

## 2 1/29/2020

### 2.1 Lecture Notes

- Analog signals can take any value across a continuous range of current, voltage, etc.
- While digital circuits can be analog too, they don't, because digital works better for their purpose. Digital signals restrict themselves to two discrete values: 0 and 1.
- Systems can be represented digitally. For example, consider an image, which is just thousands of pixels represented by bits.
- Analog signals are our physical reality. Things in analog signals are continuously variable. The design of an analog signal is extremely complex, and stability and accuracy is very difficult.
- In the beginning, we used vacuum tubes to go from analog to digital. However, due to vacuum tubes being inefficient, we eventually moved to transistors.

- Problem 2.1.1: Convert 37 to binary.

37/2	18	R1
18/2	9	R0
9/2	4	R1
4/2	2	R0
2/2	1	R0
1/2	0	R1
Answer: b101001		

- Problem 2.1.2: Convert b10111 to decimal.

$$1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$$
$$2^4 + 2^2 + 2^1 + 2^0$$
$$16 + 4 + 2 + 1$$

Answer: 23

- In the hex number system, each group of four becomes a single hex value. For example, b10100111 is xA7.

- Problem: Convert b101110101 to hex.

- First, start from the right and collect the four rightmost bits: 0101.
- Then, grab the next four bits: 0111.
- Finally, grab the remaining bit: 1. We need to put 0's in the front to pad this out to four bits, giving us 0001.
- Convert these groups to values. 0001 is equivalent to 1, 0111 is equivalent to 7, and 0101 is equivalent to 5.

Answer: x175

- Problem 2.1.3: Convert x3FA2 to binary.

- 3 in binary is 0011.
- F in binary is 1111.
- A in binary is 1010.
- 2 in binary is 0010.

Answer: 0011111110100010

## 2.2 Assigned Readings

- Addition and subtraction using binary numbers uses the same conventions you would use to add and subtract standard numbers. Four examples for addition and four examples for subtraction are shown below.

- Addition

$\begin{array}{r} 101111000 \\ 10111110 \\ + 10001101 \\ \hline 101001011 \end{array}$	$\begin{array}{r} 001011000 \\ 10101101 \\ + 00101100 \\ \hline 11011001 \end{array}$
--	---

$\begin{array}{r} 011111110 \\ 01111111 \\ + 00111111 \\ \hline 10111110 \end{array}$	$\begin{array}{r} 000000000 \\ 10101010 \\ + 01010101 \\ \hline 11111111 \end{array}$
---	---

- Subtraction

$\begin{array}{r} 001111100 \\ 11100101 \\ - 00101110 \\ \hline 10110111 \end{array}$	$\begin{array}{r} 001011000 \\ 10101101 \\ - 00101100 \\ \hline 11011001 \end{array}$
---	---

$\begin{array}{r} 010101010 \\ 10101010 \\ - 01010101 \\ \hline 01010101 \end{array}$	$\begin{array}{r} 000000000 \\ 11011101 \\ - 01001100 \\ \hline 10010001 \end{array}$
---	---

### 3 1/31/2020

#### 3.1 Lecture Notes

- Each group of bits has a size label associated with it.
  - The bit.
  - The nibble, which is 4 bits.
  - The byte, which is 8 bits.
  - The half-word, which is 16 bits.
  - The word, which is 32 bits.
  - The double word, which is 64 bits.

- Problem 3.1.1: Add b1011 to b101.

$$\begin{array}{r} \textcolor{blue}{1111} \\ 1011 \\ + \quad 101 \\ \hline \text{b10000} \end{array}$$

Answer: b10000

- Problem 3.1.2: Add b1011 to b1001.

$$\begin{array}{r} \textcolor{blue}{1\ 11} \\ 1011 \\ + \quad 1001 \\ \hline \text{b10100} \end{array}$$

Answer: b10100

- Problem 3.1.3: Subtract b111 from b1101.

$$\begin{array}{r} 1101 \\ - \quad 111 \\ \hline \text{b0110} \end{array}$$

Answer: b0110

- Problem 3.1.4: Subtract b1111 from b10101.

$$\begin{array}{r} 10101 \\ - \quad 1111 \\ \hline 110 \end{array} \quad \text{Broke it!}$$

Answer: No answer.

- Padding is the addition of zeroes in front of a number. For example, we could pad the number 429 out to 00429 without changing the value. This works in binary as well, as the value 1111 can be padded out to 001111 without its value being changed. This is important when considering hardware.
- Overflow is when the value is incorrect because it is larger than the allowed space.
- Errors also exist when subtracting binary. We can't borrow something that doesn't exist, as seen in problem 3.1.4. The result would be negative, meaning we cannot get the correct result using this method and number system.

- Problem 3.1.5: Prove adding b1101 and b111 results in a space constrained error.

$$\begin{array}{r}
 \textcolor{blue}{1111} \\
 1101 \\
 + \quad 111 \\
 \hline
 \textcolor{red}{b}10100
 \end{array}$$

Answer: Proven. The red numbers signify overflow.

- Problem 3.1.6: Prove subtracting b111 from b101 results in a borrow error.

$$\begin{array}{r}
 101 \\
 - \quad 111 \\
 \hline
 ?10
 \end{array}
 \quad \text{Broke it!}$$

Answer: Proven. Writing either “Error” or “Broke it” is acceptable.

## 3.2 Assigned Readingss

- In the signed-magnitude system, a number consists of a magnitude and a symbol indicating whether the number is positive or negative. We assume that the sign is positive if no sign is specified. There are two representations of zero, “+0” and “-0”, both with the same value.
- The signed-magnitude system is applied to binary numbers using an extra bit position used to represent the sign, called the sign bit. The most significant bit is typically used as the sign bit, with 0 representing a positive value and 1 representing a negative value.

$$\begin{aligned}
 - & 01010101_2 = +85_{10} \\
 - & 01111111_2 = +127_{10} \\
 - & 00000000_2 = +0_{10} \\
 - & 11010101_2 = -85_{10} \\
 - & 11111111_2 = -127_{10} \\
 - & 10000000_2 = -0_{10}
 \end{aligned}$$

- The signed-magnitude system has an equal number of positive and negative integers. An  $n$ -bit signed-magnitude integer lies within the range  $-(2^{n-1} - 1)$  through  $+(2^{n-1} - 1)$ .
- While the signed-magnitude system negates a number by changing its sign, a complement number system negates a number by taking its complement as defined by the system. Taking a complement is more difficult than simply changing the sign, however two numbers in a complement system can be added or subtracted directly without the sign and magnitude checks that have to be done in the signed-magnitude system.
- In a two’s complement system, the complement of an  $n$ -bit number  $B$  is obtained by subtracting it from  $2^n$ . If  $B$  is between 1 and  $2^n - 1$ , thus subtracting produces another number between 1 and  $2^n - 1$ . If  $B$  is 0, the result of the subtraction is  $2^n$ . Because of this, there is only one representation of zero in a two’s complement system.

- An unnecessary subtraction operation can be avoided by rewriting  $2^n$  as  $(2^n - 1) + 1$  and  $2^n - B$  as  $((2^n - 1) - B) + 1$ . For example, for  $n = 8$ ,  $100000000_2$  equals  $11111111_2 + 1$ .
- If we define the complement of a bit  $b$  to be the opposite value of the bit, then  $(2^n - 1) - B$  is obtained by simply complementing the bits of  $B$ . Therefore, the two's complement of a number  $B$  is obtained by complementing the number of individual bits in  $B$  and adding 1. Again, using  $n = 8$  as an example, the two's complement of  $01110100$  is  $10001011 + 1$ , or  $10001100$ .
- In the two's complement system, the MSB serves as the sign bit. A number is negative *if and only if* its MSB is 1.
- In a ones' complement system, the complement of an  $n$ -bit number  $B$  is obtained by subtracting it from  $2^n - 1$ . This is accomplished by complementing the individual digits of  $B$  without adding 1 like in the two's complement system. The MSB acts as the sign, with 0 being positive and 1 being negative. This gives two representations of zero, a positive zero and a negative zero.
- While positive number representations are the same for ones' and two's complement, negative numbers differ by one.
- A weight of  $(2^{n-1} - 1)$ , rather than  $-2^{n-1}$  is given to the most significant bit when computing the decimal equivalent of a ones' complement number.
- The main advantages a ones' complement system has is its symmetry and ease of complementation, given its usage in early computer. However, due to the added design of ones' complement being more complicated and there being two representations of zero, two's complement is more widely used presently.
- In an excess-B representation, an  $m$ -bit string whose unsigned integer value is  $M$  ( $0 \leq M \leq 2^m$ ) represents the signed integer  $M - B$ , where  $B$  is the bias of the number system.
- For example, an excess- $2^{m-1}$  system represents any number  $X$  in the range between  $-2^{m-1}$  through  $+2^{m-1} - 1$ . The range of this representation is exactly the same as that of  $m$ -bit two's complement numbers. In fact, the range of the two systems are identical with the sole difference being that the sign bits are always opposite. Excess representation is mostly used in floating point number systems.
- Because ordinary addition is just an extension of counting, two's complement numbers can be added using ordinary binary addition, ignoring any carries beyond the MSB. This result will always be accurate so long as the range of the number system is not exceeded.
- If an addition operation produces a result that exceeds the range of the number system, overflow is said to have occurred. Addition of two numbers with different signs will never produce overflow, but addition with like signs can, as shown below.

$$\begin{array}{r} 0101 \\ + 0110 \\ \hline 1011 = -5 \end{array}$$

$$\begin{array}{r} 0111 \\ + 0111 \\ \hline 1110 = -2 \end{array}$$

- Most subtraction circuits for two's complement numbers do not perform subtraction directly. Instead, they negate the subtrahend by taking its two's complement and then add it to the minuend using the normal rules of addition. This can be easily accomplished by performing a bit-by-bit complement of the subtrahend and then adding the complemented subtrahend to the minuend with an initial carry of 1 instead of 0. Examples are given below.

$$\begin{array}{r} \phantom{+} \phantom{0} 1 \\ \phantom{+} \phantom{0} 0100 \\ + \phantom{0} 1100 \\ \hline 10001 \end{array}$$

$$\begin{array}{r} \phantom{00}1 \\ \phantom{00}0011 \\ + \phantom{00}0011 \\ \hline \phantom{00}0111 \end{array}$$

$$\begin{array}{r} \phantom{00}1 \\ \phantom{+}0011 \\ + \phantom{0}1011 \\ \hline \phantom{+}1111 \end{array}$$

$$\begin{array}{r} \phantom{+} \phantom{00} 1 \\ \phantom{+} 1101 \\ + \phantom{00} 0011 \\ \hline 10001 \end{array}$$

Overflow in subtraction is detected by examining the signs of the minuend and the complemented subtrahend, using the same rule as addition.

- In unsigned addition, the carry or borrow in the most significant bit position indicates an out-of-range result. In signed two's complement addition the overflow condition defined earlier indicates an out-of-range result. The carry from the most significant bit position is irrelevant in signed addition, in the sense that the overflow can occur independently whether or not carry occurs.

## 4 2/3/2020

### 4.1 Lecture Slides

- A leading bit to the left of the number is used to tell us the sign. This special bit is called the sign bit. A sign bit of 0 represents a positive number and a sign bit of 1 represents a negative number.
- In the Sign and Magnitude system, or SAM, the sign bit is just stuck to the front of a number. If padding is necessary, you pad first and then add the sign bit. This gives us a range of  $-(2^{n-1} - 1)$  to  $(2^{n-1} - 1)$ . SAM results in two zeroes, which makes math tricky.
- Problem 4.1.1: Represent -32 in 8-bit SAM.

32/2	16	R0
16/2	8	R0
8/2	4	R0
4/2	2	R0
2/2	1	R0
1/2	0	R1

In 6-bit, 32 is b100000. We pad this out to 7-bit by adding a zero, giving us b0100000. Lastly, we add the sign bit for a negative number, giving us b10100000.

Answer: b10100000

- Problem 4.1.2: Represent -13 in 8-bit SAM.

13/2	6	R1
6/2	3	R0
3/2	1	R1
1/2	0	R1

In 4-bit, -13 is b1101. We pad this out to 7-bit by adding three zeroes, giving us b0001101. Lastly, we add the sign bit for a negative number, giving us b10001101.

Answer: b10001101

- There are two complement systems: ones' complement and two's complement. The systems involve swapping  $1 \rightarrow 0$  and  $0 \rightarrow 1$ . The complement of the complement is the original number.
- Recall that the number of bits matter. If you only add a spot for the sign bit, the number may need to be extended in the event that you need more bits. If the number is positive, just pad some zeroes. If the number is negative, extend with some 1's, or alternatively pad before taking the complement.
- In ones' complement, you add a spot for the sign and then flip the number. Ones' complement has the same double zero issue as SAM.

- Problem 4.1.3: Represent 32 with 8-bit ones' complement.

– Step 1: Get the positive binary value.

32/2	16	R0
16/2	8	R0
8/2	4	R0
4/2	2	R0
2/2	1	R0
1/2	0	R1

32 = b100000.

– Step 2: Add the positive sign bit.

b0100000.

– Step 3: Sign extend.

b00100000.

Answer: b00100000

- Problem 4.1.4: Represent -32 with 8-bit ones' complement.

– Steps 1-3: Get the unsigned value (see 4.1.3 for the work).

b00100000.

– Step 4: Flip the number.

b11011111.

Answer: b11011111

- Problem 4.1.5: Represent 13 with 8-bit ones' complement.

– Step 1: Get the positive binary value.

13/2	6	R1
6/2	3	R0
3/2	1	R1
1/2	0	R1

13 = b1101.

– Step 2: Add the positive sign bit.

b01101.

– Step 3: Sign extend.

b00001101.

Answer: b00001101

- Problem 4.1.6: Represent -13 with 8 bit ones' complement.

– Steps 1-3: Get the unsigned value (see 4.1.5 for the work).

b00001101.

– Step 4: Flip the number.

b11110010.

Answer: b11110010

- Two's complement is functionally identical to ones' complement except you add one at the end, removing the second zero.



- Problem 4.1.7: Represent 32 with 8-bit two's complement.  
A positive two's complement number is the same as a positive ones' complement number. Using the work from 4.1.3, we found positive 32 to be b00100000 in ones' complement.

Answer: b00100000

- Problem 4.1.8: Represent -32 with 8-bit two's complement.
  - Steps 1-3: Get the unsigned value (see 4.1.3 for the work).  
b00100000.
  - Step 4: Flip the number.  
b11011111.
  - Step 5: Add one.
 
$$\begin{array}{r} \text{b11011111} \\ + \quad \quad 1 \\ \hline \text{b11100000} \end{array}$$

Answer: b11100000

- Problem 4.1.9: Represent 13 with 8-bit two's complement.  
A positive two's complement number is the same as a positive ones' complement number. Using the work from 4.1.5, we found positive 13 to be b00001101 in ones' complement.

Answer: b00001101

- Problem 4.1.10: Represent -13 with 8-bit two's complement.
  - Steps 1-3: Get the unsigned value (see 4.1.5 for the work).  
b00001101.
  - Step 4: Flip the number.  
b11110010.
  - Step 5: Add one.
 
$$\begin{array}{r} \text{b11110010} \\ + \quad \quad 1 \\ \hline \text{b11110011} \end{array}$$

Answer: b11110011

- With the Excess-B system, the main idea is to move zero to a point that isn't zero, called the bias point. This allows you to have both positive and negative values. The Excess-B system is primarily used in specific systems design and IEEE-754 representation.
- There is the potential for an incorrect answer when performing binary math, specifically when there is a carry in to the sign bit but not out (or vice versa). Unless explicitly stated, use two's complement when using signed numbers.
- Addition between two positive or two negative numbers may be incorrect answer that is out of range. If there is a carry into and out of the sign bit, the answer is correct. Otherwise, the answer is false. In the event that this occurs, you can declare the answer to have "overflowed" and leave it there or redo the math with a larger amount of bits.

- Problem 4.1.11: Add -7 to 7 in 4-bit.

- Since this is signed math, we use two's complement.
- 7 in two's complement is b0111.
- -7 in two's complement is b1001.
- Next, we need to add the numbers.

$$\begin{array}{r}
 \textcolor{blue}{1111} \\
 \text{b0111} \\
 + \text{b1001} \\
 \hline
 \textcolor{red}{1}0000
 \end{array}$$

- We throw away the excess bit, or the red “1”, leaving us with b0000.

Answer: b0000

- Problem 4.1.12: Add 7 to -8 in 4-bit.

Answer: Can't be done in 4-bit, 8=1000.

- With subtraction, we exploit the idea that  $x = a - b = a + (-b)$ .

- Problem 4.1.13: Subtract 8 from -12 in 8-bit.

- Since this is signed math, we use two's complement.
- -12 in two's complement, after being padded appropriately and flipped, is b11110100.
- -8 in two's complement, after being padded appropriately and flipped, is b11111000.
- Next, we need to subtract the numbers.

$$\begin{array}{r}
 \textcolor{blue}{1111} \\
 \text{b11111000} \\
 + \text{b11110100} \\
 \hline
 \textcolor{red}{1}11101100
 \end{array}$$

- We throw away the excess bit, or the red “1”, leaving us with b11101100.

Answer: b11101100

## 4.2 Assigned Readings

- Multiplying binary numbers functions very similarly to normal multiplication. Forming shifted multiplicands is trivial in binary multiplication since the only multiplier digits are 0 and 1. An example of this multiplication is shown below.

	1011	multiplicand
×	1101	multiplier
	1011	
	0000	
	1011	shifted multiplicands
	1011	
	10001111	product

- In a digital system, it is more convenient to add each multiplicand as it is created to a partial product. Such a method looks like this:

	1011	multiplicand
×	1101	multiplier
<hr/>		
	0000	partial product
	1011	shifted multiplicand
<hr/>		
	01011	partial product
	0000↓	shifted multiplicand
<hr/>		
	001011	partial product
	1011↓↓	shifted multiplicand
<hr/>		
	0110111	partial product
	1011↓↓↓	shifted multiplicand
<hr/>		
	10001111	product

- In general, when we multiply an  $n$ -bit number by an  $m$ -bit number, the resulting product requires at most  $n + m$  bits to express. The shift-and-add algorithm requires  $m$  partial products and additions to obtain the result.
- Multiplication of signed numbers can be accomplished using unsigned multiplication. In other words, perform an unsigned multiplication of the magnitudes and make the product positive if the operands have the same sign but negative if they have different signs.
- We perform two's complement multiplication by using a sequence of two's complement additions of shifted multiplicands. Only the last step is changed, where the shifted multiplicand corresponding to the MSB of the multiplier is negated before being added to the partial product. An example of two's complement multiplication is shown below.

	1011	multiplicand
×	1101	multiplier
<hr/>		
	00000	partial product
	11011	shifted multiplicand
<hr/>		
	111011	partial product
	00000↓	shifted multiplicand
<hr/>		
	1111011	partial product
	11011↓↓	shifted multiplicand
<hr/>		
	11100111	partial product
	00101↓↓↓	shifted and negated multiplicand
<hr/>		
	00001111	product

## 5 2/5/2020

### 5.1 Lecture Slides

- Binary multiplication functions very similarly to decimal multiplication.
- When multiplying with signed numbers, you should do all multiplication in the positive form. Remember to add the sign bit in the front. If necessary, correct the product to a negative value. After multiplying, pad to the nearest power of 2.
- Problem 5.1.1: Multiply b0101 and b0011.

$$\begin{array}{r} 0101 \\ \times 0011 \\ \hline 0101 \\ 0101\downarrow \\ 0000\downarrow\downarrow \\ 0000\downarrow\downarrow\downarrow \\ \hline b00001111 \end{array}$$

Answer: Answer: b00001111

- Problem 5.1.2: Multiply b0111 and b0011.

$$\begin{array}{r} 0111 \\ \times 0011 \\ \hline 0011 \\ 0011\downarrow \\ 0011\downarrow\downarrow \\ 0000\downarrow\downarrow\downarrow \\ \hline b00010101 \end{array}$$

Answer: b00010101

- Problem 5.1.3: Multiply b0101 and b1011.

$$\begin{array}{r} 0100 \\ + 1 \\ \hline 0101 \\ \times 0101 \\ \hline 0101 \\ 0000\downarrow \\ 0101\downarrow\downarrow \\ 0000\downarrow\downarrow\downarrow \\ \hline b0011001 \\ + 1100110 \\ \hline b11100111 \end{array}$$

Answer: b11100111

- Problem 5.1.4: Multiply b1101 and b0011.

$$\begin{array}{r}
 0010 \\
 + \quad 1 \\
 \hline
 0011 \\
 \\
 \begin{array}{r}
 0011 \\
 \times 0011 \\
 \hline
 0011 \\
 0011\downarrow \\
 0000\downarrow\downarrow \\
 0000\downarrow\downarrow\downarrow \\
 \hline
 00001001 \\
 11110110 \\
 + \quad 1 \\
 \hline
 b11110111
 \end{array}
 \end{array}$$

Answer: b11110111

- Problem 5.1.5: Multiply b1101 and b1011.

$$\begin{array}{r}
 0010 \\
 + \quad 1 \\
 \hline
 0011 \\
 0100 \\
 + \quad 1 \\
 \hline
 0101 \\
 \\
 \begin{array}{r}
 0011 \\
 \times 0101 \\
 \hline
 0011 \\
 0000\downarrow \\
 0011\downarrow\downarrow \\
 0000\downarrow\downarrow\downarrow \\
 \hline
 b00001111
 \end{array}
 \end{array}$$

Answer: b00001111

- Problem 5.1.6: Multiply b1011 and b1011.

$$\begin{array}{r}
 0100 \\
 + \quad 1 \\
 \hline
 0101 \\
 \\
 \begin{array}{r}
 0101 \\
 \times 0101 \\
 \hline
 0101 \\
 0000\downarrow \\
 0101\downarrow\downarrow \\
 0000\downarrow\downarrow\downarrow \\
 \hline
 b00011001
 \end{array}
 \end{array}$$

Answer: b00011001

## 5.2 Assigned Readings

- A set of  $n$ -bit strings in which different bit strings represent different numbers or other things is called a code. A particular combination of  $n$  1-bit values is called a code word.
- There may not necessarily be a mathematical relationship between a particular code word and what it is supposed to represent. Furthermore, a code using  $n$ -bit strings doesn't need to contain  $2^n$  valid code words.
- Listed below are some common ways the ten decimal digits are represented.

Decimal digit	BCD (8421)	2421	Excess-3	Biquinary	1-out-of-10
0	0000	0000	0011	0100001	1000000000
1	0001	0001	0100	0100010	0100000000
2	0010	0010	0101	0100100	0010000000
3	0011	0011	0110	0101000	0001000000
4	0100	0100	0111	0110000	0000100000
5	0101	1011	1000	1000001	0000010000
6	0110	1100	1001	1000010	0000001000
7	0111	1101	1010	1000100	0000000100
8	1000	1110	1011	1001000	0000000010
9	1001	1111	1100	1010000	0000000001
Unused code words					
	1010	0101	0000	0000000	0000000000
	1011	0110	0001	0000001	0000000011
	1100	0111	0010	0000010	0000000101
	1101	1000	1101	0000011	0000000110
	1110	1001	1110	0000101	0000000111
	1111	1010	1111	...	...

- The most natural of these is the binary-coded decimal system, or BCD. This system encodes the digits 0 through 9 by their 4-bit unsigned binary representations. Because of this, conversions between BCD and decimal representations are trivial.
- Some computer programs place two BCD digits into one 8-bit byte in the packed-BCD representation. In this system, one byte may represent the values from 0 to 99 versus 0 to 255 for a normal, unsigned 8-bit binary number. BCD numbers for any desired value can be obtained by using one byte for every two digits.
- Similarly to binary numbers, there are many possible representations of negative BCD values. Signed BCD numbers have one extra digit position for the sign, and both the signed-magnitude and 10's complement representations are used in BCD arithmetic. In signed-magnitude BCD, the encoding of the sign bit string is arbitrary, while in 10's complement, 0000 indicates a positive value and 1001 indicates a negative value.
- Addition of BCD digits function similarly to adding 4-bit unsigned numbers, with the sole difference being that a correction must be made if the result exceeds 1001, in which case the result is corrected by adding six.

- Binary-coded decimal is a weighted code because each decimal digit can be obtained from its code word by assigning a fixed weight to each code-word bit. The weights for the BCD bits are 8, 4, 2, and 1, which leads to BCD sometimes being called 8421 code.
- Another set of weights leads to 2421 code, which has the advantage of being self-complementing, in that the code word for the 9s' complement of any digit can be obtained by complementing the individual bits of the digit's code word.
- Another self-complementing code is excess-3 code, which although not weighted does have a mathematical relationship with BCD code. The code word for each decimal digit in excess-3 is the corresponding BCD code word plus 0011.
- Decimal codes can have more than four digits, shown with the biquinary code system, using seven. The first two bits of the code word indicate whether the number is within the range of 0-4 and 5-9, and the remaining five indicate which of those five numbers is being represented. This system is used in an abacus.
- An advantage to using more than the minimum number of bits is an error-detecting property. In biquinary code, for example, if any single bit is changed to the opposite value the resulting code word immediately does not represent a decimal digit.
- 1-out-of-10 code uses ten bits instead of four.
- Gray code is a numbering system where only one bit changes between adjacent numbers. The code words for gray code are listed below.

Decimal Number	Binary Code	Gray Code
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

- A string of digits doesn't need to represent a number. In fact, most of the information processed by computers is nonnumeric. The most common type of nonnumeric data is text, or strings of characters from some character set. Each character is represented in the computer by a bit string according to an established convention, such as ASCII.
- ASCII represents each character with a 7-bit string, yielding a total of 128 different characters. The ASCII standard contains uppercase, lowercase, numerals, punctuation, and even various nonprinting control characters.

## 6 2/7/2020

### 6.1 Lecture Notes

- A code is a pattern that represents some idea, concept, or a piece of information. With numbers, for example, you have a code that represents a quantity.
- Gray code is an example of a code and is very useful for sensors. Gray code introduces us to the idea of a code word, which is simply a way of using 0s and 1s to represent information. BCD, ASCII, Unicode, and Gray Code are all codeword representations.
- In the Binary Coded Decimal, or BCD, representation, the numbers 0 through 9 are represented. Addition isn't too messy either, with a carry only needing to be forced when adding numbers totaling to over 9.
- Problem 6.1.1: Convert 97 to BCD.  
9 in BCD: b1001  
7 in BCD: b0111  

Answer: b10010111
- Problem 6.1.2: Convert 36 to BCD.  
3 in BCD: b0011  
6 in BCD: b0110  

Answer: b00110110
- When adding BCD, you must consider the possibility of errors resulting in invalid code. This can be fixed with a correction or a forced carry.
- Problem 6.1.3: Add 14 and 27 in BCD.  
$$\begin{array}{r} 14 = 00010100 \\ + \quad 27 = 00100011 \\ \hline 00111011 \\ + \qquad \qquad 0110 \\ \hline b01000001 \end{array}$$

Because the original number was 0011 1011, which is 3 and 11. The 11 isn't a single digit, so a carry must be forced.

Answer: b01000001
- Problem 6.1.4: Add 9 and 13 in BCD.  
$$\begin{array}{r} 14 = \quad 1001 \\ + \quad 27 = 00010011 \\ \hline 00011100 \\ + \qquad \qquad 110 \\ \hline b00100010 \end{array}$$

Answer: b00100010
- The American Standard Code for Information Exchange, or ASCII, is a 7-bit representation with 128 possible characters. Extending ASCII allows for the usage of the 8th bit, opening up 255 possible characters.



- The last important number representation is fixed point. When we deal with something that isn't a whole number (which happens very frequently), we are essentially adding negative powers to the right of the radix point.
- Fixed point representation uses the following equation, with the highlighted portion being the radix point:  $B = b_{n-1}b_{n-2}...b_1b_0.b_{-1}b_{-2}...b_{-k}$
- Listed below are various representations of 2 using fixed point.

- $2^{-1} = 0.5$
- $2^{-2} = 0.25$
- $2^{-3} = 0.125$
- $2^{-4} = 0.0625$
- $2^{-5} = 0.03125$

- Problem 6.1.5: Convert 6.3 into floating point.

6 = 0110

.3 · .2 = 0.6	0
.6 · .2 = 1.2	1
.2 · .2 = 0.4	0
.4 · .2 = 0.8	0
.8 · .2 = 1.6	1
.6 · .2 = 1.2	1
.2 · .2 = 0.4	0
.4 · .2 = 0.8	0
.8 · .2 = 1.6	1
.6 · .2 = 1.2	1

Answer: b110.0100110011

- There are two conditions that allow you to stop doing the above process. The first is if the number to the right of the radix point is 0, because 0 multiplied by anything is 0. 0.500000 is just .5, mathematically speaking. If this never happens, you go to ten places, which is what happened above.

## 6.2 Assigned Readings

- Boolean algebra is an algebraic system designed to “give expression ... to the fundamental laws of reasoning in the symbolic language of a Calculus.”
- Eventually, Boolean algebra was applied to analyze and describe the behavior of circuits built from relays. In a system called switching algebra, the condition of a relay contact, whether it be open or closed, is represented by a variable X that can have one of two possible values: 0 or 1.
- In switching algebra, we use a symbolic variable such as the aforementioned X to represent the condition of a logic signal. A logic signal can represent a variety of conditions depending on the technology. For each technology, one condition is 0 and another 1.

- Most logic circuits use the positive-logic convention, using 0 to represent a LOW voltage and 1 to represent a high voltage. The negative-logic convention is the inverse of this, but is rarely used.
- The axioms (or postulates) of a mathematical system are a minimal set of basic definitions that we assume to be true. Using axioms, all other information about a particular system can be derived.
- All axioms are stated as a pair. This is a characteristic of axioms in switching algebra called “duality.”
- An inverter is a logic circuit whose output signal level is the opposite, or complement, of its input signal level. We use a prime tick (') to denote an inverter function.
- This prime tick is an algebraic operator, and a statement such as  $X'$  is an expression.
- A 2-input AND gate is a circuit whose output is 1 if both of its inputs are 1. The function of a 2-input AND gate is sometimes called logical multiplication and is symbolized algebraically by a multiplication dot ( $\cdot$ ). Some mathematicians and logicians use the wedge ( $\wedge$ ) to denote logical multiplication.
- A 2-input OR gate is a circuit whose output is 1 if either of its inputs are 1. The function of a 2-input AND gate is sometimes called logical addition and is symbolized algebraically by a plus sign ( $+$ ). Some mathematicians and logicians use the vee ( $\vee$ ) to denote logical addition.
- By convention, logical multiplication has a higher precedence than logical addition.
- Switching algebra theorems are statements known to always be true that allow us to manipulate algebraic expressions for simpler analysis. For example, the theorem  $X + 0 = X$  allows us to substitute every occurrence of  $X + 0$  in an expression with just  $X$ . A list of theorems involving one variable are shown below.

(T1)	$X + 0 = X$	(T1D)	$X \cdot 1 = X$	(Identities)
(T2)	$X + 1 = 1$	(T2D)	$X \cdot 0 = 0$	(Null elements)
(T3)	$X + X = X$	(T3D)	$X \cdot X = X$	(Idempotency)
(T4)	$(X')' = X$			(Involution)
(T5)	$X + X' = 1$	(T5D)	$X \cdot X' = 0$	(Complements)

- Most theorems can be easily proven by using a technique called perfect induction.
- Switching algebra theorems with two or three variables are listed below.

(T6)	$X + Y = Y + X$	(T6D)	$X \cdot Y = Y \cdot X$	(Commutativity)
(T7)	$(X + Y) + Z = X + (Y + Z)$	(T7D)	$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$	(Associativity)
(T8)	$X \cdot Y + X \cdot Z = X \cdot (Y + Z)$	(T8D)	$(X + Y) \cdot (X + Z) = X + Y \cdot Z$	(Distributivity)
(T9)	$X + X \cdot Y = X$	(T9D)	$X \cdot (X + Y) = X$	(Covering)
(T10)	$X \cdot Y + X \cdot Y' = X$	(T10D)	$(X + Y) \cdot (X + Y') = X$	(Combining)
(T11)	$X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$			(Consensus)
(T11')	$(X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$			

## 7 2/10/2020

### 7.1 Lecture Notes

- Boolean algebra is an approach towards closed set arithmetic, or a reasoning in a symbolic language, and was developed by George Boole in 1854. Such an approach was originally simplified to only using 0s and 1s, but it was eventually expanded upon by other mathematicians.
- Claude Shannon applied the idea of Boolean algebra, allowing it to be used from circuits, which are built from relays (alternatively known as switches).
- Axioms and postulates are the base set of mathematical ideas known to be true.
- There are two forms of logic in Boolean algebra: Positive logic (where one is true and zero is false) and negative logic (where zero is true and one is false).
- All complete equations and functions are composed of three parts: An equals sign, an expression or variable on the left side, and an expression or variable on the right side.
- An inverter is the idea of complementing or inverting a variables value in an expression. Inverting is traditionally represented by a ' '.
- Logical addition is a Boolean algebra concept and is denoted with the + sign. It is fundamentally the "OR" concept.
- Logical multiplication is another Boolean algebra concept and is denoted with the \* sign. It is fundamentally the "AND" concept.
- In Boolean algebra, operators have precedence, with some operators taking priority over others. The order is parenthesis  $\leftarrow$  inverters  $\leftarrow$  multiplication  $\leftarrow$  addition.
- Problem 7.1.1: Simplify  $Y = (1 * 1 + (0 * 1 * 1 + 0 + 1) * 0 + 1) * (0 + 1)$ . Red simplifies to 1, blue to 0, and green to 1. Thus, the we can first simplify the equation to  $Y = (1 + 0 + 1) * 1$ . Purple simplifies to 1, so we can further simplify the equation to  $Y = 1 * 1$ . This, the equation simplified is equivalent to  $Y = 1$ .  

Answer:  $Y = 1$
- Problem 7.1.2: Simplify  $T = (1 * 1 * 1 * 1 * 0 * 1 + 0 * 0) * 1 + (1 + 1 * 1 * 1)$ . Red simplifies to 0, blue to 0, and green to 1. Thus, we can first simplify the equation to  $T = (0 + 0) * 1 + (1 + 1)$ . Purple simplifies to 1, and orange simplifies to 1. Thus, we can yet further simplify the equation to  $T = 0 * 1 + 1$ . Brown simplifies to 0, finally simplifying the equation  $T = 0 + 1$ , leaving the answer as  $T = 1$ .  

Answer:  $T = 1$
- Commutativity is the idea that the order of operations do not matter.

- Associativity is the idea that the order in which segments of an equation are executed do not matter.
- Covering refers to some terms being excluded when a single term “covers” all possible cases.
- Redundancy is a simplification trick when simplifying algebraic expressions.
- General idempotency is simply the idea of applying impotence repeatedly.

$$\begin{aligned}
- & x = x * x = x * x * x * x * x \\
- & x = x + x = x + x + x + x + x
\end{aligned}$$

- Problem 7.1.3: Simplify  $T = Y' * F + X * (X' + Y)$ .
  - Distributive property:  $T = Y' * F + X * X' + X * Y$
  - Identity property:  $T = Y' * F + 0 + X * Y$
  - Simplify:  $T = Y' * F + X * Y$

Answer:  $T = Y' * F + X * Y$

- Problem 7.1.4: Simplify  $Y = X * X * T + T * T + X$ .
  - Idempotency:  $Y = X * T + T + X$
  - Distributive property:  $X(T + 1) + T$
  - Simplification:  $X(1) + T$
  - Simplification:  $X + T$

Answer:  $Y = X + T$

- Augustus DeMorgan came up with a theorem that describes factoring and distributing inverted functions. His theorems are listed below.
  - $(x * y)' = x' + y'$
  - $(x + y)' = x' * y'$
  - $x + x' * y = x + y$
  - $x * (x' + y) = x * y$

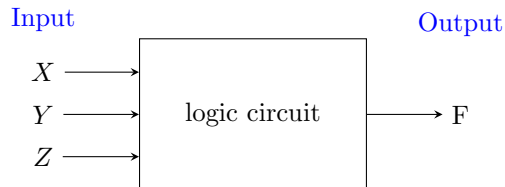
- Problem 7.1.5: Simplify  $T = X * (Y + X)' + Y$ .
  - DeMorgan’s Law:  $T = X * (Y' * X) + Y$
  - Distributive Law:  $X * Y + X * X' + Y$
  - Simplification:  $T = X * Y' + Y$

Answer:  $T = Y + X$

- The main idea behind Shannon’s expansion theorem is configuring an expression to contain more variables than the hardware would normally allow.

## 8.2 Assigned Readings


- A logic circuit can be represented with a minimum amount of detail by representing it as a “black box” with a certain number of inputs and outputs. Below is an example of a logic circuit.




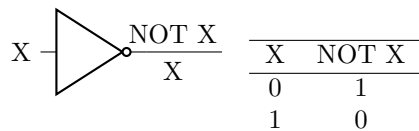
- A logic circuit whose outputs depend only on its current inputs is called a combinational circuit. The operation of such a circuit is fully described by a truth table that lists all combinations of input values and the corresponding output values. Below is an example of a truth table.

$X$	$Y$	$Z$	$F$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

- A circuit with memory whose outputs depend on the current input in addition to the sequence of past inputs are called sequential circuits. Its behavior may be described by a state table, which specifies its output and next state as functions of its current state and input.
- The most basic digital devices are called gates. Generally speaking, a gate has one or more inputs and produces an output that is a function of the current input values.
- Just three basic logic functions (AND, OR, and NOT) can be used to build any combinational logic circuit. The graphical symbols, along with their corresponding truth tables, are shown below.

$X$ $Y$ 	$X \text{ AND } Y$ $X \cdot Y$	<table> <tr><th><math>X</math></th><th><math>Y</math></th><th><math>X \text{ AND } Y</math></th></tr> <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> </table>	$X$	$Y$	$X \text{ AND } Y$	0	0	0	0	1	0	1	0	0	1	1	1
$X$	$Y$	$X \text{ AND } Y$															
0	0	0															
0	1	0															
1	0	0															
1	1	1															

$X$ $Y$ 	$X \text{ OR } Y$ $X + Y$	<table> <tr><th><math>X</math></th><th><math>Y</math></th><th><math>X \text{ OR } Y</math></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>1</td></tr> </table>	$X$	$Y$	$X \text{ OR } Y$	0	0	0	0	1	1	1	0	1	1	1	1
$X$	$Y$	$X \text{ OR } Y$															
0	0	0															
0	1	1															
1	0	1															
1	1	1															



The gates' functions are easily described using words.

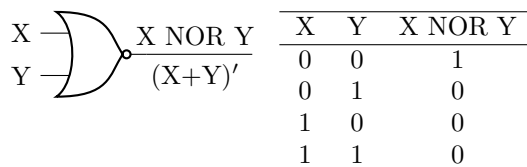
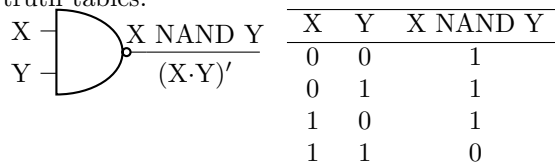
- An AND gate produces a 1 output if and only if all of its inputs are 1.
- An OR gate produces a 1 output if and only if one or more of its inputs is 1.
- A NOT gate is usually called an inverter and produces an output value the opposite of its input value.

- Take notice of the below circle from the inverter symbol's output.

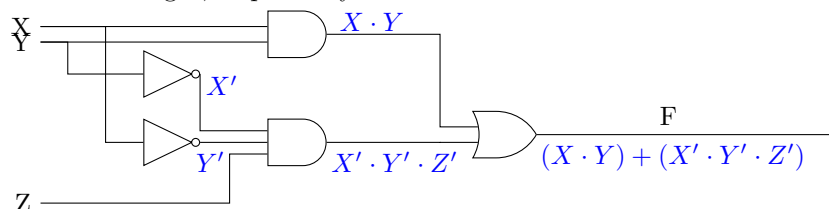


This is called an inversion bubble and is used in this and other gate symbols to denote “inverting” behavior.

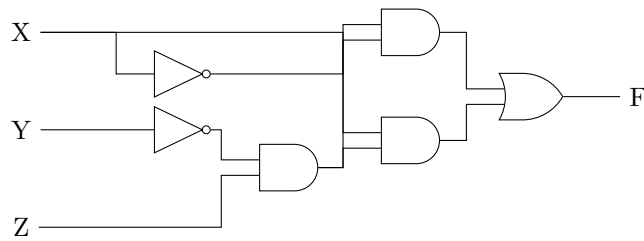
- Two additional logic functions are obtained by inverting the outputs of AND and OR gates. Shown below are their graphical symbols and their truth tables.



- A logic diagram shows the graphical symbols for multiple logic gates and other elements in a logic circuit, in addition to their interconnections (called wires). The output of each element may connect to inputs of one or more other elements. Signals in a logic diagram traditionally flow left to right, and inputs and outputs of the overall circuit are drawn on the left and right, respectively.



- Besides voltage and current, logic circuits are also useful in representing time. A timing function graphically shows how a circuit may respond to a

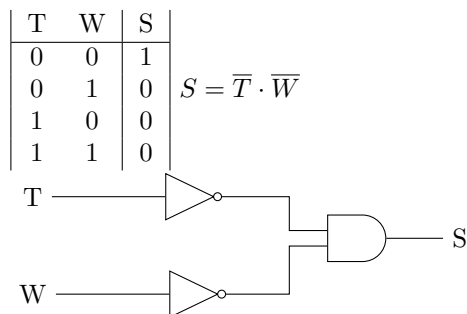


- When designing a system, there are a few important questions to ask.
  - What are the inputs?
  - What are the outputs?
  - Are there any constraints or relationships between the inputs and outputs?

After asking these questions, you can build the design.

- Problem 11.1.3: Design a system to turn on and off a sprinkler in the yard.  
 Inputs: Time(day<sup>1</sup> or night<sup>0</sup>) → T  
 Weather (rain<sup>1</sup> or no rain<sup>0</sup>) → W

Sprinkler: On<sup>1</sup> or off<sup>0</sup> = output → S



- Problem 11.1.4: Design a system to determine if you are allowed to watch Netflix.  
 Inputs: Homework (have<sup>1</sup> or have not<sup>0</sup>) → H  
 Class (in class<sup>1</sup> or not in class<sup>0</sup>) → C  
 Output: Watch Netflix (yes<sup>1</sup> or no<sup>0</sup>) → N

H	C	N
0	0	1
0	1	0
1	0	0
1	1	0

$$N = \overline{H} \cdot \overline{C}$$

$$\overline{H} \cdot \overline{C} = N$$

$$N = \overline{H} \cdot \overline{C} + H\overline{C} = \overline{C}$$

## 12 2/21/2020

### 12.1 Lecture Slides

- When solving logic systems, we usually want to take the lazier approach, so we look for methods that are simpler with less spots for error and a lower price to implement.
- Minimization aims to reduce the “cost”, which is done by reducing the number and size of gates.
- There are three key ways to reduce cost.
  1. Minimize the number of first-level gates.
  2. Minimize the number of inputs on each first level gate.
  3. Minimize the number of inputs on each second level gate.
- How are you handle how many inputs to use for a gate? Do as you’re told, do as you’re equipped to handle, and do as you want (in that order).
- Problem 12.1.1: Find the simplified POS  $F = \prod_{x,y,z}(1, 2, 5, 6)$ .  
 $F = (x + y + \bar{z})(x + \bar{y} + z)(\bar{x} + y + \bar{z})(\bar{x} + \bar{y} + z)$   
 $F = (\cancel{x}x^x + x\bar{y} + xz + yx + \cancel{y}\bar{y}^0 + yz + \bar{z}x + \bar{z}x + \cancel{\bar{z}}\bar{z}^0)^0$   
 $(\cancel{x}x^x + x\bar{y} + y\bar{x} + \cancel{y}\bar{y}^0 + yz + \bar{z}x + \bar{z}\bar{y} + \cancel{\bar{z}}\bar{z}^0)$   
 $F = x\bar{x} + x\bar{x}\bar{y} + x\bar{x}z + xy\bar{z} + xyz + x\bar{z}\bar{x} + x\bar{z}\bar{y}$   
 This takes too long! For POS functions, simplifying them algebraically is far too complicated. We need an alternative.
- Instead of solving algebraically, we use Karnaugh maps. Some Karnaugh maps are shown below.

		$X$	
		0	1
$Y$	0	0	2
	1	1	3

		$X$			
		00	01	11	10
$Z$	0	0	2	6	4
	1	1	3	7	5

$Y$



- Problem 12.1.5: Find the simplified SOP for  $F = \sum_{x,y,z}(0, 4, 5, 6, 7)$  using a K-Map.

$F:$

		$x$			
		00	01	11	10
$z$	0	0 1	2	6 1	4 1
	1	1	3	7 1	5 1
		$y$			

$$\overline{xy}z + x\overline{y}z$$

$$\overline{yz}(\overline{x} + x^1)$$

$$\overline{yz}$$

110  $xy\overline{z}$   
 100  $\overline{x}\overline{y}\overline{z}$   
 111  $xyz$   
 101  $\overline{x}\overline{y}z$

If it's the same, it stays, but if it changes it goes.

Distinguished 1's: 0, 6, 7, 5

Essential prime implicant(s):  $\overline{yz}, x$

Answer:  $F = \overline{yz} + x$

- Problem 12.1.6: Find the simplified SOP for  $F = \sum_{x,y,z}(0, 1, 4, 5, 7)$  using a K-Map.

$F:$

		$x$			
		00	01	11	10
$z$	0	0 1	2	6	4 1
	1	1 1	3	7 1	5 1
		$y$			

Distinguished 1's: 0, 1, 4, 7

Essential prime implicant(s):  $\overline{y}, xz$

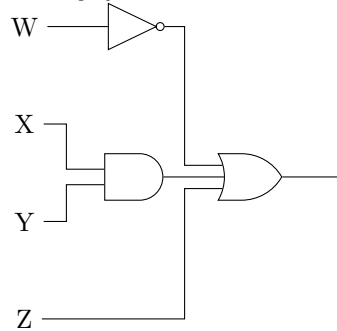
Answer:  $F = \overline{y} + xz$

## 12.2 Assigned Readings

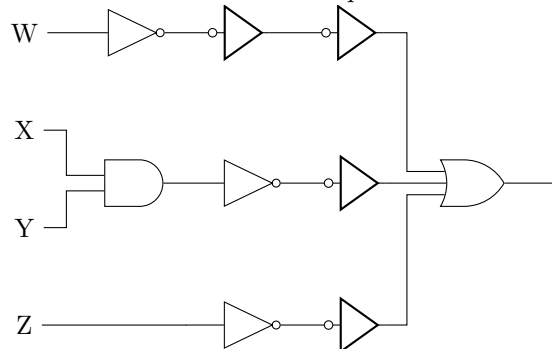
- We have previously described AND, OR, and NOT gates. We also want to use NAND and NOR gates, as these are faster than AND and OR gates in most technologies.

- NAND and NOR connectives aren't very logically intuitive, however. For example, you wouldn't say "I won't date you if you're not clean or not wealthy and also you're not smart or not friendly." You would instead say "I'll date you if you're clean and wealth or if you're smart and friendly." To produce a "natural" logic expression we need ways to translate this into other forms for a more efficient implementation.
- We can translate any logic expression into an equivalent sum-of-product expression simply by multiplying it out.
- We can insert a pair of inverters between each AND-gate output and the corresponding OR-gate input in a two-level AND-OR circuit. These inverters, per T4 (see page 21) have no effect on the output function of a circuit.
- However, if these inverters are absorbed into the AND and OR gates, we wind up with an AND-NOT gate on the first level and a NOT-OR gate on the second. These are two symbols for the same type of gate: The NAND gate. A two level AND-OR gate can be converted to a two level NAND-NAND gate simply by substituting gates. Below is a realization of a sum-of-products circuit.

a. AND-OR



b. AND-OR with extra inverter pairs



(c) 4-variable

		WX		W	
		00	01	11	10
YZ	00	0	4	12	8
	01	1	5	13	9
	11	3	7	15	11
	10	2	6	14	10

X

Z

- Observe the below K-Map.

F:		AB		A	
		00	01	11	10
CD	00	0	4	12	8
	01	1 1	5 1	13 1	9
	11	3 1	7 1	15	11 1
	10	2 1	6	14	10

B

D

Take note of how adjacent 1 cells were grouped to correspond to their prime implicant, or “product terms that cover only input combinations for which the function has a 1 output, and that would cover at least one input combination with a 0 output if any variable were removed.”

## 13 2/24/2020

### 13.1 Lecture Slides

- Problem 13.1.1: Simplify  $F = \sum_{A,B,C,D}(0, 6, 9, 11, 14)$ .

$F$ :

		$AB$		$A$	
		00	01	11	10
$CD$	00	0 1	4	12	8
	01	1	5	13	9 1
	11	3	7	15	11 1
	10	2	6 1	14 1	10
		$B$		$D$	

Distinguished 1's: 0,6,9,11,14

Essential prime implicant(s):  $\overline{A}BC\overline{D}$ ,  $A\overline{B}D$ ,  $BC\overline{D}$

Answer:  $F = \overline{A}BC\overline{D} + A\overline{B}D + BC\overline{D}$

- Problem 13.1.2: Simplify  $F = \sum_{A,B,C,D}(0, 2, 8, 10, 15)$ .

$F$ :

		$AB$		$A$	
		00	01	11	10
$CD$	00	0 1	4	12	8 1
	01	1	5	13	9
	11	3	7	15 1	11
	10	2 1	6	14	10 1
		$B$		$D$	

Distinguished 1's: 0,2,8,10,15

Essential prime implicant(s):  $ABCD$ ,  $\overline{B}\overline{D}$

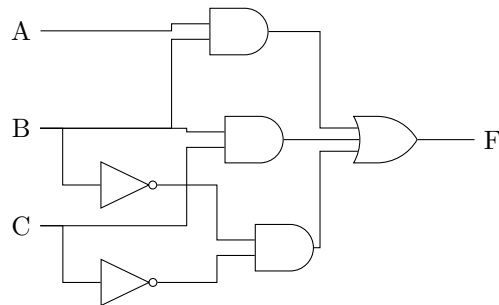
Answer:  $F = ABCD + \overline{B}\overline{D}$

- Problem 13.1.3: Find the shorthand POS and simplified SOP and draw the two level logic diagram for  $F = \prod_{A,B,C}(1,2,5)$ .

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

$F$ :

		$A$			
		00	01	11	10
$C$	0	0 1	2	6 1	4 1
	1	1	3 1	7 1	5
		$B$			



Distinguished 1's: 0, 3

Essential prime implicant(s):  $\overline{BC}, BC$

Answer:  $F = \overline{BC} + BC + AC$

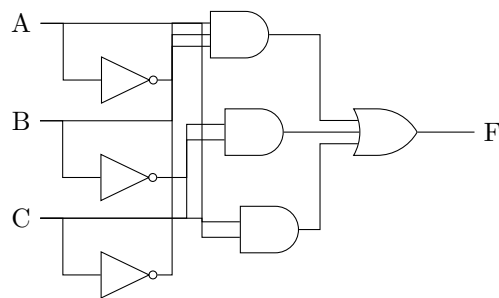
- Problem 13.1.4: Find the shorthand POS and simplified SOP and draw the two level logic diagram for  $F = \prod_{A,B,C}(0,3,4,6)$ .

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

$F$ :

$AB$		$A$			
		00	01	11	10
$C$	0	0	2 1	6	4
	1	1 1	3	7 1	5 1

$B$



Distinguished 1's: 1, 2, 7

Essential prime implicant(s):  $\overline{B}C, \overline{A}B\overline{C}, AC$

Answer:  $F = \overline{B}C + \overline{A}B\overline{C} + AC$

- Problem 13.1.5: Find the simplified SOP from  $F = \prod_{A,B,C,D}(0, 1, 2, 3, 4, 6, 8, 10, 11, 14)$  and draw the 2 level logic diagram.

$F$ :

$AB$		$A$			
		00	01	11	10
$CD$	00	0	4	12 1	8
	01	1	5 1	13 1	9 1
$C$	11	3	7 1	15 1	11
	10	2	6	14	10

$B$

$D$

Distinguished 1's: 5, 7, 12, 15, 9

Essential prime implicant(s):  $AB\overline{C}, A\overline{C}D, BD$

## 14 2/26/2020

### 14.1 Lecture Slides

- Equipment costs money, so we need to have a method to describe the cost of a function. This is done by counting the number of gates. Due to standard practice, initial NOT gates are not counted.
- Problem 14.1.1: Draw the K-Map for  $F = \sum_{A,B,C,D}(0,1,3,4,7,13,15)$  and state the cost.

$F$ :

		$AB$		$A$	
		00	01	11	10
$CD$	00	0 1	4 1	12	8
	01	1 1	5	13 1	9
	11	3 1	7 1	15 1	11
	10	2	6	14	10
		$B$		$D$	

Distinguished 1's: 4, 13

Essential prime implicant(s):  $\overline{ACD}, ABD$

Answer:  $F = \overline{ACD} + ABD + \overline{ABD} + BCD \therefore$  Cost is 5

- Problem 14.1.2: Draw the K-Map for  $F = \sum_{A,B,C,D}(1,2,7,9,10,11,15)$  and state the cost.

$F$ :

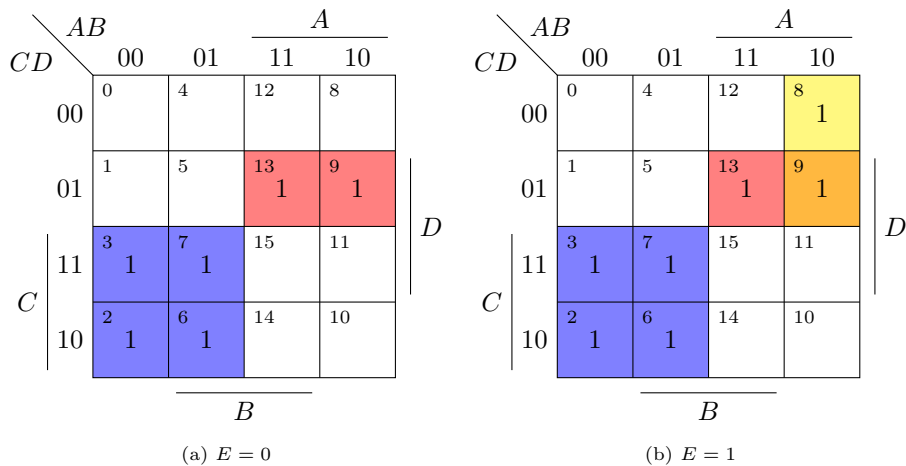
		$AB$		$A$	
		00	01	11	10
$CD$	00	0	4	12	8
	01	1 1	5	13	9 1
	11	3	7 1	15 1	11 1
	10	2 1	6	14	10 1
		$B$		$D$	

Distinguished 1's: 1, 2, 7

Essential prime implicant(s):  $\overline{BCD}, \overline{BCD}, BCD$

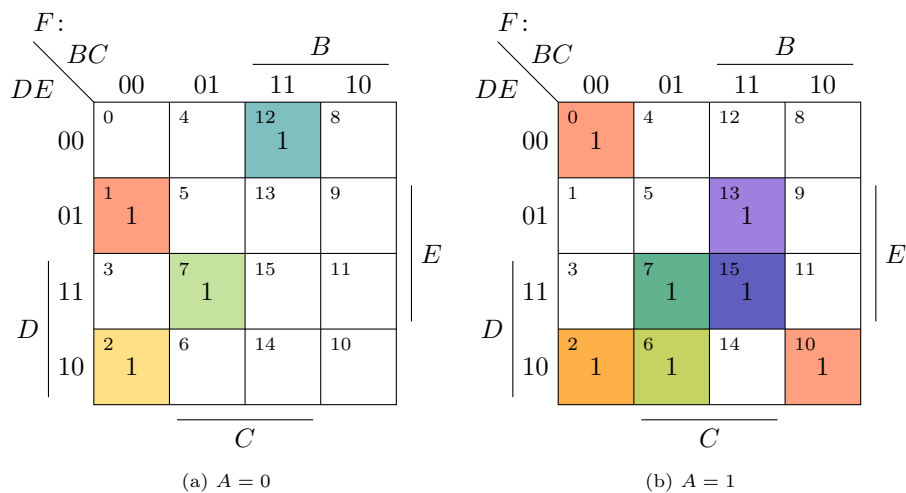
Answer:  $F = \overline{BCD} + \overline{BCD} + BCD + \overline{ABD} + \overline{ABC} \therefore$  Cost is 6

- 5-variable K-Maps take the form of  $f(A, B, C, D, E)$ . To solve these, we use two 4-variable K-Maps, one with  $A = 0$  (for 0-15) and one with  $A = 1$  (for 16-31). We combine adjacent 1's in three directions: vertically, horizontally, and out of the page. Below is an example.



Answer:  $F = AC + \overline{A}\overline{C}D + \overline{A}B\overline{C}E$

- Problem 14.1.3: Solve the 5-variable K-Map  $F(A, B, C, D, E) = \sum(1, 2, 7, 12, 16, 18, 22, 23, 26, 29, 31)$ .



Distinguished 1's: 1, 12, 2, 7, 16, 26, 29

EPIs:  $\overline{A}BC\overline{D}E, \overline{A}BCDE, \overline{B}C\overline{D}E, \overline{B}CDE, \overline{A}B\overline{C}E, \overline{A}BCE, \overline{A}C\overline{D}E$

$F = \overline{A}BC\overline{D}E + \overline{A}BCDE + \overline{B}C\overline{D}E + \overline{B}CDE + \overline{A}B\overline{C}E + \overline{A}BCE + \overline{A}C\overline{D}E + \overline{A}BCD$

- Incompletely specified functions contain a don't care condition which is denoted by an 'X'. A don't care is an input condition that either can't



occur or even if it occurs it doesn't matter. A circuit with one or more don't cares is called an incompletely specified circuit.

- When forming prime implicant's, treat don't cares as a 1, but ignore them when actually selecting prime implicants. Never select a prime implicant only to cover a don't care.
- Problem 14.1.4: Draw the 5-variable K-Map  $F(A, B, C, D, E) = \sum(1, 9, 10, 17, 21, 27) + d(3, 5, 11, 25, 26)$ .

$F:$

$BC$		$B$			
		00	01	11	10
$DE$	00	0	4	12	8
	01	1	5 X	13	9 1
	11	3 X	7	15	11 X
	10	2	6	14	10 1
$D$		$C$			

(a)  $A = 0$

$F:$

$BC$		$B$			
		00	01	11	10
$DE$	00	0	4	12	8
	01	1	5 1	13	9 X
	11	3	7	15	11 1
	10	2	6	14	10 X
$D$		$C$			

(b)  $A = 1$

Distinguished 1's: 1, 9, 10, 17, 21  
 Essential prime implicants:  $\overline{BDE}, \overline{BCE}, \overline{BCD}$   
 Answer:  $F = \overline{BDE} + \overline{BCE} + \overline{BCD}$

- Problem 14.1.5: Draw the K-Map for  $F(A, B, C, D) = \sum(1, 6, 7) + d(3, 5, 13, 15)$ .

$F:$

$AB$		$A$			
		00	01	11	10
$CD$	00	0	4	12	8
	01	1	5 X	13 X	9
	11	3 X	7 1	15 X	11
	10	2	6 1	14	10
$C$		$B$			

Distinguished 1's: 1, 6  
 Essential prime implicant(s):  $\overline{AD}, \overline{ABC}$   
 Answer:  $F = \overline{AD} + \overline{ABC}$

## 15 2/28/2020

### 15.1 Lecture Slides

- Thus far, we have been focusing on minimization with SOP, but what about POS? POS minimization is important to understand for considering other designs in the future. It has similar rules to SOP minimization, but flipped.
- Recall the “recipe” for SOP minimization from page 42. We will now construct a “recipe” for constructing a POS K-Map.
  1. Place all of the maxterms on the K-Map.
  2. Draw all of the PIs, starting with the largest size possible and then working your way down. All PIs must be powers of 2. This time, we cover 0's.
  3. Determine the distinguished 0's. Using these distinguished 0's, find the essential prime implicants.
  4. Begin creating the function using these essential prime implicants.
  5. Look at the map and check if there are any 0's not covered by prime implicants.
- Problem 15.1.1: Find the simplified POS for  $F = \prod_{w,x,y,z}(1, 2, 5, 6, 9, 10, 13, 15)$ .

$F$ :

		$WX$		$W$	
		00	01	11	10
$YZ$	00	0	4	12	8
	01	1 0	5 0	13 0	9 0
	11	3	7	15 0	11
	10	2 0	6 0	14	10 0
		$X$			

$Z$

Distinguished 0's: 1, 5, 6, 9, 10, 15

Essential prime implicant(s):  $(W + \bar{Y} + Z), (X + \bar{Y} + Z), (Y + \bar{Z}), (\bar{W} + \bar{X} + \bar{Z})$

Answer:  $F = (W + \bar{Y} + Z)(X + \bar{Y} + Z)(Y + \bar{Z})(\bar{W} + \bar{X} + \bar{Z})$

- Problem 15.1.2: Find the simplified POS for  $F = \prod_{w,x,y,z}(1, 2, 5, 6, 9, 10, 13, 15) + d(0, 3, 14)$ .

$F$ :

		$WX$		$W$	
		00	01	11	10
$YZ$	00	0 X	4	12	8
	01	1 0	5 0	13 0	9 0
	11	3 X	7	15 0	11
	10	2 0	6 0	14 X	10 0

$X$

$Z$

Distinguished 0's: 5, 6, 9, 10

Essential prime implicant(s):  $(Y + \bar{Z}), (\bar{Y} + Z)$

Answer:  $F = (Y + \bar{Z})(\bar{Y} + Z)(\bar{W} + \bar{X} + \bar{Z})$

- Problem 15.1.3: Find the simplified POS for  $F = \prod_{x,y,z}(1, 5, 6) + d(2, 7)$ .

$F$ :

		$XY$		$X$	
		00	01	11	10
$Z$	0	0	2 X	6 0	4
	1	1 0	3	7 X	5 0

$Y$

Distinguished 0's: 1

Essential prime implicant(s):  $(Y + \bar{Z})$

Answer:  $F = (Y + \bar{Z})(\bar{X} + \bar{Y})$

- You may also encounter a situation where you need to convert a function to a K-Map to ultimately end up with a simplified form.

- Problem 15.1.4: Convert the following function into a map and find the simplified SOP.  $F = \overline{A}BC\overline{D} + \overline{A}BCD + \overline{A}B\overline{D} + \overline{A}BC + \overline{A}BCD + \overline{A}BCD + \overline{A}BC + \overline{A}B\overline{D}$ .

$F$ :

		$AB$		$A$	
		00	01	11	10
$CD$	00	0 1	4	12	8 1
	01	1	5	13	9 1
$C$	11	3	7 1	15	11
	10	2 1	6 1	14	10 1
		$B$			

$D$

Distinguished 1's: 0, 7, 10, 10

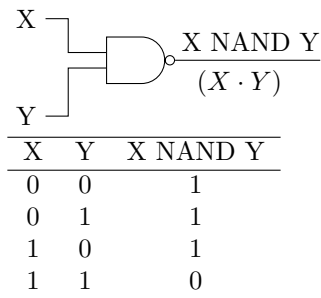
Essential prime implicant(s):  $\overline{A}BC, \overline{B}D, \overline{A}BC$

Answer:  $F = \overline{A}BC + \overline{B}D + \overline{A}BC$

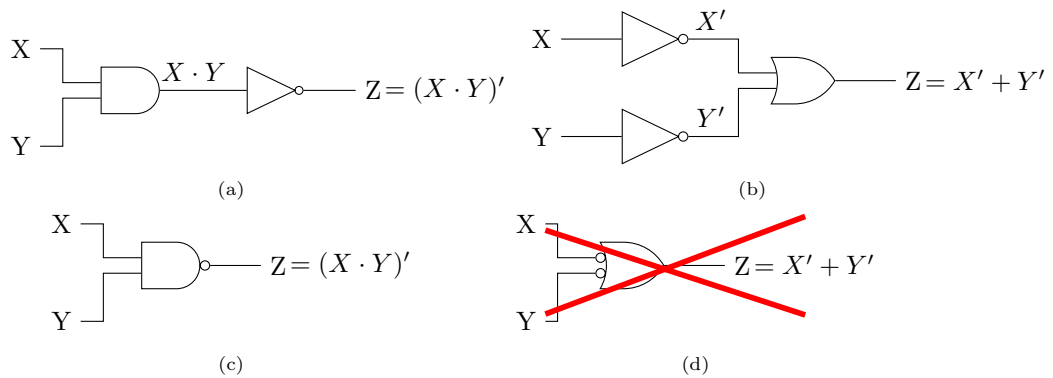
## 16 3/02/2020

### 16.1 Lecture Slides

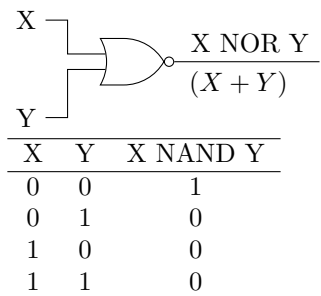
- We sometimes use other gates because they can be more efficient or convenient.
- The first gate we will go over is the NAND gate, a Not AND gate.



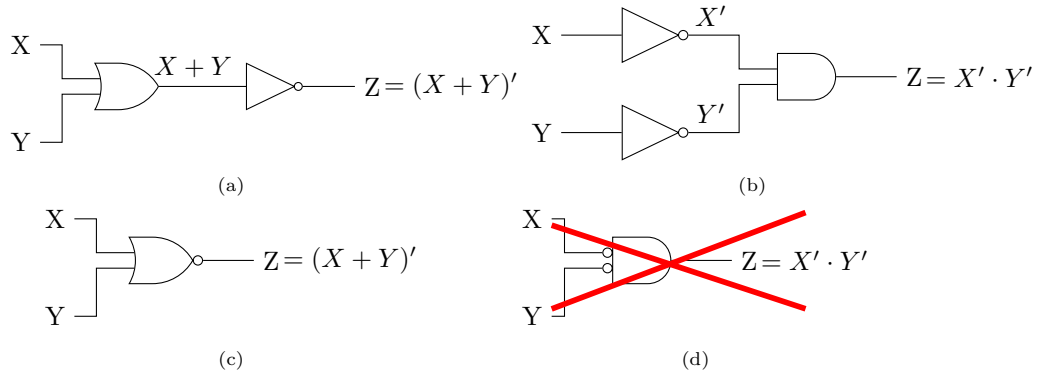
- Below are some other perspectives of a NAND gate.



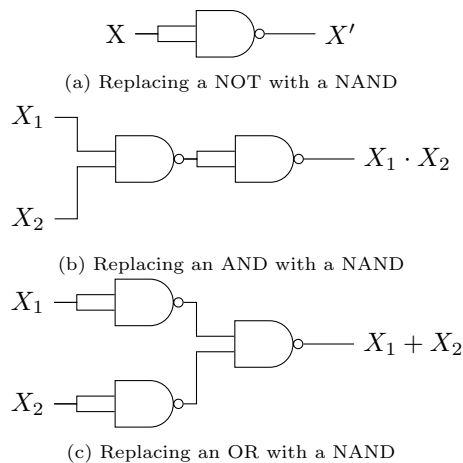
- The next gate is the NOR gate, or a Not OR gate.



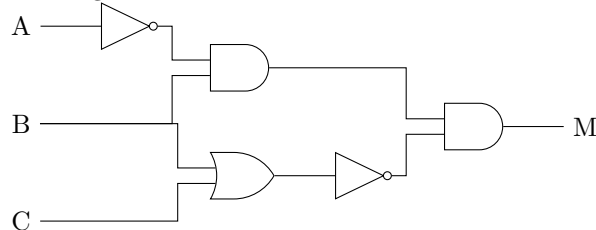
- Below are some other perspectives of a NOR gate.

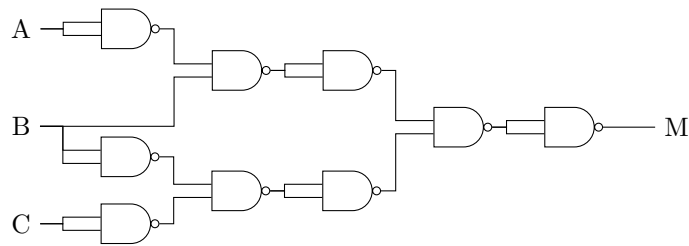


- Why do we use these new gates? The answer is simple: These are complete sets. One gate can be used to create a complete logic gate set (AND, OR, and NOT). Not having to mess around with other types of configurations makes setup much easier.
- In the end, it's all about mapping.

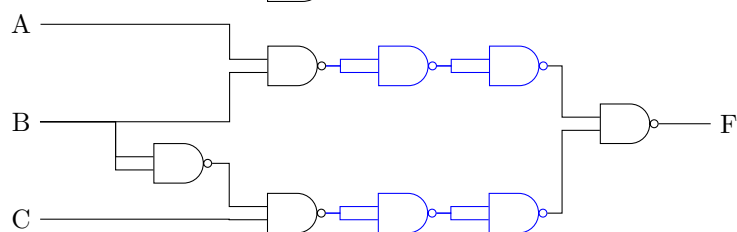
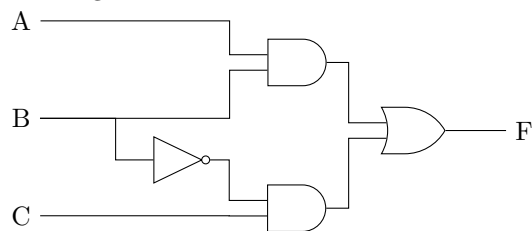


- So why does this work? In short, we are adapting DeMorgan's Law and the Involution Theorems.
- Problem 16.1.1: Convert the given logic diagram into one only using NAND gates.

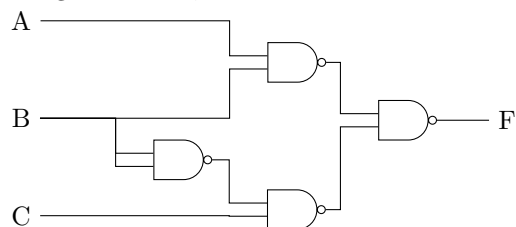




- Problem 16.1.2: Convert the given logic diagram into one only using NAND gates.



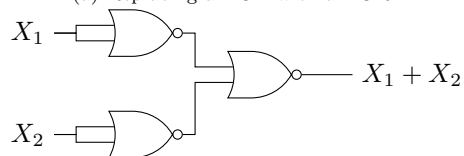
Using involution, we can cross out the blue elements, giving us...



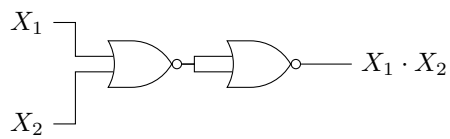
- We also have mappings for NOR gate conversions.



(a) Replacing a NOT with a NOR

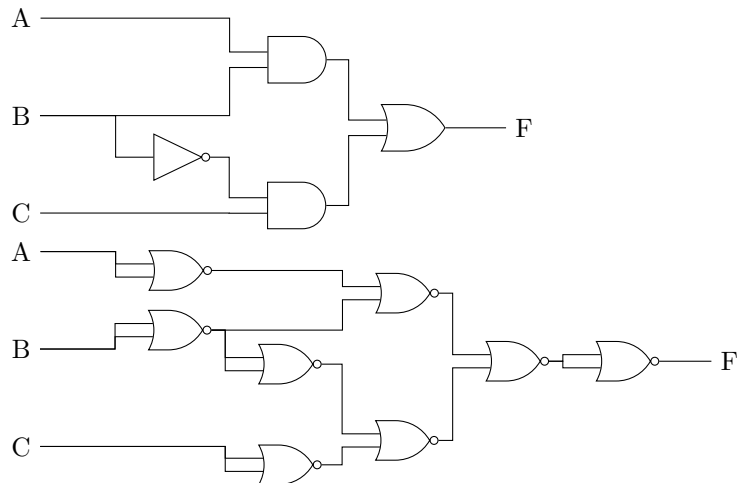


(b) Replacing an AND with a NOR

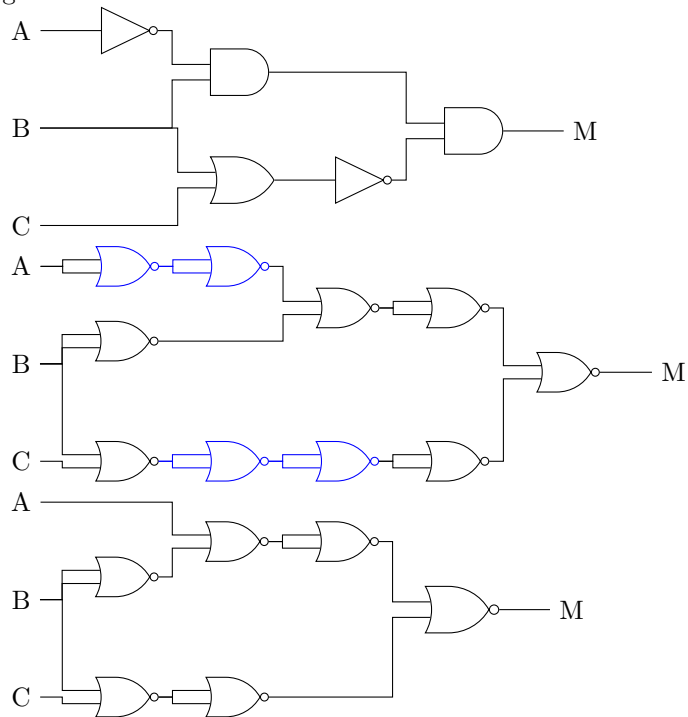


(a) Replacing an OR with a NAND

- Problem 16.1.3: Convert the given logic diagram into one only using NOR gates.



- Problem 16.1.4: Convert the given logic diagram into one only using NOR gates.





- A static-1 hazard is the possibility of a circuit's output producing a 0 glitch when we would expect the output to remain at a nice steady 1 based on a static analysis of the function. It has the following formal definition:

“A static-1 hazard is a pair of input combinations that: (a) differ in only one input variable and (b) both give a 1 output; such that it is possible for a momentary 0 output to occur during a transition in the differing input variable.”

- A static-0 hazard is the possibility of a 1 glitch when we expect the circuit to have a steady 0 output. It has the following formal definition:

“A static-0 hazard is a pair of input combinations that: (a) differ in only one input variable and (b) both give a 0 output; such that it is possible for a momentary 1 output to occur during a transition in the differing input variable.”

- Karnaugh maps can be used to detect static hazards in a two-level SOP or POS circuit. The existence or nonexistence of static hazards depends on the circuit design for a logic function.
- A properly designed two-level SOP (AND-OR) circuit has no static-0 hazards. A static-0 hazard would only exist in the circuit if both a variable and its complement were connected to the same AND gate. However, these can have static-1 hazards.
- For static-1 hazard analysis, we circle the product terms corresponding to the AND gates in the circuit and we search for adjacent 1 cells that are not covered by a single product term.
- Below is a Karnaugh map.

$F:$

		$XY$			
		00	01	11	10
$Z$	0	0	2	6 1	4 1
	1	1	3 1	7 1	5
		$Y$			

It should be immediately obvious that there is no single product term that can cover both combinations 111 and 110. Thus, it is theoretically possible for the output to momentarily “glitch” to 0 if the AND gate output that covers one of the combinations goes to 0 before the other can go to 1. This is solved by simply adding an extra product term (another AND gate) to cover the hazardous pair, shown below.

$F:$

$XY$		$X$			
		00	01	11	10
$Z$	0	0	2	6 1	4 1
	1	1	3 1	7 1	5

$Y$

The extra product term is called the consensus, which is what we add to eliminate hazards.

- A properly designed two-level POS (OR-AND) circuit has no static-1 hazards. It can however have static-0 hazards, which are eliminated in a manner dual to the foregoing.

$F:$

$WX$		$W$			
		00	01	11	10
$YZ$	00	0	4 1	12 1	8
	01	1 1	5 1	13	9
$Y$	11	3 1	7 1	15 1	11 1
	10	2	6	14 1	10 1

$X$

(a) As originally designed

$F:$

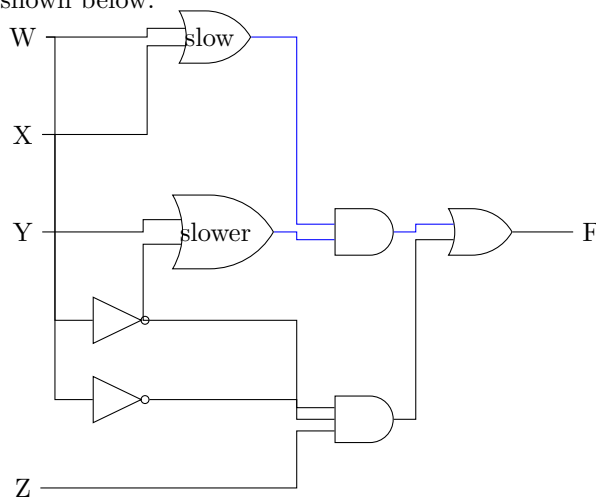
$WX$		$W$			
		00	01	11	10
$YZ$	00	0	4 1	12 1	8
	01	1 1	5 1	13	9
$Y$	11	3 1	7 1	15 1	11 1
	10	2	6	14 1	10 1

$X$

(b) With extra product terms to cover static-1 hazards

- A dynamic hazard is the possibility of an output changing more than once as the result of a single input transition. Multiple output transitions can occur if there are multiple paths with different delays from the input to the output. Dynamic hazards do not appear in properly designed two-level AND-OR or OR-AND circuits.

- Let's go over an example of a dynamic hazard. Consider the below circuit. It has three paths from input X to output F. One of the paths goes through a slower OR gate and another goes through an even slower OR gate. If the circuit's input is  $W, X, Y, Z = 0, 0, 0, 1$ , then the output will be 1, as shown below.



Now, let's suppose we change the X input to a 1. Assuming that all of the gates except the two "slow" gates are very fast, the transitions not marked blue will occur next, and the output goes to zero. Eventually the output of the "slow" gate changes and the output becomes a 1. Finally, the output of the "slower" gate changes and the output is finally at 0.

- Only a few situations (such as the design of feedback sequential circuits) require hazard-free combinational circuits. Methods for finding hazards in arbitrary circuits can be difficult, so if you absolutely need a hazard-free design, it is best to utilize a circuit structure that is simple to analyze.
- If cost isn't a problem, a brute force method for designing a hazard-free circuit is just to use the complete sum of the logic function.
- Everything that has been said about AND-OR circuits applies to NAND-NAND circuits, and everything said about OR-AND circuits applies to NOR-NOR circuits.
- It is important to note that most hazards aren't actually that dangerous. Any combinational circuit can be analyzed for the presence of a hazard, however any well designed synchronous digital system is specifically structured to prevent the occurrence of hazards. Hazard analysis is typically only necessary in asynchronous sequential circuits.

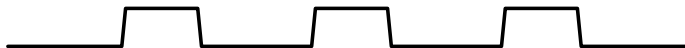
## 17 3/04/2020

### 17.1 Lecture Slides

- Time is officially a component of circuit designs, now. Why? “Time is real,” and getting used to it now will make future designs much more practical.
- We look at time using a timing diagram, which is just a twist on the normal truth table. 0 is represented as a low wave value and 1 is represented as a high wave value.



- While we do operate in 0 and 1, the values always exist and are continuous. Timing diagrams are what we actually work with and show behaviors such as verifying functionality and time impacts.



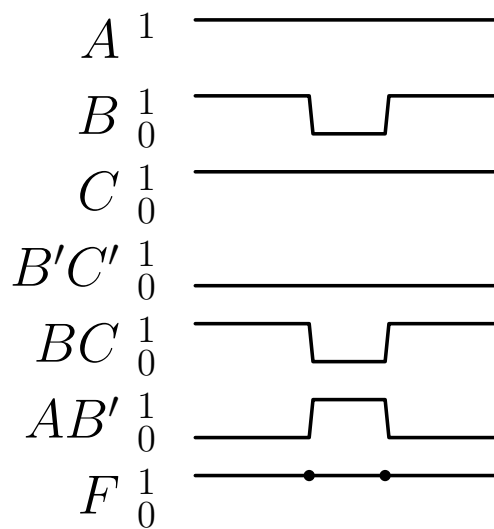
- Three variables aren't enough to draw a timing diagram for a function such as  $F = B'C' + BC + AB'$ . Those three variables only have eight possible combinations, which isn't enough. To draw the timing diagram, we need to look at all of the possible combinations.
- Determining all of the potential transitions can be annoying. We need to consider design behaviors.
- Recall that Gray Code is used with sensors because there is less change between values leaving less room for error. We can use a similar approach here to identify places where errors can occur. To find our “edge cases,” we just need to look at our K-Map. First we are going to identify where these edge cases can occur, and then we will understand what is going on at these edge cases.
- An edge case, when working in SOP, is a transition between cells containing minterms. Specifically, these occur with adjacent, but not overlapping, prime implicants. A transition inside of a prime implicant doesn't matter as its stable.

- Problem 17.1.1: Draw the timing diagram for  $F = B'C' + BC + AB'$ .

$F$ :

		$AB$		$A$	
		00	01	11	10
$C$	0	0 1	2	6	4 1
	1	1	3 1	7 1	5 1
		$B$			

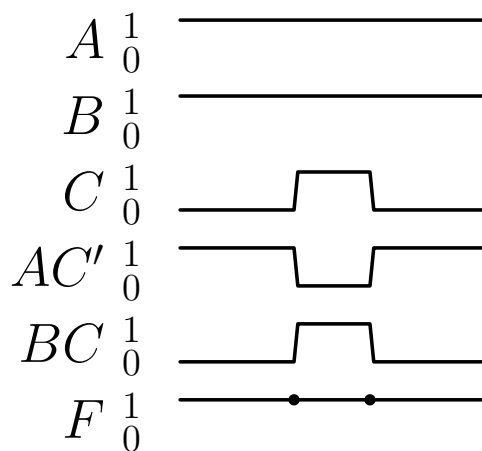
The transition between  $111 \leftrightarrow 101$  is left uncovered (Note: Yes, you can draw implicants connecting these two but here she only drew the original function)!



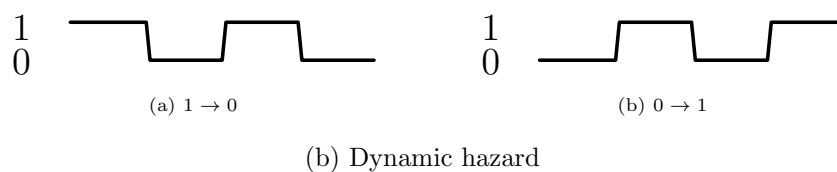
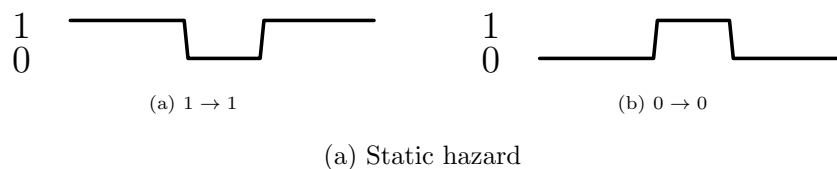
- Problem 17.1.2: Draw the timing diagram for  $F = AC' + BC$ .

$F$ :

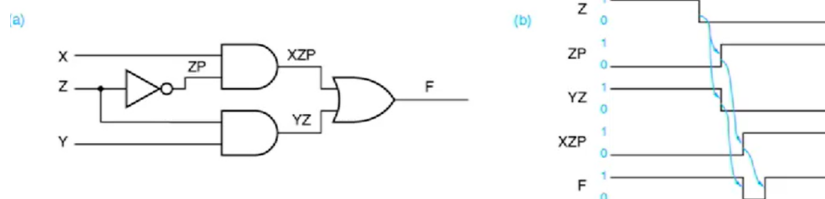
		$AB$		$A$	
		00	01	11	10
$C$	0	0	2	6 1	4 1
	1	1	3 1	7 1	5
		$B$			



- Transition times in gates and other electronical technologies result in hazards. Let's first go over static hazards, which exist when two adjacent one's in a K-Map are not covered by a prime implicant in the resulting minimal function. To fix it, include additional prime implicants whenever there are two adjacent one's. While the resulting function will no longer be minimal, there will be no undesired behavior.
- A hazard can also be described as the brief period of time where the value of an output changes even though the input's change should have left the output unaffected. A static 1 hazard is occurs when it should stay 1, and a static 0 hazard occurs when it should stay 0.



- Static-1 Hazards look like this between a circuit diagram and the timing diagram.

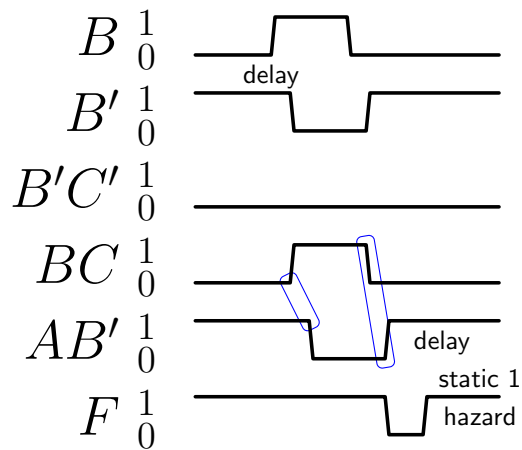


- Problem 17.1.3: Identify the hazard for  $F = B'C' + BC + AB'$ .

$F$ :

		$AB$		$A$	
		00	01	11	10
$C$	0	0 1	2	6	4 1
	1	1	3 1	7 1	5 1
		$B$			

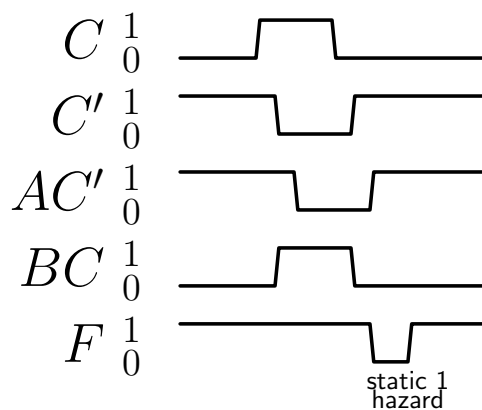
If you specify that B is the only factor (it is the only thing that changes in  $111 \leftrightarrow 101$ ), you don't have to draw A and B in the timing diagram.



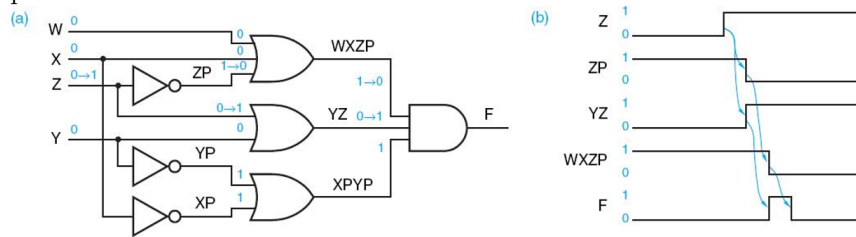
- Problem 17.1.4: Identify the hazard for  $F = AC' + BC$ .

$F$ :

		$AB$		$A$	
		00	01	11	10
$C$	0	0	2	6 1	4 1
	1	1	3 1	7 1	5
		$B$			



- You don't need to know much about static-0 hazards besides "they happen in POS."



- So how do we fix these? We need to review a previously discussed subject: Function equivalence. This is when two functions accomplish the same thing but with different terms. This is crucial in making things smaller and faster, but can come at the price of getting hazards.

## 17.2 Assigned Readings

- An adder combines two arithmetic operands using the addition rules previously discussed.
- An adder can perform subtraction as the addition of the minuend and the complemented negated subtrahend, but you can also build subtractors that perform subtraction directly.
- The simplest adder is called a half adder, and it adds two 1-bit operands  $A$  and  $B$  producing a 2-bit sum. The sum can range from 0 to 2 in base 10 and requires two bits to express. The low-order bit of the sum can be named the HS (or half sum) and the high-order bit can be named CO (or carry-out). We can write the following equations for HS and CO.

$$\begin{aligned}
 HS &= A \oplus B \\
 &= A \cdot B' + A' \cdot B \\
 CO &= A \cdot B
 \end{aligned}$$



## 18 3/06/2020

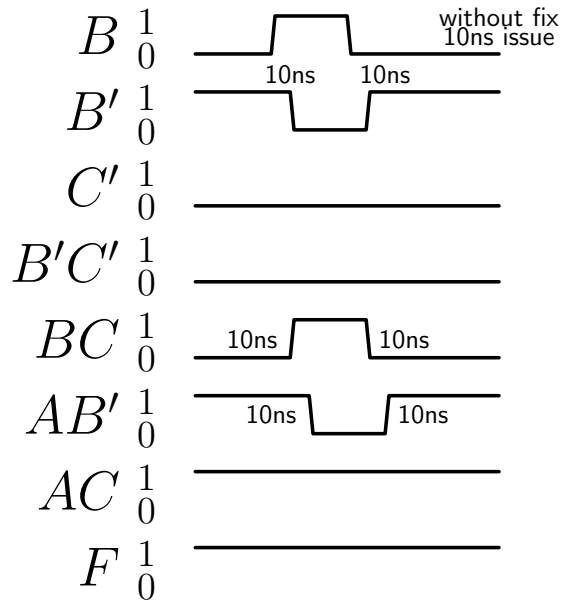
### 18.1 Lecture Readings

- Problem 18.1.1: Fix the hazard for  $F = B'C' + BC + AB'$ .

$F$ :

		$AB$				$A$			
		00	01	11	10				
$C$	0	0 1	2	6	4 1				
	1	1	3 1	7 1	5 1				
		$B$							

The hazard occurs at  $111 \leftrightarrow 101$ , so we can add  $AC$  to fix the hazard.

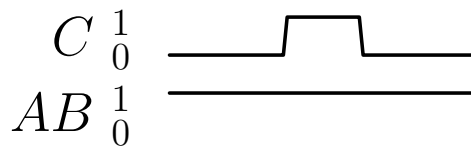


- Problem 18.1.2: Fix the hazard for  $F = AC' + BC$ .

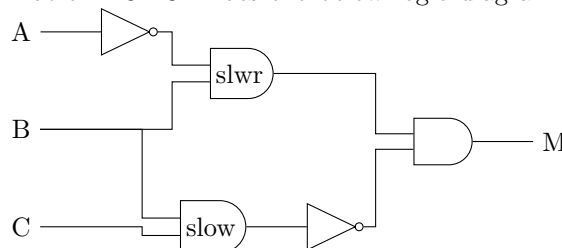
$F$ :

		$AB$				$A$			
		00	01	11	10				
$C$	0	0	2	6 1	4 1				
	1	1	3 1	7 1	5				
		$B$							

To fix the hazard, we need to fix  $AB$ , which is where the transition  $110 \leftrightarrow 111$  occurs. Since  $C$  is the only value changing, we can exclude  $A$  and  $B$  from the timing diagram.

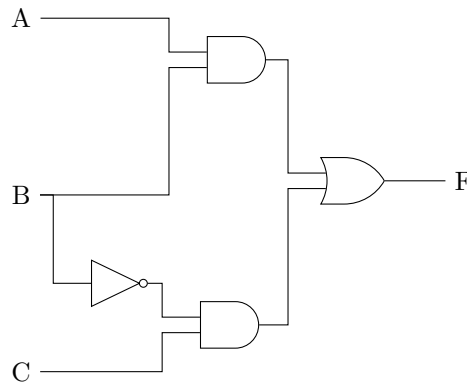


- Dynamic hazards occur when the output is supposed to change but oscillates for a brief period of time before settling down. These occur because of race conditions. We identify these through timing diagrams, logic gate inspections, and analysis.
- Problem 18.1.3: Does the below logic diagram have a dynamic hazard?



This logic diagram doesn't have any dynamic hazards, as they all cancel out. The delay the first not gate introduces to the slwr (stand-in for slower) gate is equivalent to the delay introduced to the second not gate by the slow gate.

- Problem 18.1.4: Does the below logic diagram have a dynamic hazard?



This logic diagram does indeed have a dynamic hazard. The first path ( $A \rightarrow \text{and} \rightarrow \text{or} \rightarrow F$ ) goes through only one logic gate, the and gate. The second path ( $B \rightarrow \text{not} \rightarrow \text{and} \rightarrow \text{or} \rightarrow F$ ), however, goes through two, which will introduce greater delay and thus brings a race condition.

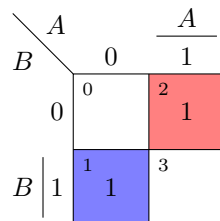
- So how do we solve these dynamic hazards? It's complicated. You should first try a two level design, described below. If that doesn't work, you can also try clocked synchronization.
  - If SOP: Check your not gates  $\rightarrow$  and gates  $\rightarrow$  or gates.
  - If POS: Check your not gates  $\rightarrow$  or gates  $\rightarrow$  and gates.

There should be a uniform number of gates in all paths.

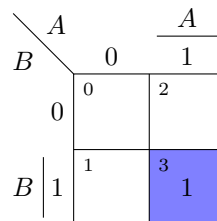
- MSI stands for medium scale integration and involves from 20 to about 200 logic gates. Our AND gate IC has four gates, making it SSI. MSI is the direction of complexity we will soon be approaching.
- The MSI combinational logic devices we will be looking at are adders, subtractors, multiplexors, decoders, encoders, and shifters.
- For this next section, you must recall binary addition. If you need a refresher, go read pages 6-8.
- The half adder is, bluntly, the addition that happens on the right most bit. There is no carry in, which is a key component of half adding. Furthermore, a sum bit and a carry out bit are produced.
- Below is an example of an implemented half adder.

Arithmetic: $A + B$		Carry bit	Sum bit
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

This half adder table essentially creates two K-Maps, both shown below.

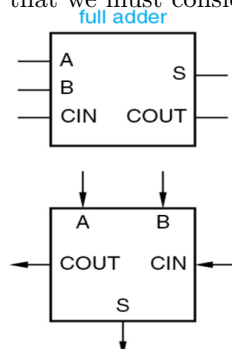


(a)  $S = A'B + AB' \rightarrow S = A \oplus B$



(b)  $C = AB$

- A full adder is what we use to consider the remainder of the bits when working with addition between two numbers. There is now also a carry in that we must consider.



- So what does a full adder look like when implemented?

$A + B + C_{in}$			$C_{out}$	$S$
$A$	$B$	$C_{in}$		
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

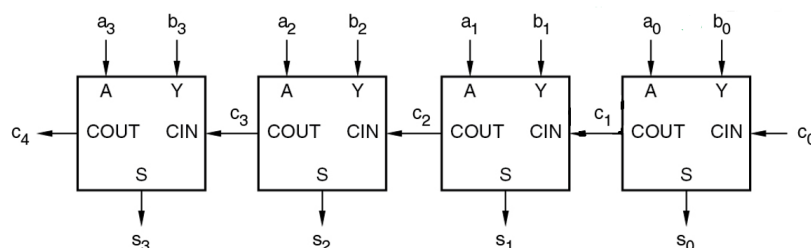
$AB$		$A$			
$C$	$B$	00	01	11	10
		0	2	6	4
0			1		1
1		1	3	7	5
		1		1	

(a)  $S$

$AB$		$A$			
$C$	$B$	00	01	11	10
		0	2	6	4
0				1	
1		1	3	7	5
			1	1	1

(b)  $C$

- If we want to calculate multiple complex numbers, we need to chain together multiple FAs, because at the end of the day one FA is still a single bit.



- The biggest issue with this approach comes from propagation delays. The FA's need a long time until they get the correct value, which can significantly lower overall efficiency.
- The solution to this comes with fast adders, which are adders specifically designed to address propagation delay. We are only briefly looking at these to understand their purpose and key ideas.
- Propagation delays are a significant problem with ripple delays, so the solution is just to carry out all of the delays at once. The only problem is that this can create *extremely* complicated.
- To avoid “gate explosion,” we divide and conquer. This is known as HCLA or the Hierarchical Carry Look Ahead Adder (also known as the Grouped Carry Lookahead Adder).