

Welcome to ENEE244-020x

Digital Logic Design

Instructor: Dana Dachman-Soled

UTFs:

- August Van Geertruyden (0201)
- Andy Zhen (0202)
- Ash Dorsey (0203)
- Natalie Nguyen (0204)
- Ojasvi Sharma (0205)

Administrative Stuff

- How to reach me:

Instructor OH!

- Email: danadach@umd.edu

Th: 9:30-10:30am

- Office: Iribe 5238

- Grading:

- Homework: 10%, Small quizzes: 5%,

- 2 midterms: 50%, Final exam (cumulative): 35%.

- For detailed information see syllabus. Feel free to contact me with any questions.

Homeworks

- Must be legible for TAs to grade it
 - Either typed or handwritten and scanned (no photos accepted) using an app such as Adobe scan
- May work in small groups but final submitted work must be your own
- The use of AI/LLM's is not allowed
- Homework is graded for both correctness and effort
- If we see signs of cheating (e.g. two very similar submissions or suspected LLM use) we will ask the student(s) to attend office hours to explain how they derived each answer

Announcements

- No discussion on Wednesday, September 2nd.
- First discussion is on Wednesday, September 9th.
 - There will be a quiz
- First homework is assigned (see course webpage)
 - Due date: Thursday, September 10th at 11:59pm
 - Submission is on Canvas

What is this course about?

- Given a Boolean function $f: \{0,1\}^n \rightarrow \{0,1\}$, design a circuit computing f .
 - Given only a truth table for f

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$(x_1 \text{ AND } x_2) \text{ OR } x_3$

There are many ways of expressing this function.

What is this course about?

- Given a Boolean function $f: \{0,1\}^n \rightarrow \{0,1\}$, design a circuit computing f .
 - Given only a truth table for f
 - Study techniques for finding the minimal circuit computing f
- Understand the design of basic circuit components
 - Adders, comparators, decoders, encoders, multiplexers

What is this course about?

- Previous topics were all examples of combinational networks.
 - Output depends only on the inputs at that instant.
- Sequential networks
 - Output depends on the state of memory
 - Flip-flops—basic logic element of sequential networks
 - Synchronous sequential networks—behavior determined by a clock signal
- Memory and Programmable Logic
 - hardware whose logic function can be configured/programmed
 - Field-programmable gate array (FPGA)

List of Topics

- Number systems
- Boolean algebra
- Combinational networks
 - Design
 - Analysis
- Synchronous sequential networks
 - Design
 - Analysis

Introduction

Digital vs. Analog

- Two general ways to represent information
 - Digital: information is denoted by a finite sequence of digits. This form is discrete.
 - Analog: a continuum is used to denote the information.
- Examples:
 - Digital vs. analog watch (range of angular displacement)
 - MP3 vs. cassette tape (quantities of magnetism stored on surface of tape)

Digital vs. Analog

Digital	Analog
Any degree of precision is possible by using more digits.	Measure to read information, precision is the number of digits obtained. Limitation on precision.
Can copy without degradation. e.g. digital recording.	Copying causes degradation. e.g. analog tape.
Groups of numbers can be compressed by finding patterns. Easy to manipulate.	Harder to compress and manipulate.

Digital Computers

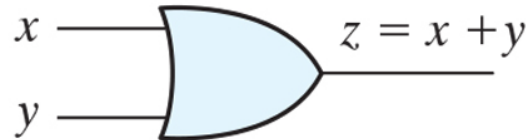
- The electrical values (i.e. voltages) of signals in a circuit are treated as integers (0 and 1)
- Alternative is analog, where electrical values are treated as real numbers.
- Usually assume only use two voltages: high and low (square wave).
 - Signal at high voltage: “1”, “true”, “set”, “asserted”
 - Signal at low voltage: “0”, “false”, “unset”, “deasserted”
- This will be our level of abstraction for most of the topics covered in this class.

Class Exercise Warm Up

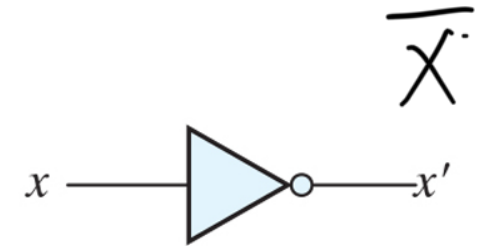
We will attempt to convert a truth table to a circuit with the following possible gates:



(a) Two-input AND gate



(b) Two-input OR gate



(c) NOT gate or inverter

0	0	0
0	1	0
1	0	0
1	1	1

0	0	0
0	1	1
1	0	1
1	1	1

0	1
1	0

Representing Data Digitally

Positional Number Systems

- The decimal number system:
 - $932.86 = 9 \times 10^2 + 3 \times 10^1 + 2 \times 10^0 + 8 \times 10^{-1} + 6 \times 10^{-2}$
- In general, can consider base r representation
 - $$N = d_{n-1}d_{n-2} \cdots d_1d_0.d_{-1} \cdots d_{-m}$$
$$= d_{n-1} \times r^{n-1} + d_{n-2} \times r^{n-2} + \cdots + d_0 \times r^0 + d_{-1} \times r^{-1} + \cdots + d_{-m} \times r^{-m}$$
- We will be mainly concerned with the binary number system.
 - Two digit symbols: 0,1
 - A digit is referred to as a bit

Positional Number Systems

- Binary, Decimal, Octal, Hexadecimal
- Octal:
 - Base 8 ($r=8$)
- Hexadecimal:
 - Base 16 ($r = 16$)
 - Digit symbols: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Positional Number Systems

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Basic Arithmetic Operations—

Examples

- Addition:

$$101100011 + 100110010$$

$$\begin{array}{r} 101100011 \\ + 100110010 \\ \hline 1010010101 \end{array}$$

- Subtraction:

$$101100011 - 100110010$$

$$\begin{array}{r} 101100011 \\ - 100110010 \\ \hline 000110001 \end{array}$$

- Multiplication:

$$1011 \times 1001$$

$$\begin{array}{r} 1011 \\ \times 1001 \\ \hline 1011 \end{array}$$

- Division:

$$100100/1100$$

$$\begin{array}{r} 000x \\ 000xx \\ 1011xxx \\ \hline 1100011 \end{array}$$

$$\begin{array}{r} 11 \\ 1100 \overline{) 1100100} \\ \underline{1100} \\ 00100 \end{array}$$

Switching Bases

Algorithmic techniques

- There are many ways to convert bases, e.g., can eyeball it.
- Outline two algorithmic approaches to converting bases.
- Can both be straightforwardly implemented on a computer.

Polynomial method of number conversion

- Convert from base r_1 to base r_2
- Express number as polynomial in base r_1
$$- N = d_2 \times r_1^2 + d_1 \times r_1^1 + d_0 \times r_1^0$$
- Switch each digit symbol d_i to its base r_2 representation and each base symbol r_1 to its base r_2 representation.
- Evaluate the polynomial in base r_2 .

Polynomial Method of Number Conversion

- Example: convert from hexadecimal to decimal
- Hexadecimal number: A78E
 - $A78E = A \times (10_{(16)})^3 + 7 \times (10_{(16)})^2 + 8 \times (10_{(16)})^1 + E \times (10_{(16)})^0$
 - $A78E = (10) \times (16)^3 + (7) \times (16)^2 + (8) \times (16)^1 + (14) \times (16)^0$
 - $A78E = 42894$
- **Use this method when converting a number into decimal form (e.g. binary to decimal)
- Why?

Iterative Method of Number Conversion

- Convert from base r_1 to base r_2 .
- Perform repeated division by r_2 . The remainder is the digit of the base r_2 number.
- Example: Convert 50 from decimal to binary
 - Divide 50 by 2, get 25 remainder 0
 - Divide 25 by 2, get 12 remainder 1
 - Divide 12 by 2, get 6 remainder 0
 - Divide 6 by 2, get 3 remainder 0
 - Divide 3 by 2, get 1 remainder 1
 - Divide 1 by 2, get 0 remainder 1
- Answer is: 110010
- Can verify using the polynomial method
- **Use when converting from decimal to another base. (e.g. decimal to binary)
- Why?

Iterative Method for Converting Fractions

- Convert from base r_1 to base r_2 .
- Perform repeated **multiplication** by r_2 . The **integer part** is the digit of the base r_2 number.
- Ex: Convert .40625 from decimal to binary
 - Multiply .40625 by 2, get 0 + .8125
 - Multiply .8125 by 2, get 1 + .625
 - Multiply .625 by 2, get 1 + .25
 - Multiply .25 by 2, get 0 + .5
 - Multiply .5 by 2, get 1 + 0
- Answer is: .01101
- Can verify using the polynomial method

Special Conversion Procedures

- When converting between two bases in which one base is a power of the other, conversion is simplified.
- Ex: Convert from 1101 0110 1111 1001 from binary to hexadecimal:
 - $1101 = 13 = D$
 - $0110 = 6$
 - $1111 = 15 = F$
 - $1001 = 9$
- Answer: D6F9
- What about from *binary* to *octal*?

Signed Numbers and Complements

Range of represented numbers

- Let ℓ be the number of binary digits that can be stored.
- Example: Store data in a single byte (8 bits).
- Using a single byte can represent unsigned numbers from 0 to 255 ($2^8 = 256$ different values).
- Alternatively, can represent the signed numbers from -128 to 127 in same amount of space ($2^7 = 128$).

Signed Numbers and Complements

- How to denote if a number is positive or negative?
 - Use a sign bit: 0_s1001 denotes positive 9, 1_s1001 denotes negative 9. This representation is called the **sign-magnitude representation**.
 - This works, but it will be convenient to use a different representation of negative numbers.
- Two methods: **r 's complement** and **$(r - 1)$'s complement**.

r 's Complement

- r 's complement of $N = r^\ell - N = 10^\ell - N$
- In our example (one byte of memory, to represent -9, (where $9 = 1001$ in binary), compute $(10)^8 - 1001 = 100000000 - 1001 = 11110111$
- Notice for negative numbers, **most significant bit** is always 1. For positive numbers, **most significant bit** is always 0.
- This bit is therefore called the **sign bit**.
- Aside: This is equivalent to doing arithmetic modulo r^ℓ .

Subtraction Terminology

$$\textcircled{10110} - \textcircled{1000}$$

minuend

subtrahend

Subtraction using r 's complement

- Just do addition as usual
- Ignore highest order carry
- This is always correct unless there is **overflow**.

Example of subtraction using r 's complement

- Assume $\ell = 6$:

- Compute:

$$11010 - 00111$$

- Compute:

$$00111 - 11010$$

Overflow in r 's complement

- Overflow occurs in the following cases:

Operation	Operand A	Operand B	Result
A + B	≥ 0	≥ 0	< 0
A+B	< 0	< 0	≥ 0
A-B	≥ 0	< 0	< 0
A-B	< 0	≥ 0	≥ 0

- These conditions are the same as:
Carry-In to sign position $\neq$ Carry-Out from sign position

Example of Overflow in r 's complement

- Assume $\ell = 6$
- Compute:

$$11100 + 10111$$

$(r - 1)$'s Complement

- $(r - 1)$'s complement of $N = r^\ell - r^{-m} - N = 10^\ell - 10^{-m} - N$
- For integers, $m = 0$, so subtract from $10^n - 1$
- In our example, to represent -9, (where $9 = 1001$ in binary), compute $(10)^8 - 1 - 1001 = 11111111 - 1001 = 11110110$
- This corresponds to flipping the bits of 00001001.
- Again, for negative numbers, n th digit is always 1. For positive numbers, n th digit is always 0.
- There are now two ways to represent 0: 00000000 or 11111111

Subtraction using $(r - 1)$'s complement

- Do addition as usual
- If there is an end carry, add it to the least significant bit.
- Most significant bit tells you the sign (unless overflow occurs).

Example of subtraction using $(r - 1)$'s complement

- Assume $\ell = 6$

- Compute:

11010 - 00111

- Compute:

00111 - 11010

Fast(er) way to compute 2's complement

- To form the 2's complement of 0110 1010:
 - Take the 1s complement: 1001 0101
 - Then add 1: 1001 0110

Ways to Represent Signed Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	—	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
-8	1000	—	—

Advantages/Disadvantages of 1's vs. 2's complement

1s complement	2s complement
Easy to compute (just flip bits)	Harder to compute (flip bits and add one)
Harder to manipulate (e.g., for subtraction, need to add in extra carry.)	Easy to manipulate (e.g., subtraction is the same as addition—no extra hardware needed)