EXPLORING DIGITAL LOGIC

With Logisim-Evolution

GEORGE SELF

July 2018 – Edition 6.0

George Self: *Exploring Digital Logic,* With Logisim-Evolution

BRIEF CONTENTS

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

Part I

THEORY

Digital Logic, as most computer science studies, depends on a foundation of theory. This part of the book concerns the theory of digital logic and includes binary mathematics, gate-level logic, Boolean algebra, and simplifying Boolean expressions. An understanding of these foundational conceptsis essential before attempting to design complex combinational and sequential logic circuits.

1 INTRODUCTION

1.1 PREFACE

1.1.1 *Introduction to the Study of Digital Logic*

Digital logic is the study of how electronic devices make decisions. It functions at the lowest level of computer operations: bits that can either be "on" or "off" and groups of bits that form "bytes" and "words" that control physical devices. The language of digital logic is Boolean algebra, which is a mathematical model used to describe the logical function of a circuit; and that model can then be used to design the most efficient device possible. Finally, various simple devices, such as adders and registers, can be combined into increasingly complex circuits designed to accomplish advanced decision-making tasks.

1.1.2 *Introduction to the Author*

I have worked with computers and computer controlled systems for more than 30 years. I took my first programming class in 1976; and, several years later, was introduced to digital logic while taking classes to learn how to repair computer systems. For many years, my profession was to work on computer systems, both as a repair technician and a programmer, where I used the principles of digital logic daily. I then began teaching digital logic classes at Cochise College and was able to share my enthusiasm for the subject with Computer Information Systems students. Over the years, I have continued my studies of digital logic in order to improve my understanding of the topic; I also enjoy building logic circuits on a simulator to solve interesting challenges. It is my goal to make digital logic understandable and to also ignite a lifelong passion for the subject in students.

1.1.3 *Introduction to This Book*

This book has two goals:

1. AUDIENCE. Many, perhaps most, digital logic books are designed for third or fourth year electronics engineering or computer science students and presume a background that includes advanced mathematics and various engineering classes. For example, it is possible to find a digital logic book that discusses topics like physically building circuits from discrete components and then calculating the heat rise of those circuits while operating at maximum capacity. This book,

though, was written for students in their second year of a Computer Information Systems program and makes no assumptions about prior mathematics and engineering classes.

2. Cost. Most digital logic books are priced at $150 (and up) but this book is published under a Creative Commons license and, though only a tiny drop in the proverbial textbook ocean, is hoped to keep the cost of books for at least one class as low as possible.

Following are the features for the various editions of this book:

1. 2012. Prior to 2012, handouts were given to students as they were needed during class; however, it was in this year that the numerous disparate documents were assembled into a cohesive book and printed by a professional printing company.

2. 2013. A number of complex circuits were added to the book, including a Hamming Code generator/checker, which is used for error detection, and a Central Processing Unit (CPU) using discrete logic gates.

3. 2014. New material on Mealy and Moore State Machines was included, but the major change was in the laboratory exercises where five Verilog labs were added to the ten gate-level simulation labs.

4. 2015. New information was added about adding/subtracting Binary Coded Decimal (BCD) numbers and representing floating point numbers in binary form; and all of the laboratory exercises were re-written in Verilog. Also, the book was reorganized and converted to LaTeXfor printing.

5. 2018. The labs were re-written using *Logisim-evolution* because students find that system easier to understand than iVerilog.

This book was written with LaTeX using TeXstudio. The source for this book is available at GITHUB, http://bit.ly/2w6qU2C, and anyone is welcomed to fork the book and develop their own version.

DISCLAIMER

I wrote, edited, illustrated, and published this book myself. While I did the best that I could, there are, no doubt, errors. I apologize in advance if anything presented here is factually erroneous; I'll correct those errors as soon as they are discovered. I'll also correct whatever typos I overlooked, despite TeXstudio's red squiggly underlines trying to tell me to check my work. –George Self

1.1.4 *About the Creative Commons License*

This book is being released under the Creative Commons 0 license, which is the same as public domain. That permits people to share, remix, or even rewrite the work so it can be used to help educate students wherever they are studying digital logic.

1.2 ABOUT DIGITAL LOGIC

1.2.1 *Introduction*

Digital logic is the study of how logic is used in digital devices to complete tasks in fields as diverse as communication, business, space exploration, and medicine (not to mention everyday life). This definition has two main components: logic and digital. *Logic* is the branch of philosophy that concerns making reasonable judgment based on sound principles of inference. It is a method of problem solving based on a linear, step-by-step procedure. *Digital* is a system of mathematics where only two possible values exist: *True* and *False* (usually represented by 1 and 0). While this approach may seem limited, it actually works quite nicely in computer circuits where *True* and *False* can be easily represented by the presence or absence of voltage.

Digital logic is not the same as programming logic, though there is some relationship between them. A programmer would use the constructs of logic within a high-level language, like Java or C++, to get a computer to complete some task. On the other hand, an engineer would use digital logic with hardware devices to build a machine, like an alarm clock or calculator, which executes some task. In broad strokes, then, programming logic concerns writing software while digital logic concerns building hardware.

Digital logic may be divided into two broad classes:

1. Combinational Logic, in which the outputs are determined solely by the input states at one particular moment with no memory of prior states. An example of a combinational circuit is a simple adder where the output is determined only by the values of two inputs.

2. Sequential Logic, in which the outputs depend on both current and prior inputs, so some sort of memory is necessary. An example of a sequential circuit is a counter where a sensed new event is added to some total contained in memory.

Both combinational and sequential logic are developed in this book, along with complex circuits that require both combinational and sequential logic.

1.2.2 *A Brief Electronics Primer*

Electricity is nothing more than the flow of electrons from point *A* to point *B*. Along the way, those electrons can be made to do work as they flow through various devices. Electronics is the science (and, occasionally, art) of using tiny quantities of electricity to do work. As an example, consider the circuit schematic illustrated in Figure 1.1:

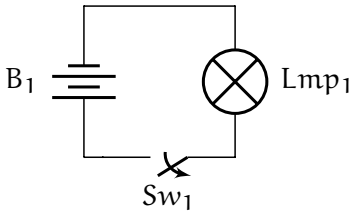

Figure 1.1: Simple Lamp Circuit

In this diagram, battery *B1* is connected to lamp *Lmp1* through switch *Sw1*. When the switch is closed, electrons will flow from the negative battery terminal, through the switch and lamp, and back to the positive battery terminal. As the electrons flow through the lamp's filament it will heat up and glow: light!

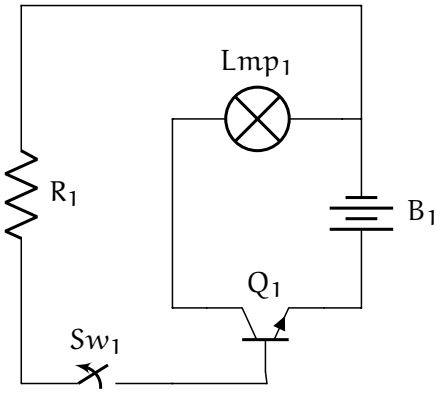

Figure 1.2: Transistor-Controlled Lamp Circuit

A slightly more complex circuit is illustrated in Figure 1.2. In this case, a transistor, *Q1*, has been added to the circuit. When switch *Sw1* is closed, a tiny current can flow from the battery, through the transistor's emitter (the connection with the arrow) to its base, through the switch, through a resistor *R1*, and back to the positive terminal of the battery. However, that small current turns on transistor *Q1* so a much larger current can also flow from the battery, through the collector port, to lamp *Lmp1*, and back to the battery. The final effect is the same for both circuits: close a switch and the lamp turns on. However, by using a transistor, the lamp can be controlled by applying any sort of positive voltage to the transistor's base; so the lamp could be controlled by a switch, as illustrated, or by the output of some other

electronic process, like a photo-electric cell sensing that the room is dark.

Using various electronic components, like transistors and resistors, digital logic "gates" can be constructed, and these become the building blocks for complex logic circuits. Logic gates are more thoroughly covered in a later chapter, but one of the fundamental logic gates is an OR gate, and a simplified schematic diagram for an OR gate is in Figure 1.3. In this circuit, any voltage present at *Input A* or *Input B* will activate the transistors and develop voltage at the output.

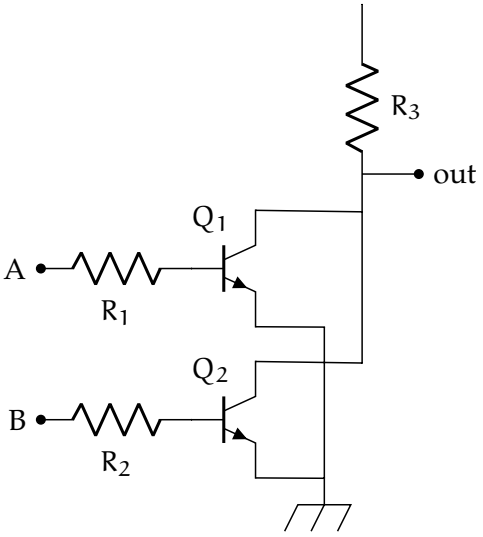

Figure 1.3: Simple OR Gate Using Transistors

Figure 1.4[1] shows a more complete OR gate created with what are called "N-Channel metal-oxide semiconductor" (or *nMOS*) transistors. The electronic functioning of this circuit is beyond the scope of this book (and is in the domain of electronics engineers), but the important point to keep in mind is that this book concerns building electronic circuits designed to accomplish a physical task rather than write a program to control a computer's processes.

Early electronic switches took the form of vacuum tubes, but those were replaced by transistors which were much more energy efficient and physically smaller. Eventually, entire transistorized circuits, like the OR gate illustrated in Figure 1.4, were miniaturized and placed in a single Integrated Circuit (IC), sometimes called a "chip," smaller than a postage stamp.

ICs make it possible to produce smaller, faster, and more powerful electronics devices. For example, the original ENIAC computer, built in 1946, occupied more than 1800 square feet of floor space and required 150KW of electricity, but by the 1980s integrated circuits in hand-held

1 This schematic diagram was created by Ramón Jaramillo and found at http://www.texample.net/tikz/examples/or-gate/

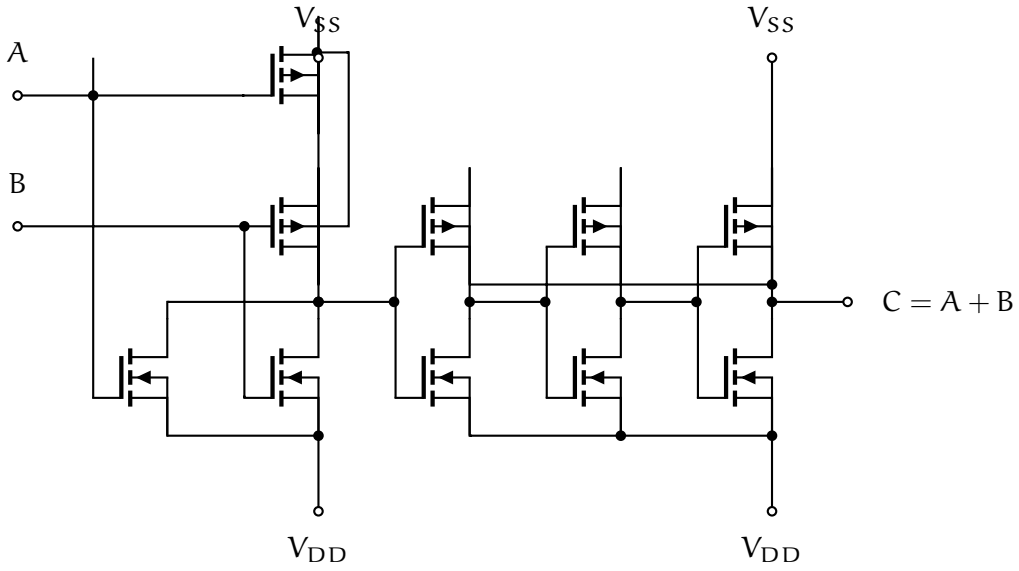

Figure 1.4: OR Gate

calculators were more powerful than that early computer. Integrated circuits are often divided into four classes:

1. Small-scale integration with fewer than 10 transistors
2. Medium-scale integration with 10-500 transistors
3. Large-scale integration with 500-20,000 transistors
4. Very large-scale integration with 20,000-1,000,000 transistors

Integrated circuits are designed by engineers who use software written specifically for that purpose. While the intent of this book is to afford students a foundation in digital logic, those who pursue a degree in electronics engineering, software engineering, or some related field, will need to study a digital logic language, like iVerilog.

1.3 BOOLEAN ALGEBRA

1.3.1 *History*

The Greek philosopher Aristotle founded a system of logic based on only two types of propositions: *True* and *False*. His bivalent (two-mode) definition of truth led to four foundational laws of logic: the Law of Identity (*A is A*); the Law of Non-contradiction (*A is not non-A*); the Law of the Excluded Middle (*either A or non-A*); and the Law of Rational Inference. These laws function within the scope of logic where a proposition is limited to one of two possible values, like *True* and *False*; but they do not apply in cases where propositions can hold other values.

The English mathematician George Boole (1815-1864) sought to give symbolic form to Aristotle's system of logic. Boole wrote a treatise on the subject in 1854, titled *An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities*, which codified several rules of relationship between mathematical quantities limited to one of two possible values: *True* or *False*, 1 or 0. His mathematical system became known as *Boolean Algebra*.

All arithmetic operations performed with Boolean quantities have but one of two possible outcomes: 1 or 0. There is no such thing as 2 or -1 or $\frac{1}{3}$ in the Boolean world and numbers other than 1 and 0 are invalid by definition. Claude Shannon of MIT recognized how Boolean algebra could be applied to on-and-off circuits, where all signals are characterized as either *high* (1) or *low* (0), and his 1938 thesis, titled *A Symbolic Analysis of Relay and Switching Circuits*, put Boole's theoretical work to use in land line telephone switching, a system Boole never could have imagined.

While there are a number of similarities between Boolean algebra and real-number algebra, it is important to bear in mind that the system of numbers defining Boolean algebra is severely limited in scope: 1 and 0. Consequently, the "Laws" of Boolean algebra often differ from the "Laws" of real-number algebra, making possible such Boolean statements as $1 + 1 = 1$, which would be considered absurd for real-number algebra.

1.3.2 *Boolean Equations*

Boolean algebra is a mathematical system that defines a series of logical operations performed on a set of variables. The expression of a single logical function is a *Boolean Equation* that uses standardized symbols and rules. Boolean expressions are the foundation of digital circuits.

Sometimes Boolean Equations are called SWITCHING EQUATIONS

Binary variables used in Boolean algebra are like variables in regular algebra except that they can only have two values: one or zero. Boolean algebra includes three primary logical functions: AND, OR, and NOT; and five secondary logical functions: NAND, NOR, XOR, and XNOR, and Buffer. A Boolean equation defines an electronic circuit that provides a relationship between input and output variables and takes the form of:

XNOR is sometimes called EQUIVALENCE *and Buffer is sometimes called* TRANSFER.

$$C = A * B \tag{1.1}$$

where A and B are binary input variables that are related to the output variable C by the function AND (denoted by an asterisk).

In Boolean algebra, it is common to speak of *truth*. This term does not mean the same as it would to a philosopher, though its use is based on Aristotelian philosophy where a statement was either *True* or

False. In Boolean algebra as used in electronics, *True* commonly means "voltage present" (or "1") while *False* commonly means "voltage absent" (or "0"), and this can be applied to either input or output variables. It is common to create a *Truth Table* for a Boolean equation to indicate which combination of inputs should evaluate to a *True* output and Truth Tables are used very frequently throughout this book.

1.4 ABOUT THIS BOOK

This book is organized into two main parts:

1. THEORY. Chapters two through six cover the foundational theory of digital logic. Included are chapters on binary mathematics, Boolean algebra, and simplifying Boolean expressions using tools like Karnaugh maps and the Quine-McCluskey method.

2. PRACTICE. Chapters seven through nine expand the theory of digital logic into practical applications. Covered are combinational and sequential logic circuits, and then simulation of various physical devices, like elevators.

There is also an accompanying lab manual where *Logisim-evolution* is used to build digital logic circuits. By combining the theory presented in this book along with the practical application presented in the lab manual it is hoped that students gain a thorough understanding of digital logic.

By combining the theoretical background of binary mathematics and Boolean algebra with the practical application of building logic devices, digital logic becomes understandable and useful.

Part II

PRACTICE

Once the foundations of digital logic are mastered it is time to consider creating circuits that accomplish some practical task. This part of the book begins with the simplest of combinational logic circuits and progresses to sequential logic circuits and, finally, to complex circuits that combine both combinational and sequential elements.

Part III

APPENDIX

a

BOOLEAN PROPERTIES AND FUNCTIONS

AND Truth Table

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Truth Table

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

XOR Truth Table

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

NAND Truth Table

Inputs		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR Truth Table

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

XNOR Truth Table

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

NOT Truth Table

Input	Output
0	1
1	0

Buffer Truth Table

Input	Output
0	0
1	1

Table a.1: Univariate Properties

Property	OR	AND
Identity	$A + 0 = A$	$1A = A$
Idempotence	$A + A = A$	$AA = A$
Annihilator	$A + 1 = 1$	$0A = 0$
Complement	$A + A' = 1$	$AA' = 0$
Involution	$(A')' = A$	

Table a.2: Multivariate Properties

Property	OR	AND
Commutative	$A + B = B + A$	$AB = BA$
Associative	$(A + B) + C = A + (B + C)$	$(AB)C = A(BC)$
Distributive	$A + (BC) = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption	$A + AB = A$	$A(A + B) = A$
DeMorgan	$\overline{A + B} = \overline{A}\,\overline{B}$	$\overline{AB} = \overline{A} + \overline{B}$
Adjacency	$AB + AB' = A$	

Table a.3: Boolean Functions

A	0	0	1	1	
B	0	1	0	1	
F_0	0	0	0	0	Zero or Clear. Always zero (Annihilation)
F_1	0	0	0	1	Logical AND: $A * B$
F_2	0	0	1	0	Inhibition: AB' or $A > B$
F_3	0	0	1	1	Transfer A to Output, Ignore B
F_4	0	1	0	0	Inhibition: $A'B$ or $B > A$
F_5	0	1	0	1	Transfer B to Output, Ignore A
F_6	0	1	1	0	Exclusive Or (XOR): $A \oplus B$
F_7	0	1	1	1	Logical OR: $A + B$
F_8	1	0	0	0	Logical NOR: $(A + B)'$
F_9	1	0	0	1	Equivalence: $(A = B)'$
F_{10}	1	0	1	0	Not B and ignore A, B Complement
F_{11}	1	0	1	1	Implication, $A + B'$, $B >= A$
F_{12}	1	1	0	0	Not A and ignore B, A Complement
F_{13}	1	1	0	1	Implication, $A' + B$, $A >= B$
F_{14}	1	1	1	0	Logical NAND: $(A * B)'$
F_{15}	1	1	1	1	One or Set. Always one (Identity)

b

RESOURCES

The following materials were used as resources for this book.

B.1 DIGITAL LOGIC

B.1.1 *Books*

1. Gregg, John, 1998, Ones and zeros. New York : IEEE Press.
2. Holdsworth, B and Woods, R. C, 2002, Digital logic design. Oxford : Newnes.
3. Langholz, Gideon, Kandel, Abraham and Mott, Joe L, 1998, Foundations of digital logic design. Singapore : World Scientific.
4. M.Rafiquzzaman., 2005, Fundamentals of Digital Logic and Microcomputer Design, 5th Edition. John Wiley & Sons.
5. Mano, M. Morris and Ciletti, Michael D, 2013, Digital design. Upper Saddle River, NJ : Pearson Prentice Hall.
6. Maxfield, Clive Max, Maxfield, Clive Max and Maxfield, Clive Max, 1998, Designus Maximus unleashed!. Boston : Newnes.
7. Maxfield, Clive, 2003, Bebop to the Boolean boogie. Amsterdam : Newnes.
8. Nisan, Noam and Schocken, Shimon, [no date], The elements of computing systems.
9. Petzold, Charles, 1999, Code. Redmond, Wash. : Microsoft Press.
10. Predko, Michael, 2005, Digital electronics demystified. New York : McGraw-Hill.
11. Saha, A and Manna, N, 2007, Digital principles and logic design. Hingham, Mass. : Infinity Science Press.
12. Tokheim, Roger L, 1999, Digital electronics. New York : Glencoe/McGraw-Hill.
13. Yarbrough, John M, 1997, Digital logic. Minneapolis/St. Paul : West Pub. Co.

B.1.2 *Found Online*

1. Cummings, Clifford, 2002, The Fundamentals of Efficient Synthesizable Finite State Machine Design using NC-Verilog and BuildGates [online]. 1. Sunburst Design, Inc. [Accessed 12 April 2016]. Available from: http://www.sunburst-design.com/papers/CummingsICU2002_FSMFundamentals.pdf

2. de Pablo, S., Cebrian, J. A., Herrero, L.C. and Rey, A.B., 2016, A very simple 8-bit RISC processor for FPGA [online]. 1. [Accessed 12 April 2016]. Available from: http://www.dte.eis.uva.es/Datos/Congresos/FPGAworld2006a.pdf

3. Digital Circuits - Wikibooks, open books for an open world, 2016. En.wikibooks.org [online]

4. Gray, Jan, 2000, Designing a Simple FPGA-Optimized RISC CPU and System-on-a-Chip [online]. 1. Gray Research, LLC. [Accessed 12 April 2016]. Available from: http://www.fpgacpu.org/papers/soc-gr0040-paper.pdf

5. Kaur, Ramandeep, 2016, 8 Bit RISC Processor Using Verilog HDL [online]. 1. Anuj el al Int. Journal of Engineering Research and Applications. [Accessed 12 April 2016]. Available from: http://www.ijera.com/papers/Vol4_issue3/Version%201/BV4301417422.pdf

6. Liu, Jianming, Zhang, Yunjie, Xu, Lili and Liu, Pengtao, 2013, [online]. 1. 2nd International Conference on Computer Science and Electronics Engineering. [Accessed 12 April 2016]. Available from: http://www.atlantis-press.com/php/pub.php?publication=iccsee-13&frame=http%3A//www.atlantis-press.com/php/paper-details.php%3Fid%3D4893

7. Nyasulu, Peter and Knight, J., 2003, Introduction to Verilog [online]. 1. Carleton University. [Accessed 12 April 2016]. Available from: http://www.doe.carleton.ca/~jknight/97.478/97.478_02F/PetervrlQ.pdf

8. Phelps, Andrew, 2006, Constructing an Error Correcting Code [online]. 1. Madison : University of Wisconsin. [Accessed 12 April 2016]. Available from: http://pages.cs.wisc.edu/~markhill/cs552/Fall2006/handouts/ConstructingECC.pdf

9. Programmable Logic - Wikibooks, open books for an open world, 2016. En.wikibooks.org [online]

10. User Guide, 2016. Icarus Verilog [online]

11. Verilog Tutorial -Table of Contents: ElectroSofts.com, 2016. Electrosofts.com [online]

12. Welcome To Verilog Page, 2016. Asic-world.com [online]

COLOPHON

The logic diagrams in this book are screen captures of *Logisim-evolution* circuits.

This book was typeset using the typographical look-and-feel classicthesis developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. classicthesis is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of classicthesis usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of February 13, 2019 (Edition 6.0).

Hermann Zapf's *Palatino* and *Euler* type faces (Type 1 PostScript fonts *URW Palladio L* and *FPL*) are used. The "typewriter" text is typeset in *Bera Mono*, originally developed by Bitstream, Inc. as "Bitstream Vera". (Type 1 PostScript fonts were made available by Malte Rosenau and Ulrich Dirr.)