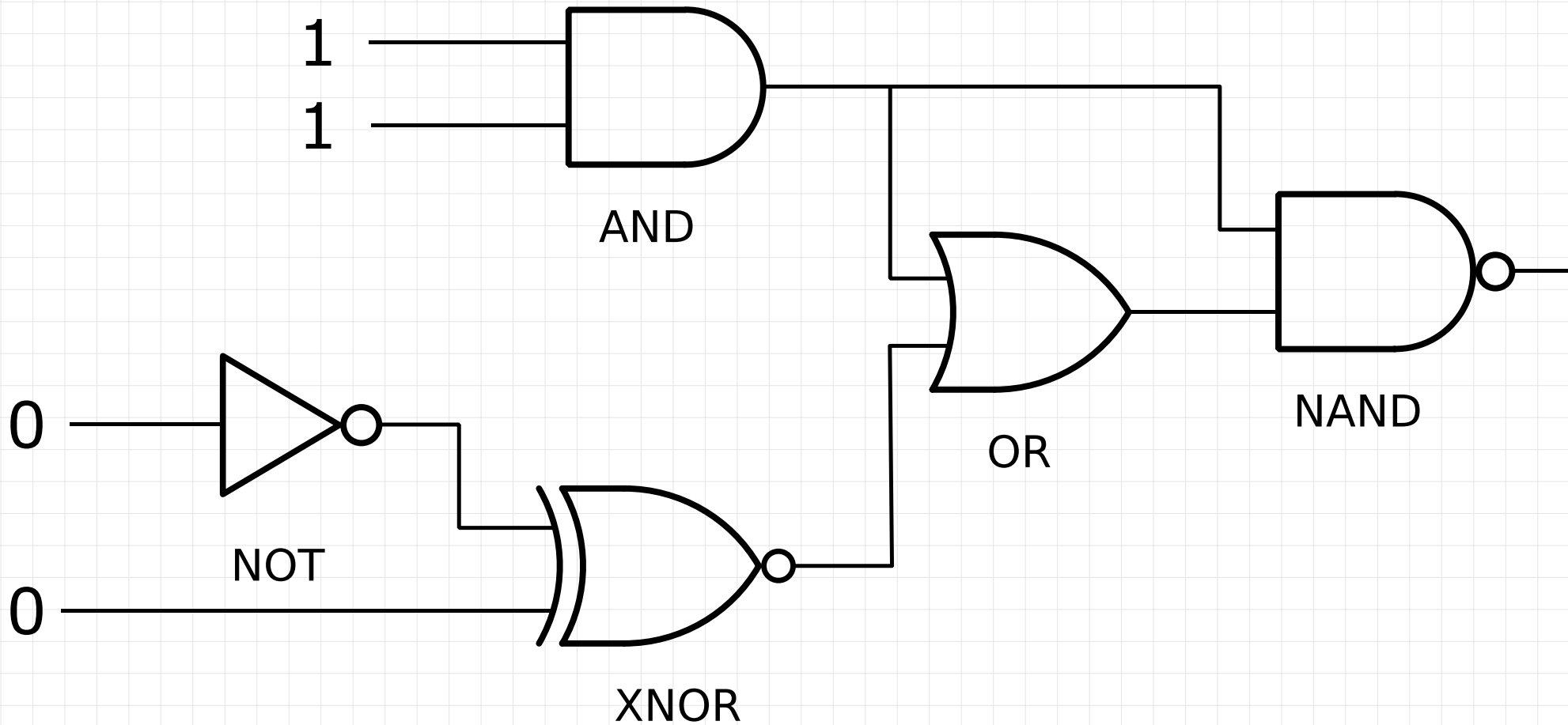
Electronic AND Gate



The building blocks of computers

- Millions of transistors and logic gates build memory and logic to form
 - Basic operations
 - Programming Languages translate complex algorithms to basic operations
- When you program in: `int a = 8 + 2;`, this code ultimately translates to:
 - Memory being allocated (transistor states being changed),
 - Current passing through transistors: **1** (or not passing through: **0**) and going through a combination of logic gates to perform:
 - Memory assignment, retrieval, addition, allocation
- From basic add / subtract / store / compare etc., we stack higher level algorithms: complex operating systems, flight control systems, financial trading algorithms, etc.





Math Shortcuts

- When working with unsigned binary numbers:
 - Shifting the bits one location to the right, halves the number
 - Shifting the bits one location to the left, doubles the number
- This technique can be used to produce faster mathematical power operations, but can cause loss of precision or overflow

$$00001100_2 = 12_{10} \Leftrightarrow 00000110_2 = 6_{10} \Leftrightarrow 00000011_2 = 3_{10}$$

Left Shift: $11000000_2 = 192_{10} \Rightarrow 10000000_2 = 64_{10}$: incorrect due to overflow

$$\text{Right Shift: } 00000011_2 = 3_{10} \Rightarrow 00000001_2 = 1_{10}$$

- **Overflow**: storing a number larger than what memory location can handle



Bitwise Operations

- Manipulating memory and variables on a bit level is essential in embedded programming

Operator	Name	Description
&	Bitwise AND	Performs AND operation on each corresponding bit of two arguments and returns result
	Bitwise OR	Performs OR operation on each corresponding bit of two arguments and returns result
^	Bitwise Exclusive OR	Performs Exclusive OR operation on each corresponding bit of two arguments and returns result
~	Bitwise NOT	Performs a NOT operation on all the bits of an argument and returns the result
<< or >>	Right shift or Left shift	Shift bits of argument 1 right/left by argument 2 places



Bitwise Operations - Logical

- Numbers can be expressed in binary, hex as well as decimal.
- Bit-level logic can be performed as shown.

```
#include <cstdint>
#include <iostream>

using namespace std;

int main(int argc, char* argv[]) {

    uint8_t a = 0b00001000;
    uint8_t b = 0b10001111;
    uint8_t c = 0xF7; /* 11110111 */

    uint8_t r1 = a & b; /* AND operation, result: 00001000 */
    uint8_t r2 = a | c; /* OR operation, result: 11111111 */
    uint8_t r3 = a ^ b; /* XOR operation, result: 10000111 */
    uint8_t r4 = ~c;    /* NOT operation, result: 00001000 */

    r1 |= b; /* OR operation, result: 10001111 */
    return 1;
}
```

Bitwise Operations – Bit Shifting

- It is often convenient, especially in embedded programming, to shift bits left or right.

```
#include <cstdint>
#include <iostream>

using namespace std;

int main(int argc, char* argv[]) {

    uint8_t a = 0b00000100;
    uint8_t b = 0b10001111;
    uint8_t c = 0xF7; /* 11110111 */

    uint8_t r1 = a >> 2; /* Shift operation, result: 00000001 */
    uint8_t r2 = b << 4; /* Shift operation, result: 11110000 */
    uint8_t r3 = (a<<1) | c; /* Shift + OR NOR operation, result: 11111111 */

    bitset<8> bin(r3);
    cout << hex << bin << endl;
    return 1;
}
```

Manipulating Individual Bits

- Given the following binary number: 10101111
- What if you wanted to change the 4th bit to 1: 10101111
 - We can assign a new number, but that is not practical
 - Instead, we perform a bitwise logical operation

```
#include <stdint>
int main(int argc, char* argv[]) {

    uint8_t a = 0b10101111;
    /* Change the 4th bit to 1 */
    a = 0b10111111; /* You can replace the whole number by a new one */
    /* Or you can manipulate the individual bit, by performing an OR operation */
    a |= 0b00010000;
    /* 0b00010000 is called a bitmask, we can use a shift operator to create a bitmask */
    a |= (1<<4); /* OR operation with 1 shifted to the left 4 bits: 00010000 */
    return 1;
}
```

Manipulating Multiple Bits

- We can extend the bitwise logical operations to multiple bits
 - Examples of: Setting, Clearing and Toggling Bits

```
#include <cstdint>
#include <iostream>
#include <bitset>
using namespace std;
int main(int argc, char* argv[]) {

    uint8_t a = 0b00101000;
    /* Change the 0th and 4th bits to 1 */
    a |= (1<<4) | 1; /* OR Operation with Bitmask: 00010000 | 00000001 = 00010001 */
    /* Change the 3rd and 5th bits to 0 */
    a &= ~((1<<5) | (1<<3)); /* AND Operation with NOT of bitmask */
    /* Toggle the 7th bit */
    a ^= (1<<7); /* XOR: Exclusive OR Operation with the bitmask */

    bitset<8> binarynumber(a);
    cout << binarynumber << endl; /* Output: 10010001 */
    return 1;
}
```

Complete the following code:

```
int main(int argc, char* argv[]) {  
  
    uint8_t a = 0xF0;  
    /* Toggle all the bits of a */  
  
    a  
  
    /* Set the 1st nibble (first 4 bits) */  
  
    a  
  
    /* Clear the 2nd nibble */  
  
    a  
  
    return 1;  
}
```

