Final Exam Study Guide

Arturo Salinas-Aguayo
CSE 2301: Principles and Practice of Digital Logic Design
Dr. Mohammad Khan, Section 003L-1248
Electrical and Computer Engineering Department

College of Engineering, University of Connecticut
Coded in LaTeX

Contents

1 Exam 1: From the Top **2**
1.1 Topical Guide Objectives 2
1.2 Number Systems and Conversion 2
1.2.1 Basic Binary Arithmetic 2
1.2.2 Decimal to Binary: 3
1.2.3 Decimal Conversion and IEEE 754 4
1.2.4 Radix and Other Systems 6
1.2.5 XS3, BCD, and Gray Code 7
1.3 Boolean Algebra 8
1.3.1 Basic Laws 8
1.3.2 Conversion Using DeMorgan's Theorem 8
1.4 Karnaugh Maps 9
1.5 Circuits 11

2 Exam 2: Getting Hairy **12**
2.1 Topical Guide Objectives 12
2.2 Vocabulary 13
2.3 Static Hazards 13
2.4 Multiplexers and Demultiplexers 13
2.5 Encoder and Decoders 14
2.6 Parity and Hamming Codes 14
2.7 Adder and Subtractor Circuits 15
2.8 A Priority Encoder/Comparator 16

3 Exam 3: Sequential Circuits **17**
3.1 Topical Guide Objectives 17

1 Exam 1: From the Top

1.1 Topical Guide Objectives

1. Use Vocabulary such as:
   - Ohm's Law: $V = IR$
   - Kirchoff's Current Law: $\sum I_{in} = \sum I_{out}$
   - Kirchoff's Voltage Law: $\sum V_{in} = \sum V_{out}$
   - TTL: Transistor-Transistor Logic (Low 0V-2V, High 2-5V)
   - CMOS: Complementary Metal-Oxide-Semiconductor (Low 0-1.5V, High 3.5-5V)
   - VLSI: Very Large Scale Integration (Typically 100,000+ transistors)
   - LSI: Large Scale Integration (Typically 1,000-100,000 transistors)
   - MSI: Medium Scale Integration (Typically 100-1000 transistors)
   - SSI: Small Scale Integration (Typically 10-100 transistors)
   - Radix: Base of a number system
   - Radix Economy: The number of bits required to represent a number
   - Essential Prime Implicant: A prime implicant that covers a minterm not covered by any other prime implicant
   - Secondary Prime Implicant: A prime implicant that covers a minterm already covered by another prime implicant
2. Be able to convert between different Number Systems including Decimal, Binary, Octal, Hexadecimal, 1's and 2's Complement.
3. Understand how XS3 (Excess-3), BCD, and Gray code encoding work.
4. Be able to perform basic arithmetic on above number systems.
5. Be able to reduce Boolean Logic algebraically
6. Be able to use DeMorgan's Theorem to simplify Boolean Logic and switch to pure NAND or NOR gate implementations.
7. Be able to set up and solve Karnaugh Maps up to 5 variables

1.2 Number Systems and Conversion

1.2.1 Basic Binary Arithmetic

This is very similar to standard math in decimal, except that you only have two digits to work with, 0 and 1. The rules are the same, but the carry is a bit different. For example:

Example 1

$$\begin{array}{r} 1101 \\ +1011 \\ \hline 11000 \end{array}$$

Overflow - Occurs when adding two positive numbers produces a negative answer or if adding two negative numbers gives a positive answer.

In 2's Compliment, an overflow occurs if an donly if the carry out of the sign position is not equal to the carry into the sign position.

1.2.2 Decimal to Binary:

Example 2 Given:

$$37_{10}$$

Convert to Binary Sign Magnitude and 2's Compliment in 8-bit binary.

Recall: Sign Magnitude simply utilizes the MSB (Big Endian) to denote the sign of the number.

Start by appending the MSB with the sign bit. 0 is positive, 1 is negative.

$$0XXXXXXXX$$

$$\begin{array}{rl} 2/37 & \\ 2/18 & \text{Remainder: } 1 \\ 2/9 & \text{Remainder: } 0 \\ 2/4 & \text{Remainder: } 1 \\ 2/2 & \text{Remainder: } 0 \\ 2/1 & \text{Remainder: } 0 \\ 0 & \text{Remainder: } 1 \end{array}$$

This results in the following binary number recall start from the bottom and work your way up when working with whole numbers (ie not decimal values).

$$100101_2$$

But this is still not finished, the question asks for 8 bits, the arithmetic produced 6 bits, and we already have our MSB from the signed operation, so we fill in one more bit via *sign extension.* Propogate the sign bit until you reach the desired bits.

$$00100101_2$$

For the 2's Compliment, we start at our six bit result from above. You flip the bits, that is, 1 becomes 0, and 0 becomes 1, why? Because... that's how you do it.

$$011010_2$$

Add 1_2

$$011011_2$$

Extend the sign.

$$00011011_2$$

Example 3 Let's move quicker now.
Given:

$$-12_{10}$$

Negative, fill in MSB.

$$1XXXXXXX$$

2/12	
2/6	Remainder: 0
2/3	Remainder: 0
2/1	Remainder: 1
0	Remainder: 1

Signed Magnitude with sign extension:

$$11111100_2$$

2's Compliment:

$$11110100_2$$

Take a moment to convince yourself you can do this.

1.2.3 Decimal Conversion and IEEE 754

Example 4 Decimal Conversion
Given:

$$-0.625_{10}$$

Convert to Binary.

Instead of dividing until we reach 0, we multiply by the Radix until we no longer conatin a decimal number to multiply.

$0.625 \times 2 = 1.250$	Remainder: 1
$0.250 \times 2 = 0.500$	Remainder: 0
$0.500 \times 2 = 1.000$	Remainder: 1

This gets filled in the opposite direction. The first operation is our MSB now instead of the LSB as with whole numbers.

$$.625_{10} = .101_2$$

Example 5 Let's now extend this and convert this number to the IEEE 754 32-bit single precision format standard.

Recall:

- Sign Bit - The first bit. A simple 0 or 1 as explained in Example 1
- Exponent - The next 8 bits represent the exponent in a biased form. A bias is added to the actual exponent to ensure that both positive and negative exponents can be represented as unsigned binary numbers. For single-precision, the bias is 127. This means:

  $$\text{Stored Exponent} = \text{Actual Exponent} + 127$$

  For example, an actual exponent of 3 would be stored as $3 + 127 = 130$ in binary. Similarly, an actual exponent of -2 would be stored as $-2 + 127 = 125$.
- Mantissa - The final 23 bits hold the *mantissa*, which represents the significant digits of the number in binary. In scientific notation, the mantissa is the number we multiply by a power of 10 (or, in binary, a power of 2). For instance, in the decimal number 4.2×10^1, the mantissa is 4.2, and the exponent is 1.

The first bit is the sign bit, the next sector is 8 bits of exponent, followed by the final 23 bits of *mantissa.* This is a simple concept, the mantissa is just what is normally the number we multiply by in scientific notation. (e.g $4.2E1$, 4.2 is the mantissa, 1 is the exponent)

1. Look at the sign and fill the MSB:

   $$.625_{10} \quad \text{Positive, MSB} \implies 0$$

2. Convert to binary if necessary and adjust to scientific notation.

   $$.625_{10} = .101_2 = 1.01 \times 2^{-1}$$

3. Hopefully this is becoming clearer now.

   Our Mantissa, or our *Fraction* is 1.01
   Our exponent is -1

4. Let's focus on the Fraction first. The leading 1 in 1.01×2^{-1} is ignored. We only care about the .01.

   The fraction portion of the standard is 23 bits long for a single precision number. So our last 23 bits are:

   $$01000000000000000000000$$

5. Now we add the Bias to the exponent and convert that to binary.

$$-1 + 127 = 126$$

Quick Maffs: Engrain $2^7 = 128$ into your head.

$$\implies 1111111_2 = 127_{10}$$

$$\implies 1111110_2 = 126_{10}$$

Remember that the exponent is 8 bits, so add the leading 0.

$$01111110_2$$

6. That leaves the final number:

$$00111111001000000000000000000000_2$$

1.2.4 Radix and Other Systems

The Radix is the base of a number system. The Radix Economy is the number of bits required to represent a number. It can be calculated by:

$$\text{Radix Economy} = \log_2(\text{Radix})]$$

The higher the number, the more efficient the system is. The peak efficiency is somewhere around $e = 2.718$. To calculate the radix economy of a given number, you multiply the amount of digits by the base.

Example 6

$$111_{10} \implies 3 \times 10 = 301101111_2 \implies 7 \times 2 = 14157_8 \implies 3 \times 8 = 24$$

The most common Radices are:

- Decimal - Base 10
- Binary - Base 2
- Octal - Base 8
- Hexadecimal - Base 16

To convert between these systems, you can use the following:

Example 7 Convert the following numbers to the specified base.

1. $101101_2 \rightarrow 8$ Group the numbers in sets of 3. Start from the LSB. 101 101 that's 5 and 5. Therefore, 55 in Octal.

$$\begin{aligned} 101101_2 &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 32 + 0 + 8 + 4 + 0 + 1 = 45_{10} \end{aligned}$$

   Now convert to Octal.

$$\begin{aligned} 45_{10} &= 5 \times 8^1 + 5 \times 8^0 \\ &= 55_8 \end{aligned}$$

2. $101101_2 \rightarrow 16$ Group the numbers in sets of 4. Start from the LSB. Now convert to Hexadecimal.

$$45_{10} = 2D_{16}$$

1.2.5 XS3, BCD, and Gray Code

These are different binary codes which are used for different purposes.

- BCD is simply the binary representation of a decimal number. With each digit represented by 4 bits. This is useful for arithmetic operations.
- XS3 is a conversion that adds 3. This means that we will have don't care's for the values of 6, 7, 8, and 9.
- Gray code is a binary code where only one bit changes at a time. This is useful for error detection and correction.

Example 8 Convert the following numbers to the specified code.

1. $101101_2 \rightarrow XS3$
   Just add 3 to the number.

$$101101_2 = 110110_2$$

2. $101101_2 \rightarrow BCD$
   This is a bit more involved. We need to convert the number to decimal first.

$$64 + 16 + 8 + 1 = 89_{10}$$

   8 is 1000 in binary, 9 is 1001 in binary. so, 89 in BCD is 1000 1001

3. $101101_2 \rightarrow Gray$

$$101101_2 = 111111_2$$

1.3 Boolean Algebra

1.3.1 Basic Laws

- **Commutative Law:** $A + B = B + A$, $AB = BA$
  We can flip the order of the operands.
- **Associative Law:** $A + (B + C) = (A + B) + C$, $A(BC) = (AB)C$
  We can change the grouping of the operands.
- **Distributive Law:** $A(B + C) = AB + AC$, $A + BC = (A + B)(A + C)$
  We can distribute the operation across the operands.
- **Identity Law:** $A + 0 = A$, $A \cdot 1 = A$
  We can **add** 0 or **multiply** by 1 without changing the value.
- **Complement Law:** $A + \overline{A} = 1$, $A \cdot \overline{A} = 0$
  The or of a variable and its complement is always 1, the and of a variable and its complement is always 0.
- **Involution Law:** $\overline{\overline{A}} = A$
  The compliment of the compliment is the original variable.
- **Absorption Law:** $A + AB = A$, $A(A + B) = A$
- **DeMorgan's Theorem:** $\overline{A + B} = \overline{A} \cdot \overline{B}$, $\overline{A \cdot B} = \overline{A} + \overline{B}$

1.3.2 Conversion Using DeMorgan's Theorem

The conversion process takes a bit getting used to, however it is straight forward.

1. Start with the original expression.
2. Apply DeMorgan's Theorem.
3. Simplify the expression.
4. Apply DeMorgan's Theorem again.
5. Simplify the expression.
6. If necessary, apply DeMorgan's Theorem again.

Example 9 Utilize DeMorgan's Theorem to implement the following logic using 2-input or 3-input NAND gates only:

$$F = \overline{A}(B + C) + CD$$

First, expand the expression.

$$F = \overline{A}B + \overline{A}C + CD$$

Now, start by adding two bars to the entire expression.

$$F = \overline{\overline{\overline{A}B + \overline{A}C + CD}}$$

Bring down the outermost bar and exchange our + for a $\cdot$.

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

Daas it.

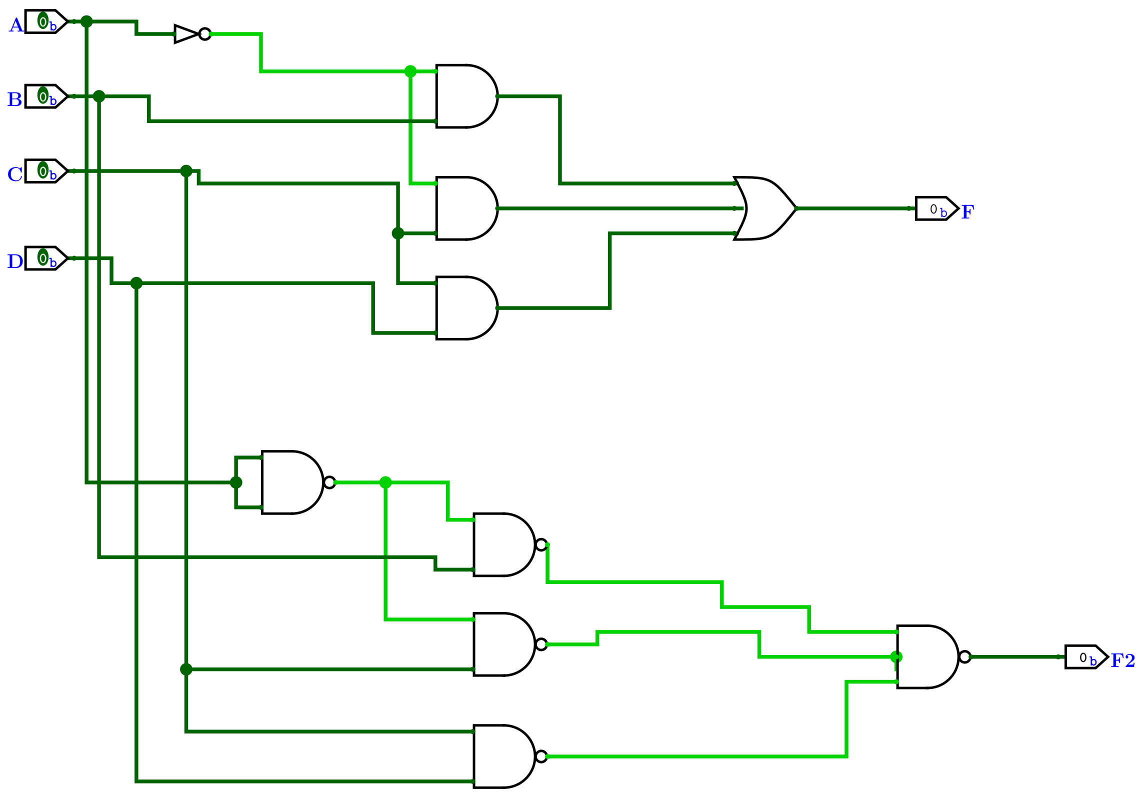


Figure 1: These are functionally the same

As a general guide when drawing these out, recall that a NAND with it's inputs tied together is an inverter. This is useful for simplifying the diagram.

For NOR, the same applies.

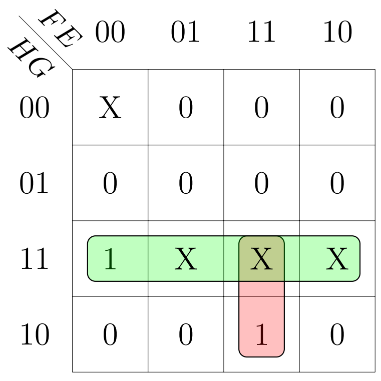
1.4 Karnaugh Maps

Karnaugh maps are a graphical representation of a truth table. They are used to simplify boolean expressions. The map is a grid of cells, where each cell represents a possible input combination. The number of cells is equal to the number of input variables. The cells are arranged in a way that adjacent cells differ by only one variable. The map is used to identify groups of cells that can be combined to simplify the expression.

Example 10 For the given XS3 to 8421 BCD converter, the truth table is as follows:

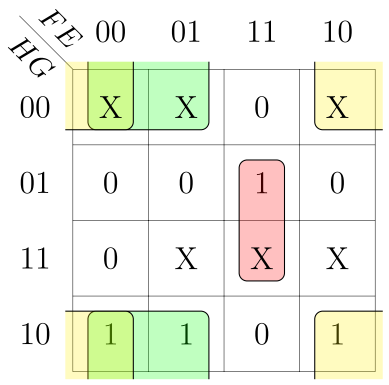
H	**G**	**F**	**E**	**D**	**C**	**B**	**A**
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1

Use Karnaugh Maps to determine the minimized function for D, C, B, and A.



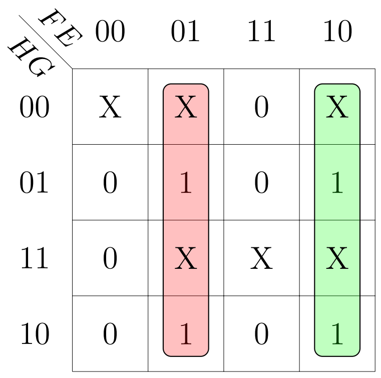
HG \ FE	00	01	11	10
00	X	0	0	0
01	0	0	0	0
11	1	X	X	X
10	0	0	1	0

(a) $D = HG + HFE$



HG \ FE	00	01	11	10
00	X	X	0	X
01	0	0	1	0
11	0	X	X	X
10	1	1	0	1

(b) $C = \overline{G}\overline{E} + \overline{G}\overline{F} + GFE$



HG \ FE	00	01	11	10
00	X	X	0	X
01	0	1	0	1
11	0	X	X	X
10	0	1	0	1

(c) $B = F \oplus E$

$A = \overline{E}$ by inspection

Figure 2: Karnaugh Map Excitation and Output Formulae

1.5 Circuits

For this section, simply remembering that a resistor is added in serial and, in parallel, the reciprocal of the resistance is added is sufficient.

$$R_{\text{total}} = R_1 + R_2 + \dots$$

$$R_{\text{total}} = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \dots}$$

To find the current drop acoss a load such as an LED, we use the equations

$$V = IR$$

Subtract the voltage drop across the LED from the total voltage to find the voltage for our equation. Then, divide by the resistance to find the current.

To find power dissipated by said LED, we use the equation

$$P = IE$$

Example 11 Assume we have a circuit with a 5V power supply, and a LED that drops 2V accross it. In series, there is a 200Ω resistor. Find the current.

$$V = IR$$
$$V = 5 - 2 = 3I = \frac{V}{R}$$
$$I = \frac{3}{200}$$
$$I = 0.015\text{A} = 15\text{mA}$$

Then, find the power dissipated:

$$P = IE$$
$$P = 0.015 \times 3$$
$$P = 0.045\text{W} = 45\text{mW}$$

Example 12 Given a 5V power source, a 500 Ω resistor in series, a 2V LED, and a parallel network of resistors at 1k Ω and 2k Ω find the current drop through the LED.

$$\begin{aligned}
V &= IR \\
R_{\text{total}} &= 500\Omega + \frac{1}{\frac{1}{1000\Omega} + \frac{1}{2000\Omega}} \\
R_{\text{total}} &= 500\Omega + 666.67\Omega = 1166.67\Omega \\
I &= \frac{V}{R} = \frac{5V - 2V}{1166.67\Omega} = 0.00256\text{A} = 2.56\text{mA}
\end{aligned}$$

The trick for this one is to remember to subtract the voltage drop across the LED and for the parallel network, to simply add the reciprocals of the resistances.

---

2 Exam 2: Getting Hairy

2.1 Topical Guide Objectives

1. Define Static Hazards and how to eliminate them
2. Be able to draw a timing diagram from a circuit with propagation delay
3. Be able to utilize standard gates to design simple sequential logic machines.
4. Be able to perform basic troubleshooting to identify test vectors
5. Understand parity error checking and how it is employed with XOR gates for both EVEN and ODD parity.
6. Understand a Triangular Parity pattern
7. Be able to translate boolean logic into a MUX operation. full adders,
8. Be able to design the simple machines:
   (a) A Parity Checker
   (b) 2-bit Comparator
   (c) 2-bit Encoder
   (d) 4-bit Decoder
   (e) 3-bit adder/subtractor
   (f) 4-bit sequential multiplier

2.2 Vocabulary

- Static Hazards - A hazard that occurs when a signal changes value before the circuit has had time to stabilize.
- Triangular Parity - A parity pattern that uses a triangular pattern to check for errors.
- A Hamming Code - A code that uses parity bits to check for errors.
- A Carry-Lookahead Adder - An adder that uses a carry-lookahead circuit to speed up the addition process.
- A MUX - A multiplexer that uses a select line to choose between inputs.
- A DEMUX - A demultiplexer that uses a select line to choose between outputs.
- A 2-bit Comparator - A circuit that compares two 2-bit numbers.
- A 2-bit Encoder - A circuit that encodes a 2-bit number.

2.3 Static Hazards

Static hazards are a problem in digital circuits where a signal changes value before the circuit has had time to stabilize. This can cause the circuit to produce an incorrect output.

There are two types of static hazards: static-0 hazards and static-1 hazards. A static-0 hazard occurs when a signal changes from 0 to 1 and then back to 0. A static-1 hazard occurs when a signal changes from 1 to 0 and then back to 1. Static hazards can be eliminated by adding additional gates to the circuit to ensure that the signal stabilizes before being used as an input to another gate.

They can be identified from a timing diagram or a karnaugh-map.

Example 13 From a timing diagram, we can see that there is a static-1 hazard at the transition from 1 to 0. This can be eliminated by adding a gate to the

---

2.4 Multiplexers and Demultiplexers

Multiplexers and demultiplexers are used to select between multiple inputs or outputs. A multiplexer is a circuit that has multiple inputs and one output. It uses a select line to choose which input to pass to the output.

A demultiplexeris a circuit that has one input and multiple outputs. It uses a select line to choose which output to pass the input to.

Example 14 Create a truth table for an 8-1 MUX as a function of B based on the following: m0 = 0, m1 = B, m2 = B', m3 = 1, m4 = 0, m5 = B, m6 = B, m7 = 0.

A	**B**	**C**	**Y**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

2.5 Encoder and Decoders

An encoder simply takes an input and selects it's output based on the input. A decoder takes an input and selects it's output based on the input.

Example 15 Design a 2-bit encoder with the following truth table:

A	**B**	**Y**
0	0	00
0	1	01
1	0	10
1	1	11

2.6 Parity and Hamming Codes

Parity is a method of error checking that uses a single bit to check for errors. There are two types of parity: even parity and odd parity. Even parity checks that the number of 1s in the data is even, while odd parity checks that the number of 1s in the data is odd.

A Hamming code is a type of error-correcting code that uses multiple parity bits to check for errors. The Hamming code can detect and correct single-bit errors and detect double-bit errors.

Example 16 The key to understanding parity circuit design is the XOR gate. The XOR gate is used to check for parity errors. For even parity, the XOR gate is used to check if the number of 1s in the data is even.
The Equation for even parity is:

$$P = A \oplus B \oplus C$$

For odd parity, the equation is:

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

In general the required number of parity bits for data bits must satisfy the equality:

$$2^p \geq m + p + 1$$

Example 17 For a given 4-bit data word, the parity bits are calculated as follows:

$$P_1 = D_1 \oplus D_2 \oplus D_3 \oplus D_4$$
$$P_2 = D_1 \oplus D_2 \oplus D_3 \oplus D_4$$
$$P_3 = D_1 \oplus D_2 \oplus D_3 \oplus D_4$$

2.7 Adder and Subtractor Circuits

Recall that a Full Adder contains both a carry in and a carry out. The carry in is used to add the carry from the previous bit. The carry out is used to pass the carry to the next bit. The sum is the result of the addition of the three inputs.

Example 18 Design a 3-bit adder/subtractor circuit. Design a circuit that takes two 3 bit numbers as input $A = A_2A_1A_0$ and $B = B_2B_1B_0$ and outputs the sum and carry out. The subtraction is carried out using the 2's complement method. The circuit should have add if $S = 0$ and subtract if $S = 1$.

This should set OVERFLOW = 1 if there's an overflow.

To design this, the XOR gate is absolute key. Recall that to take 2's compliment with an XOR gate, we simply invert the bits and add 1. The select bit can add the one by acting as the LSB Full Adders carry in.

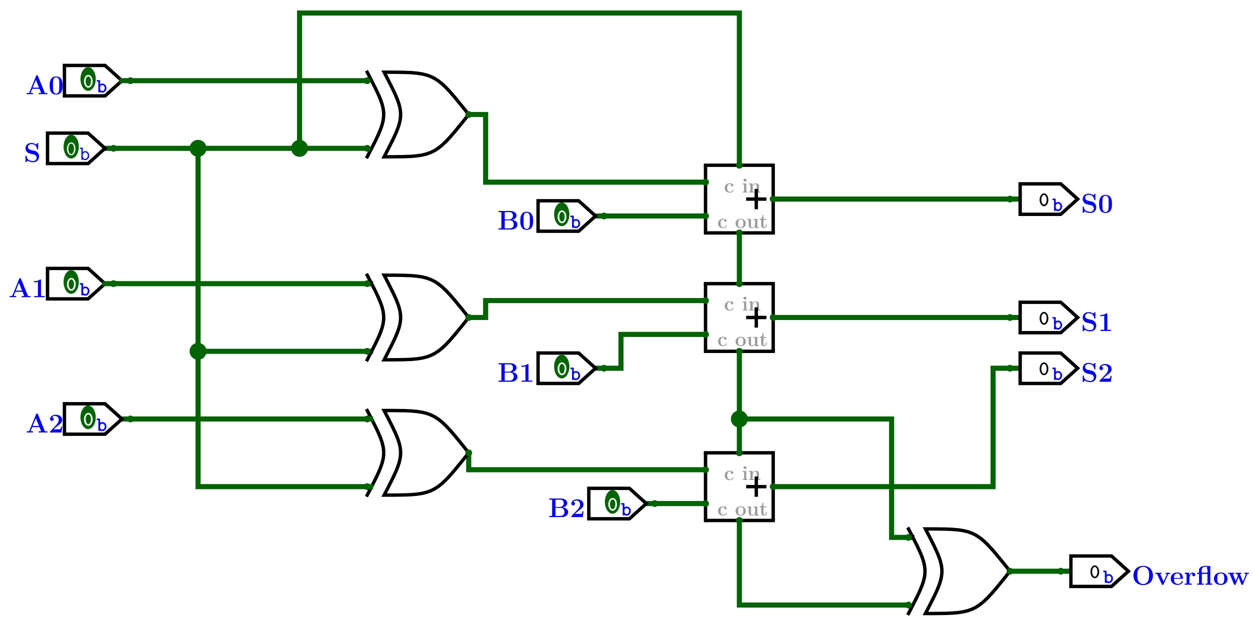


Figure 3: A simple 3 bit adder/subtractor circuit

2.8 A Priority Encoder/Comparator

Example 19 Design a Circuit to Compare two 2-bit Numbers
Given $A = A_1A_0$ and $B = B_1B_0$, design a circuit that will output 0 if the two numbers are equal, 1 otherwise.

You can only use 2 input XOR gates (limitless), one OR gate, and one INVERTER gate.

The solution for this one also lies in the XOR gate. You simply only need Three XOR gates to compare the two numbers. The output of the XOR gates will be 0 if the two numbers are equal, and 1 otherwise. The OR gate will then be used to combine the outputs of the XOR gates. The INVERTER is not needed as we are outputting 0 when the condition is true.

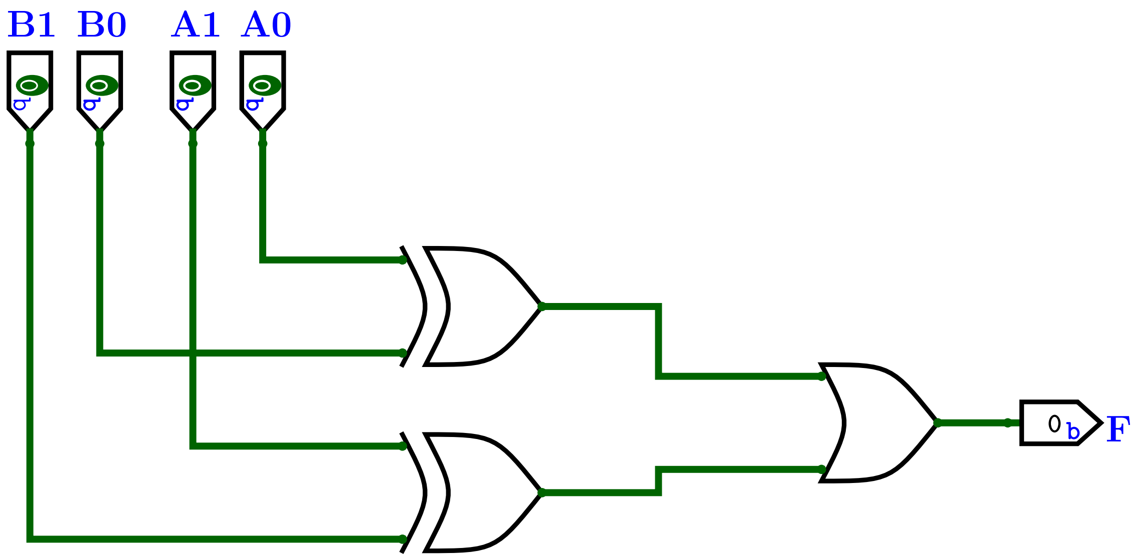


Figure 4: A simple 2 bit number comparator

3 Exam 3: Sequential Circuits

3.1 Topical Guide Objectives