Elements of Digital Logic

Student Guide

VERSION 1.0

WARRANTY

Parallax Inc. warrants its products against defects in materials and workmanship for a period of 90 days from receipt of product. If you discover a defect, Parallax Inc. will, at its option, repair or replace the merchandise, or refund the purchase price. Before returning the product to Parallax, call for a Return Merchandise Authorization (RMA) number. Write the RMA number on the outside of the box used to return the merchandise to Parallax. Please enclose the following along with the returned merchandise: your name, telephone number, shipping address, and a description of the problem. Parallax will return your product or its replacement using the same shipping method used to ship the product to Parallax.

14-DAY MONEY BACK GUARANTEE

If, within 14 days of having received your product, you find that it does not suit your needs, you may return it for a full refund. Parallax Inc. will refund the purchase price of the product, excluding shipping/handling costs. This guarantee is void if the product has been altered or damaged. See the Warranty section above for instructions on returning a product to Parallax.

COPYRIGHTS AND TRADEMARKS

This text is available in printed format from Parallax Inc. Because we print the text in volume, the consumer price is often less than typical retail duplication charges.

ISBN 1-928982-20-4

DISCLAIMER OF LIABILITY

Parallax Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, or any costs of recovering, reprogramming, or reproducing any data stored in or used with Parallax products. Parallax Inc. is also not responsible for any personal damage, including that to life and health, resulting from use of any of our products. You take full responsibility for your BASIC Stamp application, no matter how life-threatening it may be.

NTERNET DISCUSSION LISTS

We maintain active web-based discussion forums for people interested in Parallax products. These lists are accessible from www.parallax.com via the Support → Discussion Forums menu. These are the forums that we operate from our web site:

- BASIC Stamps – This list is widely utilized by engineers, hobbyists and students who share their BASIC Stamp projects and ask questions.
- Stamps in Class® – Created for educators and students, subscribers discuss the use of the Stamps in Class curriculum in their courses. The list provides an opportunity for both students and educators to ask questions and get answers.
- Parallax Educators –Exclusively for educators and those who contribute to the development of Stamps in Class. Parallax created this group to obtain feedback on our curricula and to provide a forum for educators to develop and obtain Teacher's Guides.
- Translators – The purpose of this list is to provide a conduit between Parallax and those who translate our documentation to languages other than English. Parallax provides editable Word documents to our translating partners and attempts to time the translations to coordinate with our publications.
- Robotics – Designed exclusively for Parallax robots, this forum is intended to be an open dialogue for robotics enthusiasts. Topics include assembly, source code, expansion, and manual updates. The Boe-Bot™, Toddler™, SumoBot®, HexCrawler and QuadCrawler robots are discussed here.
- SX Microcontrollers and SX-Key – Discussion of programming the SX microcontroller with Parallax assembly language SX – Key® tools and 3rd party BASIC and C compilers.
- Javelin Stamp – Discussion of application and design using the Javelin Stamp, a Parallax module that is programmed using a subset of Sun Microsystems' Java® programming language.

ERRATA

While great effort is made to assure the accuracy of our texts, errors may still exist. If you find an error, please let us know by sending an email to editor@parallax.com. We continually strive to improve all of our educational materials and documentation, and frequently revise our texts. Occasionally, an errata sheet with a list of known errors and corrections for a given text will be posted to our web site, www.parallax.com. Please check the individual product page's free downloads for an errata file.

Table of Contents

Preface

The purpose of Elements of Digital Logic is not simply to acquaint students with the concepts of basic logic, but to fully immerse them into a sight, sound, and hands-on logic environment. The phrase logic environment is used because this text utilizes both hardware and software simultaneously to teach logic. When fully immersed in logic, the student learns, understands, and retains more. A student with a firm grasp on the basics will more readily understand more advanced concepts. A hands-on approach gives the student greater confidence and will hold his or her attention, since it is more fun than simply reading about it. Students who go on to major in hardware, software, or any field that relies on a strong mathematics background will require a sound background in logic. Like many fields, the disciplines of hardware and software have become specialized. Because of this increased specialization, instruction in some of the basics has largely been abandoned. The irony of this situation is that, because logic is rarely taught as a stand-alone course anymore, it has become more important than ever to teach the fundamentals of logic that are still relevant and needed by today's student. This course in logic is integrated with software and hardware such that it serves as an introduction to either and both disciplines. The hoped-for result is a well-rounded student who can 'hit the ground running'.

AUDIENCE AND TEACHER'S GUIDE

This course is for anyone who is either on the path to technical greatness via his or her major, or for those who must pass a technical elective and want to enjoy their learning experience. It is an entry level book and assumes that the user has very little technical knowledge in either hardware or software. It is designed for those individuals seeking careers in any facet of hardware or software design, or careers that rely on a strong mathematics background. This text is also geared for use as both introductory classes for non-majors and as a platform for the first semester of either the software or hardware curriculum. For instructors, Parallax provides a free email list server called the Parallax Educator's list. The list serves as a platform for teachers and members of the Parallax educational team to discuss educational related issues. Students are welcome to join any of the other list servers that are available. To select or join a list, simply go to www.parallax.com/sic and follow the on-line instructions.

FOREIGN TRANSLATIONS

Parallax educational texts may be translated to other languages with the permission of Parallax (e-mail stampsinclass@parallax.com). If the user plans on doing any translations please contact Parallax to obtain the correctly-formatted MS Word documents, images, etc. Parallax also maintains a discussion group for Parallax translators that the user may join. Go to: www.yahoogroups.com and search for “Parallax Translators". This will ensure that the user is kept current on frequent text revisions.

SPECIAL CONTRIBUTORS

During the evolution of this text, many individuals contributed greatly to its success. The author would like to thank both Andy Lindsay and Steve Dill, for without their guidance this document would have become an unmanageable mess. Steve and Andy are engineers for Parallax. Together they comprise the driving force behind the Javelin Stamp and know a great deal about Microsoft Word. The author would also like to thank Aristides Alvarez for his high attention to detail in ensuring this text adhered to a consistent format. Aristides is responsible for translating all of Parallax’s educational books to as many foreign languages as possible. Rich Allred's steady hand, artistic eye, and tempered demeanor are solely responsible for all of the amazingly clear illustrations throughout this book. Rich is the Electro-Mechanical Technician for Parallax and is the key illustrator for most every Stamps In Class book in existence, and a few that are yet to come.

The author would also like to thank his lovely wife, Peggy Wells, for without her help, this document would have read like incoherent ramblings of the awkward and dim bloke she married, replete with split infinitives and every grammatically horrid thing including, but not limited to, run-on sentences.

Finally, the author would like to thank the entire Parallax team, who over the years has become his extended family, for bearing with him and supporting him every step of the way.

Chapter #1: Introduction

ACTIVITIES IN THIS CHAPTER

1. Activity 1: Software Simulator installation
2. Activity 2: BASIC Stamp Editor installation

By the end of this chapter, you should have:

1. Installed the simulator software.
2. Installed the BASIC Stamp software
3. Become familiar with the Do's and Don'ts of the hardware.

Logic in our world

People use logic countless times a day. By studying logic, its rules, and its nature, the student will be able to use it effectively, thereby enhancing his or her job skills, skill sets in other disciplines, communication skills, etc.

This manual assumes the student possesses little or no knowledge of electronics, software programming, and logic. Therefore, in the following chapters, the basic elements of electricity and logic will be addressed in a step-by-step manner. These principles are fundamental. A clear understanding of these principles is absolutely necessary before proceeding. Those who are new to logic will learn many new concepts. Those individuals who are already familiar with either software or hardware should consider reviewing this section to shore up what they already know. In order to facilitate this, the Parallax Logic Simulator (PLS) will be used to gain an understanding of basic electricity, learn a few of the more common schematic symbols, and take a brief look at the most basic logic functions.

Most people grasp the idea of IF-THEN logical concepts. Example: IF Sally opens the door, THEN the dog may enter the house. In this example, it is clear that if Sally did not open the door, the dog would have to remain outside. More often than not however, the problem is slightly more complicated, and IF-THEN logic is not sufficient. Example: If Roy has the shopping list, THEN he will buy the correct items. What can be inferred if Roy forgets to bring the list? Can it be assumed that he will buy what he should? Can it be assumed that he'll fail and buy only a few items that were on the list? If the description were more comprehensive, Roy's behavior would be more deterministic. A

more complete way of predicting Roy's behavior could be: IF Roy has the shopping list, THEN he will buy the correct items, ELSE he'll buy nothing. The use of IF-THEN-ELSE logic makes Roy's behavior predictable.

This concept applies more to hardware and software design than it does to people. Since people naturally use IF-THEN logic, they tend to apply it to their software and hardware designs. As seen in the example above, this approach leaves 'holes' in the logic, and therefore the subject is prone to behaving in an undeterminable fashion. This phenomenon is more readily recognized as the proverbial bug.

By learning IF-THEN-ELSE logic well and adopting the habit of always using it, the students will be conditioned to design their systems using airtight logic.

Structure

It is assumed that the student has no prior knowledge of electronics, circuit design, schematic reading, and software programming, but has a modest amount of knowledge in software and PC usage. Throughout the manual, each item of information is provided one at a time. In certain areas where the student may crave more background information, an appendix is provided.

This text was written with the Parallax Logic Simulator, Parallax Digital Trainer, and the BASIC Stamp 2 in mind. Additionally, it can greatly benefit the student's learning by viewing certain signals while working through this text. Though not required to complete the curriculum, Parallax recommends that the Parallax USB Oscilloscope or equivalent oscilloscope be used where the instructor deems advantageous.

Symbols

Throughout this manual, a few symbols are used to direct the student's attention to certain questions or problems. The following are the symbols and the description of their general usage.

Question: A question is placed before the student and answered. This is something to think about.

Information: A unit of information. The student needs to understand this concept before proceeding. It will be on the test.

Watch out! A caption in this box should be heeded for the safety of all circuits involved. This box is sometimes used to denote the difference between theory and reality and a number of other important concepts.

Getting Started

Bill of Materials

Please take a moment to confirm the presence of the items required along with this text to complete the lessons that follow.

1. Parallax Logic Simulator (PLS.zip). This will be on a diskette or a CD. If it is on the CD, just follow the instructions to find the install files section.
2. Parallax Digital Trainer. It is a large green circuit board with "Parallax Digital Trainer" printed on it.
3. Jumper wire. There should be three rolls of 22 gauge solid wire. Each roll should be a different color.
4. Wire strippers for stripping 22 gauge wire easily.

Optionally, the student may employ the following items to further enhance the learning experience:

1. Parallax USB Oscilloscope or equivalent oscilloscope.

Dos and Don'ts

- Do not wire the board while its "hot". Construct the circuits while the Parallax Logic Board is de-energized, (power not applied).
- Do not wire outputs to other outputs. Doing so may damage the hardware. Be certain that outputs are connected only to inputs. (Including the BASIC Stamp 2's I/O pins).
- Do be static sensitive. If unfamiliar with the proper procedures for handling static sensitive electronic devices, please review Appendix G.
- Do make sure that you understand each lesson before proceeding to the next. Each lesson is predicated upon the previous lessons.
- Remove metallic jewelry and watches before working with the PDT. Even though you will be working with low voltages, short circuits can

cause excessive heat sufficient to cause painful burns, as well as damage the ICs of the PDT.

- Do not troubleshoot while hot. If your circuit is not working properly, make a quick mental note of the status of the outputs and immediately power-down the PDT. Prolonged short-circuits can cause heat and damage.
- Do allow the imagination to ask "what if" - then wire it up and see what happens (after understanding the basics).
- Do have fun. Logic can be fascinating.

ACTIVITY #1: SIMULATOR SOFTWARE INSTALLATION

If the Parallax Logic Software was provided on a diskette, follow these steps:

1. Within the Program Files folder, create a new folder called, "Parallax Inc".
2. Within that folder, create a new folder called "Logic Simulator".
3. If you have a file called, "PLS.exe", copy it to the Logic Simulator folder.
4. If you have a file called, "PLS.zip", you will have to unzip it into the Logic Simulator folder.
5. Right click while the mouse pointer is over the PLS.exe icon.
6. Create a shortcut and drag it onto the desktop.

Alternately, if the Parallax Logic Simulator was provided to you on a Parallax CD, insert it into your CDROM drive and follow the directions to install new software. The program you are looking for is called, "Logic Simulator Setup.exe". The Parallax Logic Simulator will be used exclusively in chapters 1 through 4.

ACTIVITY# 2: BASIC STAMP SOFTWARE INSTALLATION

To install the BASIC Stamp Windows Editor Program:

1. Insert the Parallax CD
2. Select "Software", and follow the instructions to install the BASIC Stamp Windows Editor v2.0 (install the latest version).

The BASIC Stamp Editor will be used in chapter 9.

Chapter Review Questions

1. Is it OK to construct circuits on the PDT while it is powered-up?
2. Is it OK to connect an output to another output?
3. What can you do to minimize the likelihood of static electricity damage to the PDT?
4. Is it OK to skip a lesson here or there?
5. Is it OK to wear a bracelet while working with the PLS? Explain.
6. Is it OK to wear a necklace while working with the PDT? Explain.
7. What should you do if your circuit doesn't work as you think it should?

Chapter #2: Tutorial

ACTIVITIES IN THIS CHAPTER

1. Activity 1: Exploring DC Circuits
2. Activity 2: Active High Pushbutton
3. Activity 3: Active Low Pushbutton
4. Activity 4: The AND Function
5. Activity 5: The OR Function

By the end of this chapter, you should know:

1. Basic DC Circuit operation: resistor, LED, battery, etc.
2. Current, voltage, resistance, and other electrical terms
3. Schematic symbols: resistor, switch, pushbutton, ground, and battery
4. Switch and pushbutton operation: active-high, active-low
5. The logical AND function
6. The logical OR function

Before forging ahead, a few basic ideas need to be covered. While covering these basic ideas, new words will be defined and concepts explored. In this section we will cover simple DC circuits, basic switch and pushbutton operation, and the concept of TRUE and FALSE which is essentially the foundation of all digital binary systems.

SIMPLE DC CIRCUITS

ACTIVITY #1: EXPLORING DC CIRCUITS

Please start the Parallax Logic Simulator application. The screen on Figure 2-1, will be presented after the software has started. If the screen appears drastically different from the screen shot pictured below, please ask your instructor for assistance, or contact Parallax at (916) 624-8333 and ask for Technical Support.

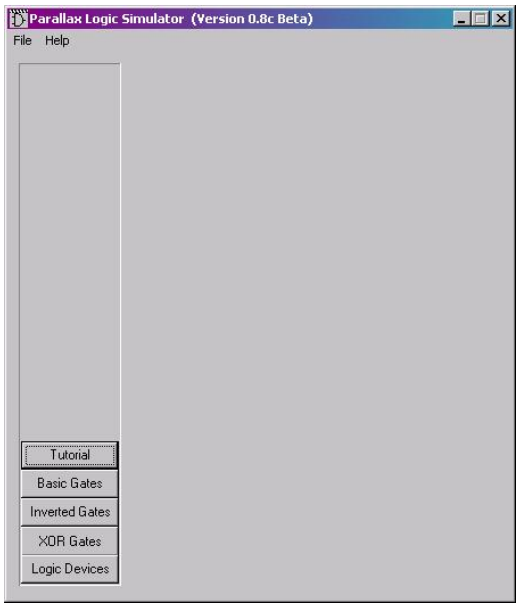

Figure 2-1
PLS - First screen.

Use the mouse to click on the "Tutorial" button. The "Tutorial" button on Figure 2-2, should slide to the top of the screen and reveal several menu icons.

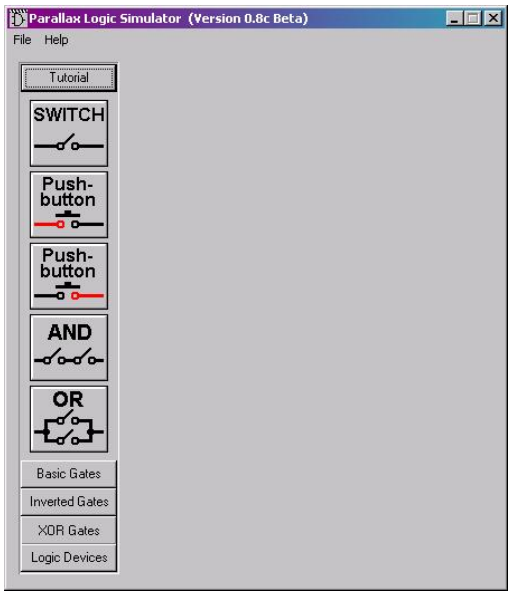

Figure 2-2
PLS – Tutorial section

Please click on the "Switch" menu icon showed on Figure 2-3. A simple DC circuit with a switch will appear in the work area. Some lines that make up the circuit on Figure 2-4 are red and some are black. Colors are used to signify the various voltages present in the circuit. Red signifies 5 Volts DC; black signifies that there is no voltage in the wire (0 Volts).

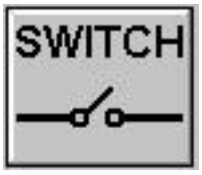

Figure 2-3
SWITCH

It is a device used to electrically connect or disconnect two points. The schematic symbol for a switch is made from two horizontal lines, two small circles, and a slanted line. This switch is depicted as open (off).

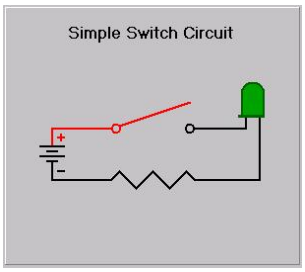

Figure 2-4
PLS – Switch circuit selected

If you move you mouse pointer over the various elements of the Simple Switch Circuit, hints will be displayed that show the name of the components therein. If you haven't any experience with electrical circuits, you'll need to become familiar with the following terms.

Electricity: a form of energy made up of positive and negative charges. Like charges repel each other, while dissimilar charges attract each other. A negative charge is made when there are more electrons than protons in a given area. Conversely, a positive charge is formed when there are fewer electrons than protons in a given area.

Voltage: the force that propels electrons, measured in Volts, from a negative charge towards a positive charge and is represented in equations by a capital "E" (for energy). When expressive the amount of Volts, the symbol used is a capital V. Water analogy: voltage is to electrons in a wire, as pressure is to water in a pipe.

Current: the rate of flow of electrons, measured in Amperes (Amps), and is represented in equations by a lower-case "i" (for intensity). When expressing the amount of Amps, the symbol used is a capital A. Water analogy: the rate of flow of electrons through a wire is, as the number of gallons per minute flowing through a pipe..

DC: Direct Current is current that flows in only one direction.

Resistance: is that property which opposes the flow of electrical current. It is represented in equations by a capital "R". When expressing an amount of resistance, the symbol used is the omega character, Ω. Water analogy: A resistor to a circuit is like a dam to a river.

Figure 2-5
RESISTOR

It is a device that has resistance and therefore opposes the flow of current. Resistors come in a wide variety of resistance values and are frequently used to limit the flow of current. The schematic symbol for a resistor is comprised of several slanted lines and two horizontal lines, (the legs).

The Battery is the heart of the circuit; it is essentially a pump that delivers electrons. The schematic symbol for a battery is made up of alternately long and short horizontal lines. These lines symbolize the cells that make up a typical lead-acid battery. Note that the end of the battery with the shorter horizontal line is the negative lead. The end of the battery with the longer horizontal line is the positive lead.

LED: Light Emitting Diode. Basically, it is an electronic version of a flashlight bulb. Please visit Appendix A for more information on LEDs.

The Switch either makes (connects) or breaks (disconnects) the circuit. When the switch is closed, the circuit is complete and will allow current to flow through the LED and it will light. Since this is very much like the way every light switch in your home works, you can see that when the switch is open, the circuit is broken, no current flows, and the LED goes out.

Position the mouse pointer over the Switch and click once. The switch obediently closes thereby closing the circuit, current flows through the resistor and LED, and the LED lights.

Note: If the above circuit were actually constructed, you would see that each component has legs that protrude from its body. Also, you would see the connection points where two legs of different components are connected. In our simulator software, we are depicting this circuit in schematic format. In schematic format, such connection points are not visible except for when three or more points are joined. The exception here to the schematic format is the LED. A picture of an LED was used instead of the actual schematic symbol because it can easily be made to “glow” as if it were energized. These conventions were utilized both to clarify schematics and to help convey the behavior of the circuits.

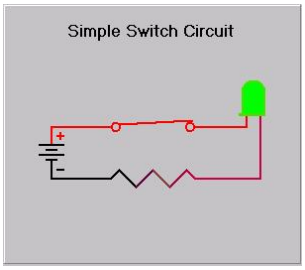

Figure 2-6
PLS – Switch closed

In Figure 2-6 note the color of the wires. Since color denotes the relative voltage in the wires and components, you can see that 5 Volts DC is present on the positive side of the battery, across the entire switch, and on the positive lead of the LED. The color of the wire dims in intensity on the negative lead of the LED. This is because the LED requires 1.4VDC to light. So, the voltage on the negative lead is 5V – 1.4V = 3.6V. 3.6V is present on the positive side of the resistor as well.

The resistor's function in this circuit is to limit the amount of current that flows to a safe level. If the negative side of the LED were connected directly to the negative lead of the battery, instead of through a current limit resistor, the LED would immediately burn out. If you wish to learn more about this, please refer to Appendix A, DC Circuit theory.

REFERENCE: the point we regard when measuring something. In electronics, we usually regard ground (the negative lead of the battery) when measuring the voltage of the circuit.

It is important to note that whenever we measure something, we do so with regard to a reference. Example: Q: How high is the airplane? A: It is at 5,000 feet. Now that is an incomplete and dangerous answer. Why? Well it will make a big difference if that 5,000 feet is with regard to sea level or the ground. How so? Well assume you are flying an airplane and you must fly over a mountain that rises 5,000 feet high over the valley floor. After take-off, you climb until your altimeter reads 8,000 feet and head towards the mountain. All is fine until you hit the mountain - literally HIT the mountain. Where did you go wrong? Its all in the reference: the mountain may rise 5,000 feet over the valley floor, but how high is it with regard to sea level? The altimeter in the cockpit indicates its altitude with regard to sea level. Get the point? If not, don’t become a pilot. Seriously, let's get back to the lesson.

Since, in electronics, we measure all voltages with respect to ground, ground (the negative lead of the battery) is, by definition, 0 VDC. In the simulator, when wires or components are de-energized, they are depicted in the color black. The color in the resistor changes proportionately from red to black, indicating that it is dissipating the voltage eventually to 0 VDC.

Of course, wires and components in real life do not change their color when they are energized (except in extreme cases not covered here). The color scheme employed in the simulator software was used to imbue a sense of how voltage behaves.

Go ahead and exercise the switch a few times to get the idea of how the circuit functions. Note that clicking on the Switch menu icon simply resets the switch to its off state.

Exercises

1. What is the relationship between the state of the switch and the state of the LED?
2. Complete the table below to depict this relationship.

SWITCH	LED
ON	?
OFF	?

ACTIVITY #2: ACTIVE HIGH PUSHBUTTON

Pushbuttons are everywhere. If you don't believe it, try to go a day without using one. And, as long as mankind has at least one usable finger, pushbuttons will continue to exist. Since the pushbutton has become mankind's favorite input device, we'll study it first.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Tutorial button, and then choose the Active-high Pushbutton showed in Figure 2-7. Here is what the Active-High Pushbutton menu button looks like.

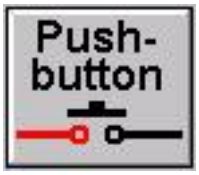

Figure 2-7
PLS – PUSHBUTTON

It is a device used to electrically connect or disconnect two points. The schematic symbol for a pushbutton is made from two horizontal lines, two small circles, and another horizontal line with a smaller line atop in the middle. The left lead is red - this denotes that this pushbutton is the Active-High Pushbutton.

This should be the screen that appears:

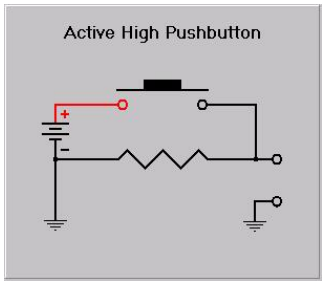

Figure 2-8
PLS –Active-high Pushbutton

There are two new schematic symbols in the circuit above. They are called ground, Figure 2-9, and the output, Figure 2-10. All symbols and wires that connect to the ground symbol are connected together. The Figure 2-9 shows a typical ground symbol.

Figure 2-9
GROUND

As mentioned before, ground is the reference from which we measure voltage and is essentially the negative lead of the battery. The ground symbol is made from one vertical line and several horizontal lines that form a triangle.

The ground symbol is a form of schematic short-hand. Normally, any device's connection to the negative side of the battery would be depicted with a connecting line. In large schematics, this can clutter the drawing and add a lot of confusion. One way to keep the schematic clean and clear is to use the schematic symbol of ground instead of showing a wire. Mentally, we know that all points in the schematic with the ground symbol attached are actually connected to the negative side of the battery.

Another difference you may have noticed is the absence of the LED. The LED was instrumental in depicting the general relation of the switch and its output, but now we're ready to depict our circuits using a more professional style.

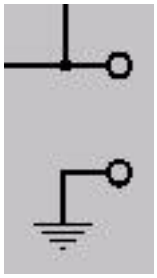

Figure 2-10
OUTPUT TERMINALS

A connection point for an output device. Instead of showing the actual output device, it will be better for us from now on to depict an abstraction of an output device. The symbol to the left represents the connection point of an output device. The output terminal symbol consists of two circles: one connected to ground, while the other is the active output. Although the output terminal connected to the ground appears to be something completely separate from the circuit, it, plus the segment of the output terminal symbol directly above it, constitute the entire output terminal circuit symbol. It is depicted this way to show that the output is referenced to ground.

One other item you may have noticed is that the hint describes the resistor as a PULL-DOWN resistor. A pull-down resistor is just a regular resistor used to pull the voltage on a wire down towards 0 VDC. If there is something forcing a non-zero voltage on that wire, the pull-down resistor will allow that non-zero voltage to exist. As soon as the device that was forcing the non-zero voltage on that wire ceases to force the voltage, the pull-down resistor drains the voltage on that wire to 0 VDC. This sounds complicated, but watching it work in the simulator should help clarify it.

Click on the pushbutton to change its state back and forth between ON and OFF. Note that the switch maintains the position that you leave it in. This type of pushbutton is called push-push.

As you move the mouse pointer over the various components, the hints will tell you the name of each component. Exercise the pushbutton, pay attention to the state of the pushbutton and how its state relates to that of the output.

Exercises

1. What is the relationship between the state of the pushbutton and the state of the output?
2. Construct a table, similar to the one in Chapter 2, Activity #1, that depicts this relationship.

ACTIVITY #3: ACTIVE LOW PUSHBUTTON

For our purposes, there are two ways a pushbutton can be used. In Activity #2 we explored the Active-High Pushbutton. This section explores the Active-Low pushbutton, from Figure 2-12. Both methods have their advantages and are widely used, so we must learn, study, and fully grasp each style.
If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Tutorial button, and then choose the Active-Low Pushbutton. The Figure 2-11 shows how the Active-Low Pushbutton menu button looks like.

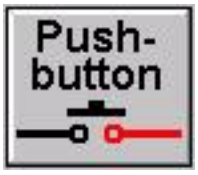

Figure 2-11
PLS – PUSHBUTTON

It is a device used to electrically connect or disconnect two points. The schematic symbol for a pushbutton is made from two horizontal lines, two small circles, and another horizontal line with a smaller line atop in the middle. The right lead is red - this denotes that this pushbutton is the Active-Low Pushbutton.

After you click the button, the screen on Figure 2-12 should appear:

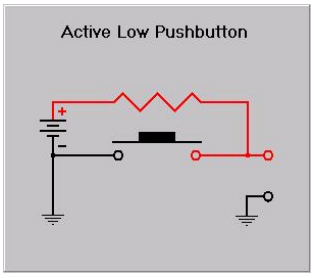

Figure 2-12
PLS – Active-Low Pushbutton

Though the circuit contains the exact same components as the previous circuit, it is configured differently. You should note that the output is on, yet the pushbutton has not been pushed yet. The Active-Low Pushbutton yields the opposite output of the Active-High Pushbutton circuit.

Exercises

1. What is the difference between the Active-High Pushbutton configuration and the Active-Low pushbutton configuration?
2. How do you suppose each style of pushbutton got its name? Please speculate.

ACTIVITY #4: THE "AND" FUNCTION

Problems can be complex or simple. By definition, simple problems are easy to solve. Solving a complex problem is merely an issue of breaking it down into small bite-size functions. If any of these functions are still too complex, simply continue to break down those functions until all you have is a collection of bite-size functions. The first bite-size function to examine is the AND function.

FUNCTION: a relation between the elements of two sets in which each element in one set corresponds to exactly one element in the other set.

By now we are quite familiar with our simple switch or pushbutton circuit in one form or another. Now consider the circuit's behavior if we use two switches in series with each other.

SERIES: orientation of two or more objects such that they are in-line, (next to one another). Basically, what flows through one must flow through the other, in order to be a series circuit.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Tutorial button, then choose the AND function button, showed on Figure 2-13. Here is what the AND function menu button looks like.

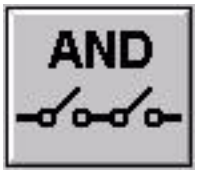

Figure 2-13
PLS – AND Function menu button

A function involving two inputs in which both inputs must be 'TRUE' to yield a 'TRUE' output, otherwise the output is 'FALSE'.

Figure 2-14 shows the window that will appear on your screen.

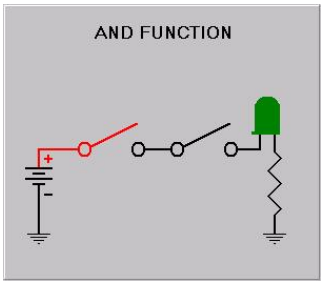

Figure 2-14
PLS –AND Function

You may have noticed that the hint given when the mouse is positioned over the LED or resistor has changed to, "Output circuit". Since the LED and resistor together form the output circuit, they may be referred to as the Output circuit from here on.

Use the mouse and exercise the switches until you become very familiar with the relationship of the switches (their states) to the state of the output. The essence of the AND function is this: both switches must be in the closed position for the LED to light, otherwise, the LED will not be on. Another way to express this statement is:

`IF` *Switch A* `AND` *Switch B are both ON*,
`THEN` *the LED will be ON*,
`ELSE` *the LED will be OFF*.

Exercises

1. How many different combinations of states can be made with two switches?
2. Construct a table that describes the behavior of the AND function.

ACTIVITY #5: THE "OR" FUNCTION

The most sophisticated computers in the world can be constructed from just three types of functions: the AND, OR, and NOT functions. It seems reasonable that, if we master these three functions, we can then build anything we set our minds to.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Tutorial button, and then choose the OR function. The Figure 2-15 shows how the OR Function menu button looks like.

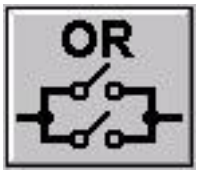

Figure 2-15
PLS – OR Function menu button

A function involving two inputs in which both inputs must be 'FALSE' to yield a 'FALSE' output, otherwise the output is 'TRUE'.

Figure 2-16 should be the screen that appears.

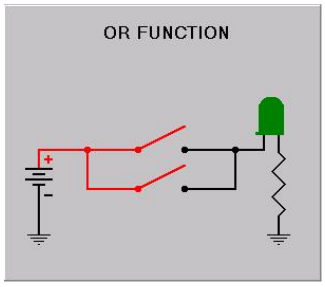

Figure 2-16
PLS – OR Function

The OR function is comprised of two switches (inputs) connected in parallel with each other. Use the mouse and exercise the switches until you become very familiar with the relationship of the switches (their states) to the state of the output.

The essence of the OR function is this: both switches must be in the open position for the LED to remain off, otherwise, the LED will light. Another way to express this statement is:

`IF` *either Switch A* `OR` *Switch B are ON*,
`THEN` *the LED will be ON*,
`ELSE` *the LED will be OFF*.

Exercises

1. How many different combinations can be made with two switches in this configuration?
2. Construct a table depicting these possible input combinations and the resulting output.
3. List five real-world problems/applications that either the AND or the OR functions could solve.

Chapter Review Questions

1. What is electricity?
2. What is the flow of electricity called?
3. What is the force that propels electricity called?
4. What is the opposition of this flow referred to as?
5. What is the function of a battery?
6. What is the function of a resistor?
7. What is the function of a switch?
8. Describe the behavior of the AND function in your own words.
9. Describe the behavior of the OR function in your own words.

Chapter #3 Exploring Gates with the Simulator

ACTIVITIES IN THIS CHAPTER

1. Activity 1: The "AND" Gate
2. Activity 2: The "OR" Gate
3. Activity 3: The "NOT" Gate
4. Activity 4: The "NAND" Gate
5. Activity 5: The "NOR" Gate
6. Activity 6: The "XOR" Gate
7. Activity 7: The "XNOR" Gate

By the end of this chapter, you should know:

1. Schematic symbols for each gate
2. Operation and truth table for each gate.

Congratulations! You've made it through the tutorial and are ready for your first foray into the fascinating world of logic.

BASIC GATES

There are many kinds of gates used to implement logical functions. All of them can be constructed from just three kinds: the AND Gate, the OR Gate, and the NOT Gate. Chapter 3 explores first the Basic Gates, then the Inverted Gates, and lastly the Exclusive OR Gates. Each activity will introduce a new logic device and possibly one or two new logic concepts as well.

ACTIVITY #1: THE "AND" GATE

This activity explores the AND Gate and introduces a new circuit element used in the simulator software; specifically, the new symbol for a pushbutton.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Basic Gates button, then choose the AND Gate. Figure 3-1 shows how the AND Gate menu button looks like:

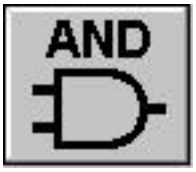

Figure 3-1
PLS – AND Gate menu button

A function involving two inputs in which both inputs must be 'TRUE' to yield a 'TRUE' output, otherwise the output is 'FALSE'.

The screen that appears should be like Figure 3-2.

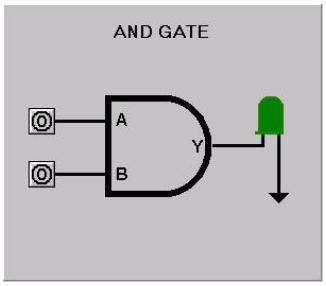

Figure 3-2
PLS – AND Gate along with an output LED and two pushbutton inputs.

Please consider Figure 3-2 while reading the following. While you may have recognized the LED, you're probably wondering where the resistor went. The resistor is not depicted because it itself is not necessary to understand how the AND gate functions. In reality, a current limit resistor is always necessary to limit current to a safe level for the LED. These simplifications are made for the sake of clarity.

The large item in the middle of Figure 3-2 is the schematic symbol for the AND Gate. When speaking of gates, we generally refer to the inputs as A and B and the output as Y. Two symbols for pushbuttons serve as inputs instead of switches.

Figure 3-3
PLS – Pushbutton Symbol: Active-High - Off State - Not Pushed.

Figure 3-4
PLS – Pushbutton Symbol: Active-High - On State - Pushed.

Please note the shading at the perimeter of the new symbols: it is meant to make the button look three dimensional. Figure 3-3 is meant to look like it is raised up while Figure 3-4 is meant to look like it is pushed into the paper. Also, note that the character in the center of the image changes from a '0' to a '1'. The character in the center indicates the state of the output of the pushbutton. A '1' output state means that the pushbutton is outputting 5 Volts DC, while a '0' output state means that the pushbutton is outputting ground (0 Volts DC).

The last new item to cover before proceeding is called the Truth Table, as shown in Figure 3-5. The Truth Table describes the state of the output for any given state of the input.

Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

Figure 3-5
PLS – Truth table for the AND Gate.

Use the mouse and exercise the pushbuttons until you become comfortable and re-acquainted with the relationship of the pushbuttons (their states) to the state of the output. As you exercise the pushbuttons, you should note that the color of the wire changes (just as in previous examples) to help convey the state of the gate's input. Does the truth table look familiar? This should closely resemble the table you constructed in Activity #4 of Chapter 2.

Exercises

1. Confirm the proper operation of the Parallax software simulator by comparing its output to that predicted by the truth table for all possible combinations of inputs.

ACTIVITY #2: THE "OR" GATE

This activity explores the OR Gate and introduces the OR Gate schematic symbol.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Basic Gates button, and then choose the OR Gate. Figure 3-6 shows how the OR Gate menu button looks like:

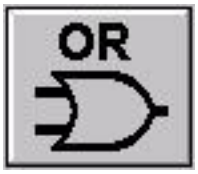

Figure 3-6
PLS – OR Gate menu button

A function involving two inputs in which both inputs must be 'FALSE' to yield a 'FALSE' output, otherwise the output is 'TRUE'.

Figure 3-7 should be the screen that appears.

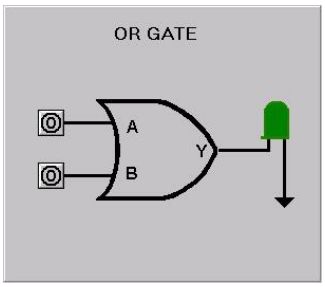

Figure 3-7
PLS – OR Gate along with an output LED and two pushbutton inputs.

The large item in the middle of Figure 3-7 is the schematic symbol for the OR Gate. Just like the AND Gate, the OR Gate has two inputs, A and B, and one output, Y. Similarly, two symbols for pushbuttons serve as inputs instead of switches.

Figure 3-8 shows the truth table that describes the behavior of the OR Gate, ().

Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

Figure 3-8
PLS – Truth table for the OR Gate.

Use the mouse and exercise the pushbuttons until you become comfortable and re-acquainted with the relationship of the pushbuttons (their states) to the state of the output. Does the truth table look familiar? This should closely resemble the table you constructed in Activity #5 of Chapter 2.

Exercises

1. Confirm the proper operation of the Parallax software simulator by comparing its output to that predicted by the truth table for all possible combinations of inputs.

ACTIVITY #3: THE "NOT" GATE

This activity explores the NOT Gate and introduces the NOT Gate schematic symbol.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Basic Gates button, then choose the NOT Gate. Figure 3-9 shows how the NOT Gate menu button looks like:

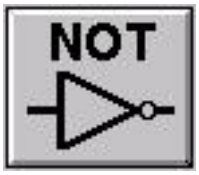

Figure 3-9
PLS – NOT Gate menu button

A function involving one input and one output. The output is always the opposite state of the input.

Figure 3-10 should be the screen that appears.

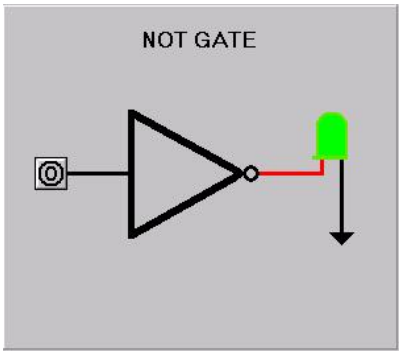

Figure 3-10
PLS – NOT Gate along with an output LED and pushbutton input.

The large item in the middle of Figure 3-10 is the schematic symbol for the NOT Gate. Unlike the previous gates, the NOT Gate has only one input: A. Like the other gates, the NOT Gate still has only one output: Y. One pushbutton serves as the input instead of a switch.

At this point it is appropriate to explain the small circle present at the end of the NAND, NOR, and NOT gates. The circle itself implies an inversion of the signal. To invert any binary digital signal is to change its state to the opposite of what it currently is. In fact, if this NOT Gate was missing its small circle, it would not invert the signal and would be called a YES Gate or a PASS Gate instead of a NOT Gate.

Figure 3-11 is the truth table that describes the behavior of the NOT Gate.

Input A		Output Y
0		1
1		0

Figure 3-11
PLS – Truth table for the NOT Gate.

Use the mouse and exercise the pushbutton until you become comfortable and well acquainted with the relationship of the pushbutton's state to the state of the output.

Exercises

1. Confirm the proper operation of the Parallax software simulator by comparing its output to that predicted by the truth table for both possible states of the input.

ACTIVITY #4: THE "NAND" GATE

This activity explores the NAND Gate and introduces the NAND Gate schematic symbol.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Inverted Gates button, then choose the NAND Gate. Figure 3-12 shows how the NAND Gate menu button looks like:

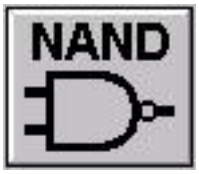

Figure 3-12
PLS – NAND Gate menu button

A function involving two inputs in which both inputs must be 'TRUE' to yield a 'FALSE' output, otherwise the output is 'TRUE'. The name NAND is a contraction for NOT-AND.

Figure 3-13 should be the screen that appears.

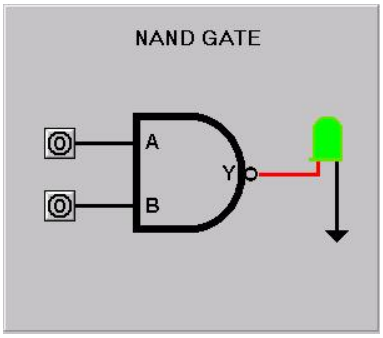

Figure 3-13
PLS – NAND Gate along with an output LED and two pushbutton inputs.

The large item in the middle of Figure 3-13 is the schematic symbol for the NAND Gate. Just like the AND Gate, the NAND Gate has two inputs, A and B, and one output, Y. Similarly, two symbolic pushbuttons serve as inputs instead of switches.

Figure 3-14 is the truth table that describes the behavior of the NAND Gate.

Input A	Input B	Output Y
0	0	1
0	1	1
1	0	1
1	1	0

Figure 3-14
PLS – Truth table for the NAND Gate.

Use the mouse and exercise the pushbuttons until you become comfortable and well acquainted with the relationship of the pushbuttons (their states) to the state of the output.

Exercises

1. Confirm the proper operation of the Parallax software simulator by comparing its output to that predicted by the truth table for all possible combinations of inputs.

ACTIVITY #1: THE "NOR" GATE

This activity explores the NOR Gate and introduces the NOR Gate schematic symbol.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Inverted Gates button, then choose the NOR Gate. Figure 3-15 shows how the NOR Gate menu button looks like:

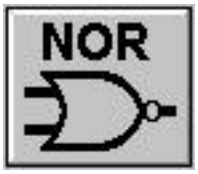

Figure 3-15
PLS – NOR Gate menu button

A function involving two inputs in which both inputs must be 'FALSE' to yield a 'TRUE' output, otherwise the output is 'FALSE'. The name NOR is a contraction for Not-OR.

Figure 3-16 should be the screen that appears.

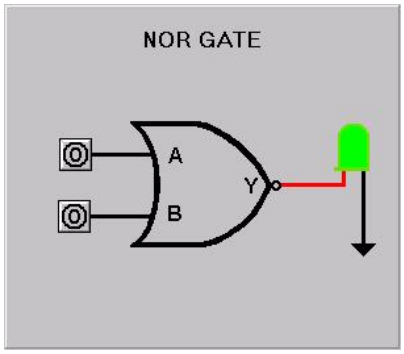

Figure 3-16
PLS – NOR Gate along with an output LED and two pushbutton inputs.

The large item in the middle of Figure 3-16 is the schematic symbol for the NOR Gate. Just like the NAND Gate, the NOR Gate has two inputs, A and B, and one output, Y. Similarly, two symbolic pushbuttons serve as inputs instead of switches.

Figure 3-17 is the truth table that describes the behavior of the NOR Gate.

Input A	Input B	Output Y
0	0	1
0	1	0
1	0	0
1	1	0

Figure 3-17
PLS – Truth table for the NOR Gate.

Use the mouse and exercise the pushbuttons until you become comfortable and well acquainted with the relationship of the pushbuttons (their states) to the state of the output.

Exercises

1. Confirm the proper operation of the Parallax software simulator by comparing its output to that predicted by the truth table for all possible combinations of inputs.

ACTIVITY #2: THE "XOR" GATE

This activity explores the XOR Gate and introduces the XOR Gate schematic symbol.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the XOR Gates button, then choose the XOR Gate. Figure 3-18, shows how the XOR Gate menu button looks like:

Figure 3-18
PLS – XOR Gate menu button

A function involving two inputs and one output. Output is TRUE only when the inputs are at different states from each other.

Figure 3-19 should be the screen that appears.

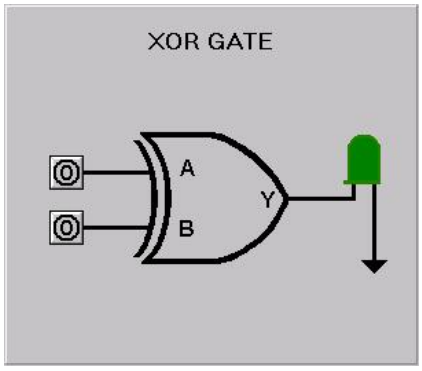

Figure 3-19
PLS – XOR Gate along with an output LED and pushbutton inputs.

The large item in the middle of Figure 3-19 is the schematic symbol for the XOR Gate. Like several other gates, the XOR Gate has two inputs, A and B, and just one output. Two pushbuttons serve as inputs.

Figure 3-20 is the truth table that describes the behavior of the XOR Gate.

Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	0

Figure 3-20
PLS – Truth table for the XOR Gate.

Use the mouse and exercise the input pushbuttons until you become comfortable and well acquainted with the relationship of the pushbuttons state's to the state of the output.

Exercises

1. Confirm the proper operation of the Parallax software simulator by comparing its output to that predicted by the truth table for all possible combinations of inputs.

ACTIVITY #3: THE "XNOR" GATE

This activity explores the XNOR Gate and introduces the XNOR Gate schematic symbol.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the XOR Gates button, then choose the XNOR Gate. Figure 3-21 shows how the XNOR Gate menu button looks like:

Figure 3-21
PLS – XNOR Gate menu button

A function involving two inputs and one output. Output is TRUE only when the two inputs are at the same state.

Figure 3-22 should be the screen that appears.

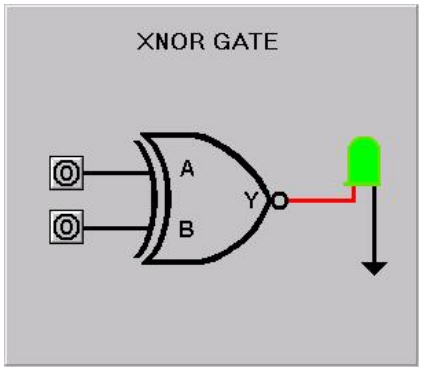

Figure 3-22
PLS – XNOR Gate along with an output LED and pushbutton inputs.

The large item in the middle of Figure 3-22 is the schematic symbol for the XNOR Gate. Like several other gates, the XNOR Gate has two inputs, A and B, and just one output. Two pushbuttons serve as inputs.

Figure 3-23 is the truth table that describes the behavior of the XNOR Gate.

Input A	Input B	Output Y
0	0	1
0	1	0
1	0	0
1	1	1

Figure 3-23
PLS – Truth table for the XNOR Gate.

Use the mouse and exercise the input pushbuttons until you become comfortable and well acquainted with the relationship of the pushbuttons state's to the state of the output.

Exercises

1. Confirm the proper operation of the Parallax software simulator by comparing its output to that predicted by the truth table for all possible combinations of inputs.

Summary

All logic circuits are comprised of logic gates at an elemental level. Chiefly, these gates are: AND, OR, NOT, NAND, NOR, XOR, and XNOR Gates. A working knowledge of these gates will help you with many aspects of computer and electronics engineering.

Truth tables can be used as references when designing or troubleshooting logic circuits. However, it would be best if you committed the functionality of each common type of gate to memory. If you are having difficulty remembering each gate's functionality, you may try this other way to view the basic gates.

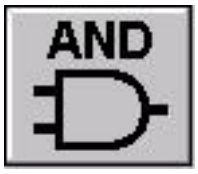

Figure 3-24
AND Gate

Think of the AND function as the ALL function:
If ALL of the inputs are high, the output will be high.

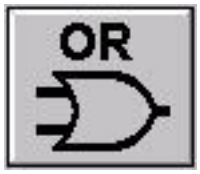

Figure 3-25
OR Gate

Think of the OR function as the ANY function:
If ANY of the inputs are high, the output will be high.

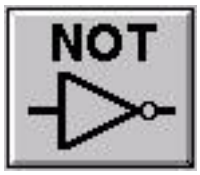

Figure 3-26
Not Gate

The circle implies an inversion. If you do not see a circle, think high.
If you see a circle, think low.

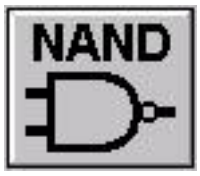

Figure 3-27
NAND Gate

Since a NAND Gate is a AND Gate and a NOT Gate put together, you can think of it as: ALL highs in, is a low output.

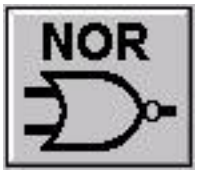

Figure 3-28
NOR Gate

Since a NOR Gate is an OR Gate and a NOT Gate put together, you can think of it as: ANY high in, is a low output.

Figure 3-29
XOR Gate

The XOR Gate yields a true output when the inputs are different.
Think of this gate as the difference gate:
If the inputs are different, the output is high.

Figure 3-30
XNOR Gate

The XNOR Gate yields a true output when the inputs are the same.
Think of this gate as the same gate:
If the inputs are the same, the output is high.

Using this way of thinking of gates enables you to derive the truth table by remembering **one statement** for each gate instead of entire truth tables.

Chapter Review Questions

1. Draw the schematic symbols for the Basic Gates.
2. From memory, derive the truth table for each Basic Gate.
3. Draw the schematic symbols for the Inverted Gates.
4. From memory, derive the truth table for each Inverted Gate.
5. Draw the schematic symbols for the XOR Gates.
6. From memory, derive the truth table for each XOR Gate.

Chapter #4 Logic Devices

ACTIVITIES IN THIS CHAPTER

1. Activity 1: The Multiplexer
2. Activity 2: The Demultiplexer
3. Activity 3: The Counter
4. Activity 4: The Flip-Flop

By the end of this chapter, you should know:

1. the difference between static and transitional inputs.
2. the nature of an active-high transitional input.
3. the nature of an active-low transitional input.
4. what a multiplexer, demultiplexer, counter and flip-flop are, and their basic operations.

Now that you've made it through the fundamental lessons, four interesting logic devices will be previewed. Early exposure to these circuits is useful in teaching the various kinds of inputs that various logic devices require. In this chapter, only the basics of these logic devices will be covered; they will be explored in depth in subsequent chapters.

LOGICAL INPUTS

Until now you have been exposed to two types of input switches - active-high and active low, and have only used one type - the active-high switch. The more interesting logic devices require a few different kinds of inputs. This chapter will, activity by activity, introduce a new logic device as well as a new type of logic input. As each type of input is explored, the nature of that input will be defined and explained.

ACTIVITY #1: THE MULTIPLEXER

A multiplexer is a device that selects and connects one of many possible inputs to its output. Think of a multiplexer as a selector switch. Frequently, multiplexers are used to "steer" logic signals within a design. The nickname for a multiplexer is a MUX.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Logic Devices button, and then choose the MUX. Figure 4-1 shows how the MUX menu button looks like:

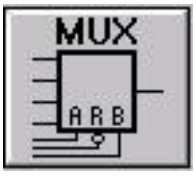

Figure 4-1
PLS – MUX

Short for multiplexer: think of it as an input selector.

Figure 4-2 should be the screen that appears.

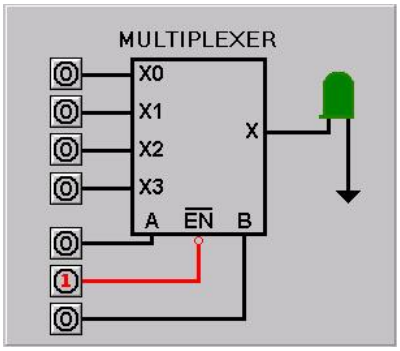

Figure 4-2
PLS – Multiplexer.

The sheer number of inputs makes this device look rather intimidating at first glance. Fear not; all of the inputs are easily explained, understood, and assimilated.

First consider the output X connected to the LED. That part of the above image is very similar to the output of the gates you've seen thus far. Now turn your attention to the inputs called: X0, X1, X2, and X3. These are the "possible" inputs. Any time this device is enabled, exactly one of these inputs is connected to the output X. Which input is

connected is determined by two other inputs: A and B. The most novel input in Figure 4-2 is the EN input. The EN input is an active-low input; that is why there is a bar over the EN. Another clue as to what kind of input EN is, it is the small circle connected to the input line. An active-low input implies that a '0' (0 Volts DC) must be present to enable the device; otherwise, the device is disabled. Another name for an active-low input is a low-true input. So, true is low (0 Volts) for the EN input. Since the EN line is an active-low, an active-low pushbutton is used.

The pictures and text below describe an active-low pushbutton.

Figure 4-3
PLS – Pushbutton Symbol: Active-Low - Off State - Not Pushed.

Figure 4-4
PLS – Pushbutton Symbol: Active-Low - On State - Pushed.

Note that the character in the center of the image changes from a '1' in the not-pushed state, to a '0' in the pushed state. The character in the center indicates the state of the output of the pushbutton. A '1' output state means that the pushbutton is asserting 5 Volts DC, while a '0' output state means that the pushbutton is asserting ground (0 Volts DC).

Figure 4-5 depicts the behavior of the multiplexer for all possible inputs.

Control Inputs			Connects
Enable	B	A	output X to:
1	X	X	NONE
0	0	0	X0
0	0	1	X1
0	1	0	X2
0	1	1	X3

Figure 4-5
PLS – MUX truth table

First note the state of the enable input as it relates to the connection of the X output. Figure 4-5 shows that if the enable input (active-low input) is high, the X output is not connected to any of the inputs. Conversely, when the enable input is low, then, based on the states of inputs A and B, one of the inputs will be connected to output X. The two Xs immediately below A and B in Figure 4-5 are special characters that have nothing to do with output X; they mean "don't care". So, when the EN is high, we "don't care" about the state of the A and B inputs since they have no effect.

Once that is understood, you may then focus on the meaning of the A and B inputs. Figure 4-5 shows when A and B are both '0', and the device is enabled, ('0' on EN), that input X0 will be connected to output X. This doesn't mean that the output will automatically switch on; it means that whatever state that input X0 is at will be echoed to output X. Similarly, when both A and B are '1' and the device is enabled, input X3 will be 'connected' to output X. These novel notions may remain a little foggy until you see how it works within the simulator environment.

Use the mouse and exercise the pushbuttons until you become comfortable and acquainted with the functionality of the multiplexer. Perform the exercise below:

Exercises

1. Confirm the proper operation of the Parallax software simulator by comparing its output to that predicted by the truth table for all possible combinations of inputs.

ACTIVITY #2: THE DEMULTIPLEXER

A demultiplexer is a device that steers an input to one of many possible outputs. Like the multiplexer, you can think of a multiplexer as a selector. Similarly, demultiplexers are used to 'steer' logic signals within a design. The nickname of the demultiplexer is the DEMUX. MUXs and DEMUXs are so similar that it makes sense to view the short descriptions of each side-by-side.

MUX: Multiplexer: steers/connects/selects one of several inputs to the output.

DEMUX: Demultiplexer: steers/connects/selects the input to one of several outputs.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Logic Devices button, then choose the DEMUX menu button. Figure 4-6 show how the DEMUX menu button looks like:

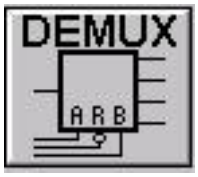

Figure 4-6
PLS – DEMUX

Demultiplexer: think of it as an output selector.

Figure 4-7 should be the screen that appears.

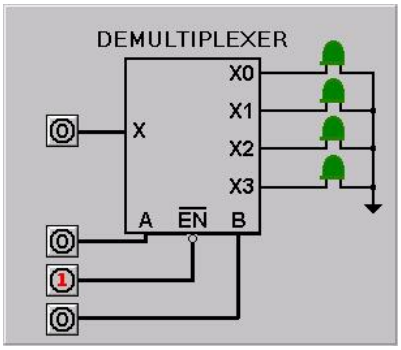

Figure 4-7
PLS – Demultiplexer

This device has exactly the same number of connections as the multiplexer. The inputs and the outputs are swapped (in a manner of speaking). Instead of connecting one of many inputs to the output as in the multiplexer, the demultiplexer connects the input to one of many outputs.

The A, B, and EN inputs function just as they did in the multiplexer. Figure 4-8 depicts the behavior of the demultiplexer for all possible inputs.

Control Inputs			Connects
Enable	B	A	input X to:
1	X	X	NONE
0	0	0	X0
0	0	1	X1
0	1	0	X2
0	1	1	X3

Figure 4-8
PLS – DEMUX truth table

Use the mouse and exercise the pushbuttons until you become comfortable and acquainted with the functionality of the demultiplexer. Perform the exercise below:

Exercises

1. Confirm the proper operation of the Parallax software simulator by comparing its output to that predicted by the truth table for all possible combinations of inputs.

Static Vs Transitional Logic

The logic gates covered thus far rely on "static inputs". A "static input" refers not to an input that does not change, but to the input before it changes or *once it has* changed. The distinction is made because, in logic, values between '0' and '1' are not defined nor are they discussed therefore we do not care about the state of the input *as* it changes. Often it is useful to consider *when* an input changes states, so some devices have transitional inputs that "trigger" when an input changes state. ('Trigger' is electronic term referring to the start of some event.) A transitional input is one which triggers (performs its function) *when* the input changes state. Transitional events are usually defined by a single type of transition. Some transitional inputs are triggered when the input goes from low to high, some are triggered when the input goes from high to low. The next logic device to be examined uses a transitional input.

STATIC Input: An input whose state (voltage level) is regarded only when it is not changing.

ACTIVITY #3: THE COUNTER

A counter is a device that counts in binary from 0000 to 1111 (0 to 15 in decimal). Each time it is triggered, the outputs are incrementally activated. For more information on binary numbers, the student may refer to Appendix D.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Logic Devices button, and then choose the Counter menu button. Figure 4-9 shows how the Counter menu button looks like:

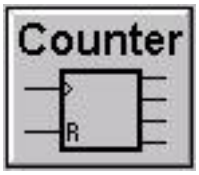

Figure 4-9
PLS – Counter

Incrementally activates the outputs (in a binary fashion) each time the clock input transitions from high to low.

Figure 4-10 should be the screen that appears.

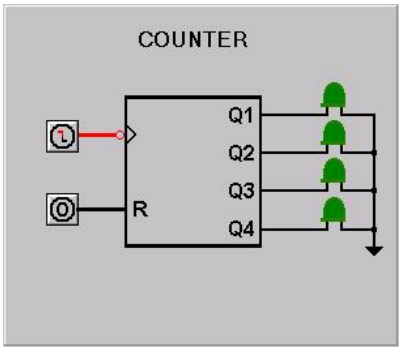

Figure 4-10
PLS - Counter

The Counter has two inputs. One of the inputs is the Reset input and is designated with 'R'. Whenever the reset line is high ('1'), all four outputs are switched off and the input is ignored. The other input is called the clock input and has a unique symbol ">".

The clock input requires a transitional input. Specifically, it requires the input to transition from '1' to '0' to trigger. This kind of transitional input is referred to as an off-transitional input.

Transitional Input: An input whose state is relevant only when it changes.

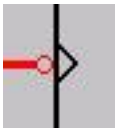

Figure 4-11
PLS – Active-Low Transitional input symbol

The horizontal line is the input lead, the circle indicates that this is an active-low input, the ">" means that this input is transitional, the vertical line is the perimeter of the device schematic symbol.

This counter has four outputs: Q1, Q2, Q3, and Q4. Each output is connected to an LED so that we can observe their states. An active-low transitional pushbutton is connected to the Clock input. An active-high pushbutton is connected to the Reset input. If the Reset is low, '0', the device is enabled and will 'count' each high to low transition on the clock input. This behavior is depicted in the Counter's truth table on Figure 4-12.

Inputs		Outputs
CLOCK	RESET	
X	1	ALL 0
↓	0	Advance to the next state

Figure 4-12
PLS – Counter truth table

The standard pushbutton is sufficient to trigger any circuit requiring a transitional input. After all, toggling the pushbutton from ON to OFF would create a transition. But, to familiarize you with and emphasize the nature and behavior of transitional inputs, a transitional pushbutton showed on Figure 4-13 will now be introduced.

Figure 4-13
PLS – Off-Transitional Pushbutton

Transitions from '1' to '0' when pushed, then returns to the '1' state. Note that the symbol within the circle has two colors: red(gray) and black. The red is on top and is meant to signify that, when you push this button, its output will change from '1' (red) to a '0' (black). Even though this button's output will return to a '1' after it is pushed, the symbol depicts the transition from high to low to emphasize that this is the transition that causes an action to occur.

In addition to the visual indication of the pushbutton being pressed and released, the line (wire) that connects the pushbutton to the input momentarily changes color from red to black and then back to red to further convey the idea that the input to the device has transitioned from '1' (5 Volts DC) to '0' (0 Volts DC), then back to a '1'. The symbol within this pushbutton signifies the fact that its idle state is '1' and its active state is '0'.

Use the mouse and exercise the pushbuttons until you become comfortable with the functionality of the counter. Perform the exercises below:

Exercises

1. Confirm the proper operation of the Parallax software simulator by comparing its output to that predicted by the truth table for all possible combinations of inputs.
2. Reset the counter. With the reset input low, stimulate the clock input until the output LEDs are all off. Construct a table that correlates the output states of the LEDs to the number of clock cycles given for each state. How many cycles of the clock input are required in order to get the pattern ?

ACTIVITY #4: THE FLIP-FLOP

A flip-flop is a logic device that records the state of the data line when its clock input is triggered. Essentially, a flip-flop is a bit of memory. A flip-flop can remember one bit of information, and that one bit must be either a '0' or a '1'. It is interesting to note that eight bits used together form a byte. When you cascade 1,048,576 groups of eight flip-flops together, you get a megabyte. Yes, the same megabyte (albeit one of many) that is inside your PC. At last, we have covered enough logic to arrive at a tangible device.

If the Parallax Logic Simulator application is not already running, please launch it. Once it has started, please click on the Logic Devices button, and then choose the Flip-Flop menu button. Figure 4-14 show how the Flip-Flop menu button looks like:

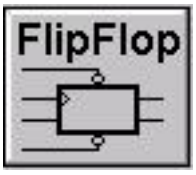

Figure 4-14
PLS – Flip-Flop

A device, similar to a bit of memory, that remembers the state of the data line when the clock input transitions high.

Figure 4-15 should be the screen that appears.

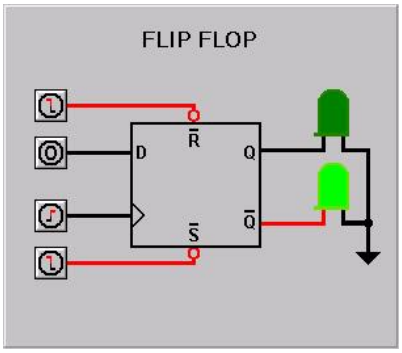

Figure 4-15
PLS - Flip-Flop

In Activity #3, the active-low transitional pushbutton was introduced. The Flip-Flop uses one of these, as well as another new type of input: the active-high transitional pushbutton.

Figure 4-16
PLS – Active-High Transitional Pushbutton

Transitions from '0' to '1' when pushed, then returns to the '0' state. Note that the symbol within the circle has two colors: black and red(gray). The black is on the bottom and is meant to signify that, before you push the button, the output is '0'. The red portion of the switch signifies that when you push this button, its output will change from '0' (black) to a '1' (red). Even though this button's output will return to a '0' after it is pushed, the symbol depicts the transition from low to high to emphasize that this is the transition that causes an action to occur.

The active-high transitional input is normally low, '0'. When pressed, (clicked on), it transitions high, momentarily outputs a '1', then returns to the low state. This type of pushbutton is useful when an active-high input is needed as in the case of the flip-flop. Another name for this type of input is "edge-triggered", because it is the edge of the transition that triggers the device into action. Figure 4-17 show how the active-high clock input schematic symbol looks like.

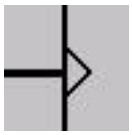

Figure 4-17
PLS – Active-High Transitional Input symbol

The horizontal line is the input lead, the ">" is the transitional input symbol, the vertical line is the perimeter of the device schematic symbol. Since there is no circle before the ">" symbol, this is an Active-High Transitional input.

The Flip-Flop has two outputs, Q and NOT Q. The bar over the lower Q indicates that it will have the opposite state of Q. The D input is the Data input. The clock input (C) is the active-high transitional type. This means that the circuit will copy the state of D to Q each time the clock input transitions high. There are two active-low inputs: NOT R and NOT S. *NOT R is the reset line. When the NOT R input transitions low, the flip-flop is reset to the state shown above. When the *NOT S input transitions low, the flip-flop is set; specifically, Q is set to '1' and, of course, NOT Q obediently switches to '0'. This behavior is depicted in the Flip-Flop's truth table in Figure 4-18.

Inputs				Outputs	
S	R	C	D	Q	$\overline{Q}$
1	⌝	X	X	0	1
⌝	1	X	X	1	0
1	1	⌟	0	0	1
1	1	⌟	1	1	0

Figure 4-18
PLS – Flip-Flop truth table [1]

[1]For most flip-flops, the Set and Reset inputs are not transitional but are in fact level sensitive. The author chose to depict the flip-flop in the simulator with transitional inputs for two reasons: to simplify the truth table, and further expose the student to transitional inputs.

Use the mouse and exercise the pushbuttons until you become comfortable with the functionality of the flip-flop. Perform the exercises below:

Exercises

1. Confirm the proper operation of the Parallax software simulator by comparing its output to that predicted by the truth table for all possible combinations of inputs.

Summary

There are many new words and ideas covered by the first four chapters. Some of those items may require clarification. The following list is provided to help clarify the more complicated terms.

HIGH and LOW

Logic low refers to the voltage of an input or output and is usually 0 Volts DC. Logic high refers to a voltage greater than 0 Volts DC, typically 5 Volts DC.

'0' and '1'

Logic '0' implies that a particular input or output is at a low voltage. Logic '1' implies that a particular input or output is at 5 Volts DC.

TRUE and FALSE

Since the words TRUE and FALSE are frequently used in spoken language, most people easily understand them. In the world of computer logic, TRUE can mean either high or low, depending on whether the device input requires an active-high input or an active-low input.

Active-high

Active-high refers to inputs or outputs that are said to be enabled when a logic high voltage is upon them.

Active-low

Active-low refers to inputs or outputs that are said to be disabled when a logic low voltage is upon them.

I/O

The term, I/O, is a technological colloquialism for Inputs and Outputs. Electronic devices have essentially three types of connections to the outside world: dedicated input pins, dedicated output pins, and configurable pins collectively referred to as I/O pins. I/O pins can be either input pins or output pins.

IPO

IPO is an acronym for Input-Process-Output. This term was not mentioned before, though the concept is important enough to cover it here. Each of the menu selections in the simulator is organized and designed to depict the inputs to the left, the process in the center, and the outputs on the right: I-P-O. This suggests a flow of reasoning from left to right. The author wished to imbue a sense of IPO in the student so that it becomes second nature that inputs are on the left, and outputs are on the right.

Truth table

A truth table is a table used to list the possible inputs and define the corresponding output for a particular logic device.

Chapter Review Questions

1. What are the differences between static and transitional inputs?
2. What are the differences between active-high and active-low inputs?
3. Describe the basic operation of a multiplexer.
4. Describe the basic operation of a demultiplexer.
5. Describe the basic operation of a counter.
6. Describe the basic operation of a flip-flop.

Chapter #5 Static Logic

ACTIVITIES IN THIS CHAPTER

1. Activity 1: Parallax Digital Trainer - Guided Tour
2. Activity 2: Wiring and Exploring Gates: AND, OR, NOT
3. Boolean Algebra Introduction
4. Activity 3: Deriving NAND and NOR Gates
5. Activity 4: Deriving XOR and XNOR Gates

By the end of this chapter, you should be able to:

1. Confirm the proper operation of the discreet gates on your PDT.
2. Identify, name, and state the purpose of each section of the PDT.
3. Wire-up and operate each kind of discreet gate on the PDT.
4. Derive the XOR function using Boolean Algebra

Parts Required

The parts used in this chapter's activities are:

(1) Parallax Digital Trainer (PDT)
(1) 9 Volt DC power supply with a 2.1mm coaxial power plug (center positive).
(1) Wire kit (Consists of three different colored rolls of 24 gauge solid wire.)
(1) Pair of wire strippers (not included in PDT kit).

24 gauge wire: is the only size of wire you should use on the PDT. Use of larger gauge wire will damage the sockets on the PDT and void your warrantee.

Now that enough of the basics have been covered, you're ready to work directly with the hardware. The following tour will guide you through several noteworthy points located on the PDT.

ACTIVITY #1: PARALLAX DIGITAL TRAINER - GUIDED TOUR

A guided tour of the Parallax Digital Trainer (PDT) will highlight the various areas of importance and acquaint you with its general operation. Please keep your arms and legs safely inside the vehicle at all times. Your PDT should look like Figure 5-1.

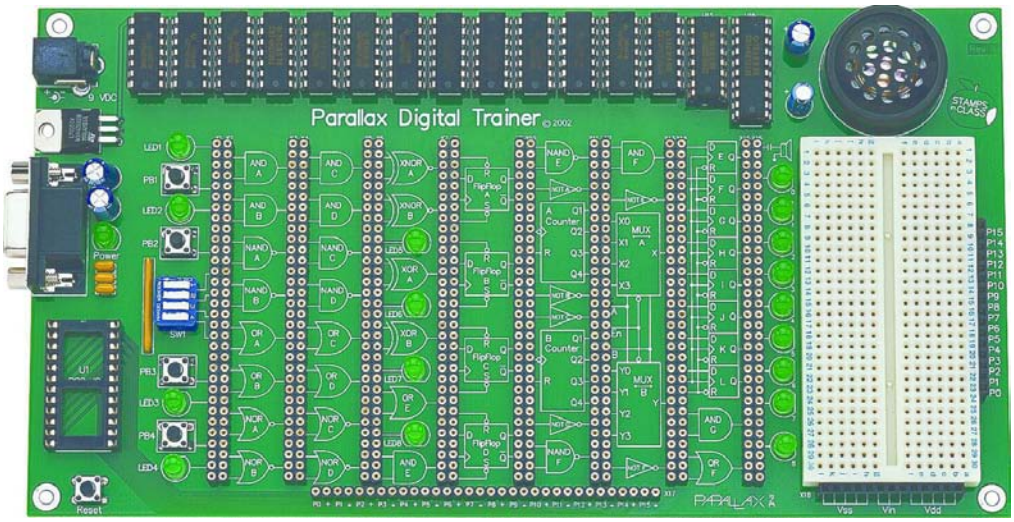

Figure 5-1**:** The Parallax Digital Trainer

General

The Parallax Digital Trainer (PDT) is designed with the idea that it will be used in a learning environment. Even though students are careful and instructors are watchful, mistakes can and do happen from time to time. All of the logic ICs (integrated circuits) are mounted in sockets that can easily be replaced if they become damaged. The breadboard (large white area) is attached to the circuit board (green part) with adhesive tape and can also be easily replaced should it become worn or damaged. The on-board jumper wire sockets (vertical black strips) are selected based on their high cycle ratings for maximum life expectancy.

Power

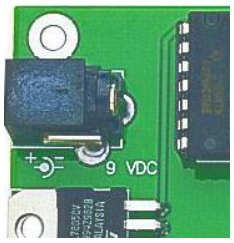

Figure 5-2
Power Connector

The power connection is located in the upper left-hand corner of the PDT, as showed in Figure 5-2. The PDT requires 9 Volts DC @ 300mA. The type of connector required is called a 2.1mm coaxial plug. Voltage supplies always have two terminals: the positive lead, and the negative lead. It is very important to connect power to the PDT observing the correct polarity. The center conductor in the power connector is the positive lead. Power may be supplied by a wall-pack style power supply or by a 9 Volt DC battery with an adapter connector.
Just below the power connector is the voltage regulator. The voltage regulator accepts 9 VDC as an input and provides a regulated 5 VDC output. This 5 VDC output is the supply for all the circuitry used on the PDT. Another name for the 5VDC output is Vcc. Vss is the name of the negative side of the power source (also called ground) and its voltage is 0 VDC.

Figure 5-3 is another view of the power connector:

Figure 5-3
Power Connector - on end, showing the center prong.

BASIC Stamp Socket

Figure 5-4 shows the BASIC Stamp Socket

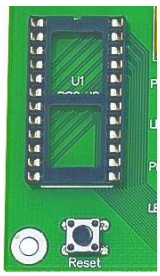

Figure 5-4
BASIC Stamp 2 DIP-24 socket and reset pushbutton

In the lower left corner of the PDT you will see a 24-pin DIP (Dual In-line Package) socket. This socket accommodates the BASIC Stamp 2, used in Chapter 9. Please note that there is a notch at the top in the center of the socket. This is the clue that pin 1 of the BS2 aligns with the pin immediately to the left of the notch.
Please note the small pushbutton below the socket. When pressed, the BASIC Stamp 2, (Stamp), is reset and when released, it will run the program it contains.

Programming/Debugging Port

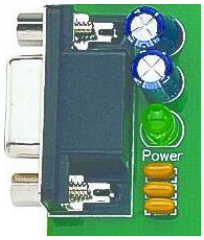

Figure 5-5
Serial Port (DB9) connection and power LED.

Near the middle of the left edge of the PDT you will find a female DB9 connector, showed in Figure 5-5. This connector will be used to connect the PDT to a PC so that the Stamp can be programmed and monitored. Note the 'Power' LED. When power is properly applied, the LED should light.

Input Devices

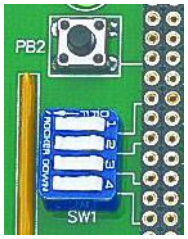

Figure 5-6
Input Devices: pushbutton and a gang of four rocker switches.

Near the left side of the PDT you will find four pushbuttons and four rocker switches. Each pushbutton is denoted with a designator, like PB2, depicted in Figure 5-6. The gang of rocker switches has the designation SW1. Each switch on the gang is further denoted: 1, 2, 3, and 4 as shown in Figure 5-6. A designator like SW1-4 is used to refer to switch 4 of the SW1 switch gang. All of the pushbuttons and all of the rocker switches are configured for Active-Low operation. However, since the rocker switches maintain their positions, (as opposed to the pushbuttons which spring back), they can be used as either active-high or active-low inputs.

Output Devices

Figure 5-7
LED

Shown with its implied connection to the jumper wire sockets.

The left side the middle and right side of the PDT all have LEDs as shown in Figure 5-7 which can be connected to outputs for the purpose of indication. Recall that LEDs require a resistor to limit the current flow to a safe level. These LEDs appear to have no current-limit (series-resistor) to limit current to a safe level. The series resistor is there; integrated into the LED itself. Therefore, it is perfectly safe to connect any source of 5 VDC to the output LEDs. Do not connect these LEDs to Vin for doing so will damage them.

Figure 5-8
Speaker

The upper right corner of the PDT is where you will find the speaker shown in Figure 5-8. Speakers require their input signals be filtered and decoupled. There are two pre-wired capacitors near the speaker that perform the decoupling and filtering. The speaker is accessed via jumper wire socket connection. Only the decoupling capacitor is depicted on the PDT and is represented by a vertical line adjacent to a slightly curved line. The filter capacitor is omitted for clarity.

Figure 5-9
Speaker connection with speaker and capacitor schematic symbol.

The connection point for the speaker can be found on the jumper wire socket designated as "X15X16", showed on Figure 5-9, on the right side of the PDT. Please note that there are two sockets, (side-by-side) you can use to connect outputs to the speaker.

Breadboard Basics

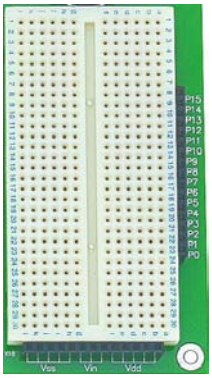

Figure 5-10
The breadboard and associated socket jumper wire sockets

A breadboard, as the one showed on Figure 5-10, is a device that allows one to assemble electrical circuits without the burden of soldering. The breadboard is divided into two sides: left and right. Each side has many holes that can accommodate 22 or 24AWG (American Wire Gauge) wires or the legs of chips (ICs). The holes on any particular side of the breadboard are electrically connected to the holes to the left and to the right. There are no electrical connections between holes that located above or below one another. Holes on the right half are not connected to any holes on the left side and vice-versa. Figure 5-11 graphically depicts these connections.

Attached to the bottom and the side of the breadboard are two jumper wire socket strips: X18 and X19. X18 contains the three connections that deal with electrical power: Vss, Vin, Vdd. Vss is the term used for ground (the negative terminal of the battery) or 0 VDC. Vin is the term used for input voltage, which is 9VDC on the PDT. Vdd is the term used for the positive lead of the battery or supply voltage; logic '1' or 5 VDC. X19 contains connections to the Stamp's 16 I/O pins.

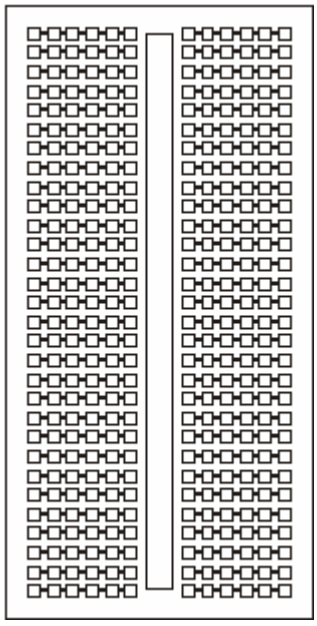

Figure 5-11
The hidden breadboard connections exposed.

A breadboard is very handy to have around. You can construct and change circuits with ease and without the hassle of soldering. Components with large diameter leads should not be used in this breadboard. The breadboard will be used in the last chapter, where the most interesting experiments and exercises are.

Work Area

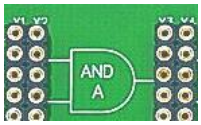

Figure 5-12
An AND Gate in the work area of the PDT.

The work area is the central part of the PDT. There are many jumper wire socket pins and schematic symbols on the printed circuit board (PCB). Depicted in Figure 5-12 is the schematic symbol for an AND Gate. It should look rather familiar except for the fact that there is currently nothing connected to the inputs or the output. That will be remedied shortly.

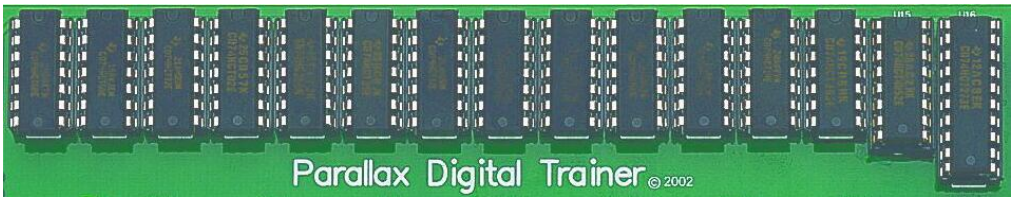

Figure 5-13: Logic ICs

Of course, the schematic symbols in the work area don't actually do anything; they are mere representations for us to use. The real work is performed by the logic ICs that are depicted in Figure 5-13. The connections to these ICs are routed to the jumper wire sockets that are adjacent to the symbols in the work area. You can look at the backside of the PDT and see this for yourself. While you are viewing the back of the PDT, please note two items printed thereon: the IC Legend, and the Logic Legend. The IC Legend is there to help if disaster strikes and you "burn up" a chip. For example, if AND Gate A no longer works, you can look it up on the IC Legend and see that U2 is associated with AND Gate A and that U2 is supposed to be a type 74C08 chip. So, you would then pop

U2 out of the socket and replace U2 with a fresh 74C08 chip. Note: despite the fact that the logic chips specified are all within the CMOS family, the actual chips used are of the HCT logic family. When possible, please replace any damaged ICs with those of the HCT family. All ICs used by the PDT are commonly available from most any electronics supplier. The Logic Legend is comprised of many truth tables and the logic symbols that they correlate to, and serves as a quick reference guide while you are in the "lab."

Figure 5-14: I/O Connectors

Throughout this course, you will need to make a variety of connections in and around the work area on the PDT. The jumper wire socket X17 showed on Figure 5-14, was designed to help you by providing access to many ground and power points, as well as connections to Basic Stamp I/O pins. Please note that none of these pins are electrically connected to their neighbors; each connection is connected exactly as depicted by the little symbol that is located immediately below the connection.

This concludes the nickel tour of the Parallax Digital Trainer. Please follow the signs to see the Giant Egress...[2]

Dos and Don'ts

- Do not wire the board while it is "hot". Construct and change the circuits only while the Parallax Logic Board is de-energized, (power not applied).
- Do not wire outputs to other outputs. Doing so may damage the hardware. Be certain that outputs are connected only to inputs. (Including the Stamp's I/O pins).
- Do be static sensitive. If unfamiliar with the proper procedures for handling static sensitive electronic devices, please review Appendix G.
- Do make sure that you understand each lesson before proceeding to the next. Each lesson is predicated upon the previous lessons.

[2] A reference to Mr. P. T. Barnum, which has nothing to do with logic, circuitry, or electronics, but has everything to do with sales and marketing.

- Remove metallic jewelry and watches before working with the PDT. Even though you will be working with low voltages, short circuits can cause excessive heat sufficient to cause painful burns, as well as damage the ICs of the PDT.
- Do not troubleshoot while hot. If your circuit is not working properly, make a quick mental note of the status of the outputs and immediately power-down the PDT. Prolonged short-circuits can cause heat and damage.
- Do allow the imagination to ask "what if" - then wire it up and see what happens (after understanding the basics).
- Do have fun. Logic can be fascinating.

Real World Note

On the PDT, the required power and ground connections for the ICs are made for you so you do not have to think about them as well as the logical connections. In the "real world", you would have to make these connections yourself. Appendix E depicts the pin configurations of most of the ICs used on this board. Other devices that may not be discussed in this text but are on the PDT may be discovered in a full schematic of the PDT available from Parallax's website.

ACTIVITY #2: EXPLORING GATES: AND, OR, NOT

By this time you should have a familiarity with how AND, OR, and NOT Gates function. That knowledge will now be put to the test. In this lesson you will locate all of the basic gates on the PDT, wire up their inputs to input devices and their outputs to output devices, and observe their operations. By accomplishing these tasks, you will verify that the discrete gates function properly on your PDT, gain confidence in basic wiring, as well as acquire a sure feeling of familiarity with basic gate functionality.

Exercises

1. Ensure that power is NOT applied to the PDT. On the PDT, locate AND Gate D ("AND D"). Using two wires connect each input to a discrete pushbutton using the adjacent black jumper wire socket strips as shown in Figure 5-15. Using one wire, connect the output to one of the LEDs. This is collectively referred to as 'wiring up a gate'. Once the wiring is complete, apply power. Your wiring should look something like the image below.

Note: Pushbuttons, frequently, are imperfect devices. Specifically, they don't simply open and close like the ones in the simulator. When closing, the contacts of real pushbuttons

bounce a few times before finally coming to rest. This phenomenon is referred to as contact bounce. The contact bounce will become more noticeable as the nature of the experiments become more sensitive to this.

Pushing the buttons slowly will aggravate the contact bounce, while pushing the buttons crisply will minimize the contact bounce. Contact bounce can make it seem like your circuits are malfunctioning. While you work through this book, it is recommended that you push the buttons as deliberately and crisply as possible to avoid contact bounce.

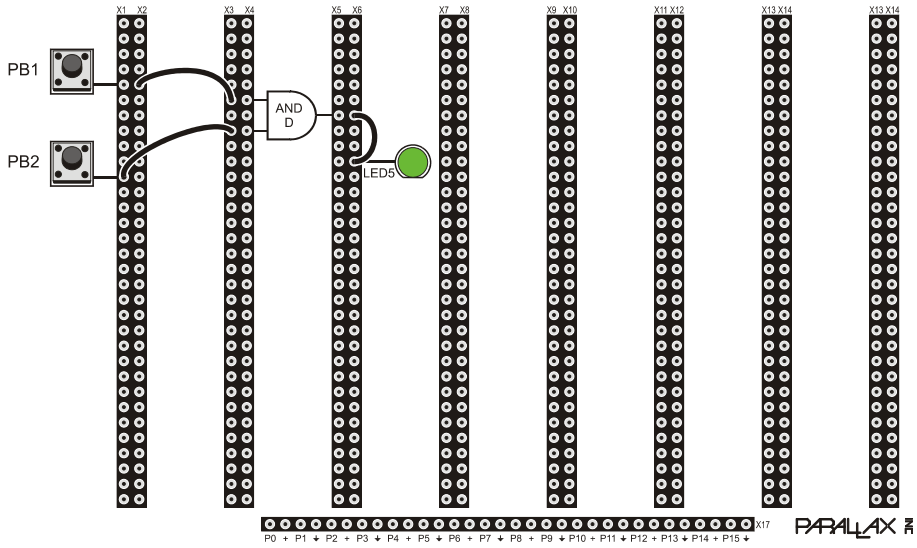

Figure 5-15: AND D properly wired.

Question: Did the output LED light right away? Why? Given the functionality of AND Gate, both inputs must be high ('1') for that to happen. Remember, all pushbuttons on the PDT are configured as Active-Low which means that they will output a '1' until pushed.

2. Considering that the type of pushbuttons on the PDT are active-high, exercise them to confirm the proper operation of the AND Gate by recording the state of the output versus the states of the inputs. Confirm the proper operation by comparing the truth table you recorded to that of the AND Gate (located on the back of the PDT).

3. De-energize the PDT, and in turn, wire each of the remaining six AND Gates, and confirm the proper operation of each gate by comparing the inputs and outputs observed to the truth table you constructed in step 2. Report any anomalies to your instructor for repair.

Information: A point for clarity's sake: Unless otherwise instructed, please de-construct the prior circuit before building the new circuit directed. In advanced lessons, when each lesson adds to the previous, you will be instructed to NOT de-construct the previous circuit.

Information: Though you may be tempted to skip through this next section, it is highly recommended that you invest the time and perform each step deliberately. Doing so will confirm the proper operation of the PDT and the repetition will benefit you greatly.

4. Repeat steps 2 and 3 for each of the five OR Gates on the PDT. Again, remember to de-energize the PDT while wiring, and test each OR Gate individually. Report any anomalies to your instructor for repair.
5. Individually, wire each of the six NOT Gates (also known as Inverters): connect the input to a pushbutton; connect the output to an LED. With the PDT powered-up, exercise the pushbutton while observing the LED connected. Confirm the proper operation of all six inverters. Report any anomalies to your instructor.
6. Individually, wire each of the four NOR Gates: connect the inputs to pushbuttons, connect the output to an LED. With the PDT powered-up, exercise the pushbuttons while observing the LED connected. Confirm the proper operation of all four NOR Gates. Report any anomalies to your instructor.
7. Individually, wire each of the six NAND Gates: connect the inputs to pushbuttons, connect the output to an LED. With the PDT powered-up, exercise the pushbuttons while observing the LED connected. Confirm the proper operation of all six NAND Gates. Report any anomalies to your instructor.
8. Individually, wire both of the XOR Gates: connect the inputs to pushbuttons, connect the output to an LED. With the PDT powered-up, exercise the pushbuttons while observing the LED connected. Confirm the proper operation of both XOR Gates. Report any anomalies to your instructor.
9. Individually, wire both of the XNOR Gates: connect the inputs to pushbuttons, connect the output to an LED. With the PDT powered-up, exercise the pushbuttons while observing the LED connected. Confirm the proper operation of both XNOR Gates. Report any anomalies to your instructor.

BOOLEAN ALGEBRA

"Use the right tool for the right job." Not exactly proper grammar, but it is a catchy phrase. Words of wisdom in any endeavor, and this endeavor's requirements are no different. Boolean algebra is the tool of choice used to design and implement logic around the world. Invented in the 19th century by George Boole, Boolean algebra provides a simple algebraic formulation of how to combine logic values. Simply put, its rules offer an easy way to describe a function and simplify the expression; the resulting expression then describes just how to implement the solution using gates. It is the fast track to the answer.

The realm of Boolean algebra consists of variables such as A, B, and Y, and values such as 0 and 1. Also in this realm is a tightly defined set of simple functions: AND, OR, and NOT. These functions behave exactly the same as the AND, OR, and NOT gates previously studied. The operations of Boolean algebra adhere to certain properties called laws. Appendix C contains a fairly complete set of Boolean laws for your reference.

The way to express an AND function using Boolean algebra is: A AND B = Y. Similarly, the way the OR function is expressed is: A OR B = Y. And, the NOT function is expressed as: NOT A = Y. Most people find it rather cumbersome to spell out the name of each function in an expression, so several shorthand versions have been developed. Lets looks at a couple of familiar functions, and how they are viewed in Boolean terms.

AND Function

Recall that the AND function requires that both inputs, A and B, need to be true for the output, Y, to be true; otherwise, the output will be false. In Figure 5-16, the truth table reveals this at a glance.

Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

Figure 5-16
The AND Function truth table.

The AND function: A AND B = Y. There are several shorthand versions widely used, but one style will be used within this text. The shorthand version of the AND function that we may use is AB = Y. This expression would be read as: A AND B = Y. Note that this

function works mathematically too. Viewed as a regular math equation, the AND function expression becomes, A x B = Y. Please note that the word AND in mathematics implies addition, but in logic, the AND operation is much more like multiplication. Just plug in the numbers from the above truth table to check this for yourself.

OR Function

Recall that the OR function requires that both inputs, A and B, need to be false for the output, Y, to be false; otherwise, the output will be true. In Figure 5-17, the truth table reveals this at a glance.

Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

Figure 5-17
The OR Function truth table.

The OR function: A OR B = Y. There are several shorthand versions widely used, but one style will be used within this text. The shorthand version of the OR function that we may use is A|B = Y. Viewed as a regular math equation, the OR function expression becomes, A + B = Y. This expression would be read as: A OR B = Y. Just plug in the numbers from the above truth table to check this for yourself.

Question: Does the OR function as expressed in a mathematical way really work for all cases? Since when does 1 + 1 = 1? The difference is in the operation itself. In the world of computer programming, the OR, AND, and NOT functions are referred to as bit operators. Trust the truth tables - they know everything about logic.

NOT Function

Recall that the output of the NOT function is merely the state of the input inverted (or negated). The NOT function's truth table of Figure 5-18 confirms this to be true.

Input A		Output Y
0		1
1		0

Figure 5-18
The NOT Function truth table.

The NOT function: NOT A = Y. There are several shorthand versions widely used, but one style will be used within this text, example:

$\text{NOT A} = \sim \text{A} = \text{A}' = \overline{\text{A}}$

The shorthand version of the NOT function that we will use is $\overline{\text{A}} = \text{Y}$. The way, this expression would read is: The NOT of A is Y.

Commutative Axiom: The operations of Boolean algebra are commutative: you can change the order of the variables without changing the meaning of the expression. Example: Given the equation AB = Y, it can be assumed that the equation BA = Y is exactly equivalent.

ACTIVITY #3: DERIVING THE NAND AND NOR GATES

Sometimes it is better to have a '1' rather than a '0'. Sometimes it is not. Because of this, different combinations of gates were created. Specifically, some gates come with inverted outputs. One could easily append a NOT Gate to the output of an AND Gate to get such an device. That is exactly what a NAND Gate is; it is a NOT - AND Gate. Thus the contraction N-AND, NAND.

From a Boolean algebraic perspective, the NAND Gate can be derived using an algebraic method called substitution. Here's how that works: Given the two equations that represent the AND function and the NOT function, AB = Y, and $\overline{\text{A}} = \text{Y}$, substitute the input of the NOT function (A) input of the AND function (AB) such that the expression becomes $\overline{(\text{AB})} = \text{Y}$. What was done was the algebraic union of the two gates thus deriving the NAND Gate.

Beware: To avoid damaging your PDT, always switch off the power before making any circuit changes.

Exercises

1. Construct a NAND Gate using an AND Gate and a NOT Gate. Here's how: Connect a pushbutton to each input of one of the AND Gates, then connect the output of the AND Gate to the input of a NOT Gate, then connect the output of the NOT Gate to an LED.

2. Construct a truth table that describes the output for all possible combinations of inputs. Does the truth table match that of the NAND Gate?
3. Rewire the circuit such that the inputs are inverted instead of the output. Here's how: Connect a pushbutton to the input of a NOT Gate and the output of that NOT Gate to the A input of an AND Gate. Connect another pushbutton to the input of another NOT Gate and the output of that NOT Gate to the B input of the same AND Gate.
4. Construct a truth table that describes the output for all possible combinations of inputs. Does the truth table match that of the NAND Gate? If not, which gate's truth table does it match?

Question: Consider the NAND Gate: is the NOT function applied to the inputs or the output to effect this gate? Since the NOT function is applied to the output wouldn't it be more appropriate to call it a AND-NOT Gate or ANDN Gate rather than a NAND Gate? The answer is clear if you think of it in Boolean terms: N(AND) = NAND. Since the NAND Gate's output is exactly opposite of the AND Gate's for any given input, we say that the function is inverted, and therefore NOT AND or NAND.

The same train of thought was used to derive the NOR Gate: NOT-OR, N-OR, NOR. When you inverted the inputs of the AND Gate, the truth table yielded the same result as that of the NOR Gate. What would you get if you inverted the inputs of an OR Gate? You are about to find out.

5. Construct a NOR Gate using an OR Gate and a NOT Gate. Here's how: Connect a pushbutton to each input of one of the OR Gates, then connect the output of the OR Gate to the input of a NOT Gate.
6. Construct a truth table that describes the output for all possible combinations of inputs. Does the truth table match that of the NOR Gate?
7. Rewire the circuit such that the inputs are inverted instead of the output. Here's how: Connect a pushbutton to the input of a NOT Gate and the output of that NOT Gate to the A input of an OR Gate. Connect another pushbutton to the input of another NOT Gate and the output of that NOT Gate to the B input of the same OR Gate.
8. Construct a truth table that describes the output for all possible combinations of inputs. Does the truth table match that of the NOR Gate? If not, which gate's truth table does it match?

ACTIVITY #4 DERIVING THE XOR AND XNOR GATES

There are still more functions beyond AND, OR, NOT, NAND, and NOR. Two frequently encountered Boolean functions are XOR and XNOR. They are not basic gates themselves, but are derived gates just as the NAND and NOR gates are. The “X” in each of the names of these two gates means “exclusive”. Two other names for the XOR gate are the Inequality Gate and the Difference Function. The reason is that the XOR's output is true only when the states of the inputs are different from one another; otherwise, the output is false. Conversely, the other names for the XNOR function are Equality Gate and Coincidence Function. The reason is that the output of the XNOR function is true only when the states of the inputs are the same; otherwise the output is false.

Given that a gate is needed to provide a true output when the inputs are different, how does one go about designing it? "*Use the right tool for the right job.*" Here is a method that you may use to derive the XOR Gate.

First, describe the problem in written terms.

> When the inputs are different, the output is true.
> When the inputs are the same, the output is false.

Then elaborate on the details of the problem so that it is fully described.

> If A = 0 and B = 0 then Y = 0
> If A = 0 and B = 1 then Y = 1
> If A = 1 and B = 0 then Y = 1
> If A = 1 and B = 0 then Y = 0

Then, translate the text into Boolean algebra.

If A = 0 and B = 0 then Y = 0	becomes	$\overline{A}\,\overline{B} = \overline{Y}$
If A = 0 and B = 1 then Y = 1	becomes	$\overline{A}B = Y$
If A = 1 and B = 0 then Y = 1	becomes	$A\overline{B} = Y$
If A = 1 and B = 1 then Y = 0	becomes	$AB = \overline{Y}$

Choose the terms that produce a true output

If A = 0 and B = 1 then Y = 1 becomes $\overline{A}B = Y$

If A = 1 and B = 0 then Y = 1 becomes $A\overline{B} = Y$

Since both conditions are required to satisfy the gate requirements (Y), simply OR them together to obtain the XOR function.

$$\overline{A}B + A\overline{B} = Y$$

There is a special symbol for the XOR function, $\oplus$. With this symbol, the XOR function can be written in shorter notation as:

$$A \oplus B = Y$$

Both expressions are valid ways to depict the XOR function. The longer notation expression offers hints that can be used to construct the physical gate. The first term is $\overline{A}B$.

To make the sub-function $\overline{A}B$ you will need to first invert the A input with a NOT Gate, then AND the result with input B. The diagrams in Figure 5-19, depict the circuit thus far and the wiring for the actual implementation showed on Figure 5-20.

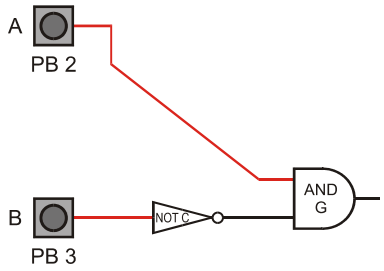

Figure 5-19
Schematic representation of the first sub-function of the derived XOR Gate.

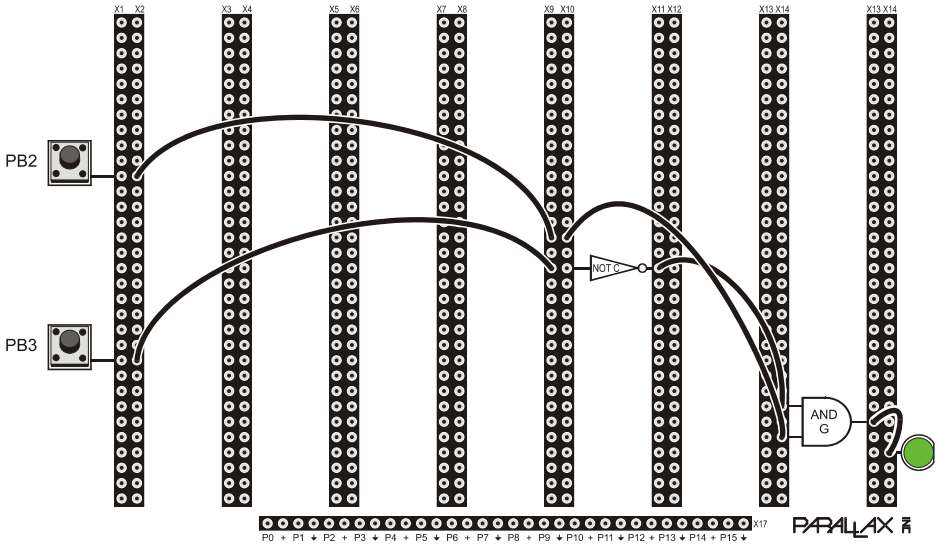

Figure 5-20: Implementation of the first sub-function of the derived XOR Gate.

The second term is $A\overline{B}$. To make the sub-function $A\overline{B}$, you will need to first invert the B input with another NOT Gate. Then AND the output of that NOT Gate with the A input, as shown in Figure 5-21 and Figure 5-22.

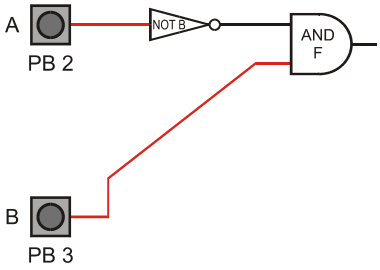

Figure 5-21
Schematic representation of the second sub-function of the derived XOR Gate.

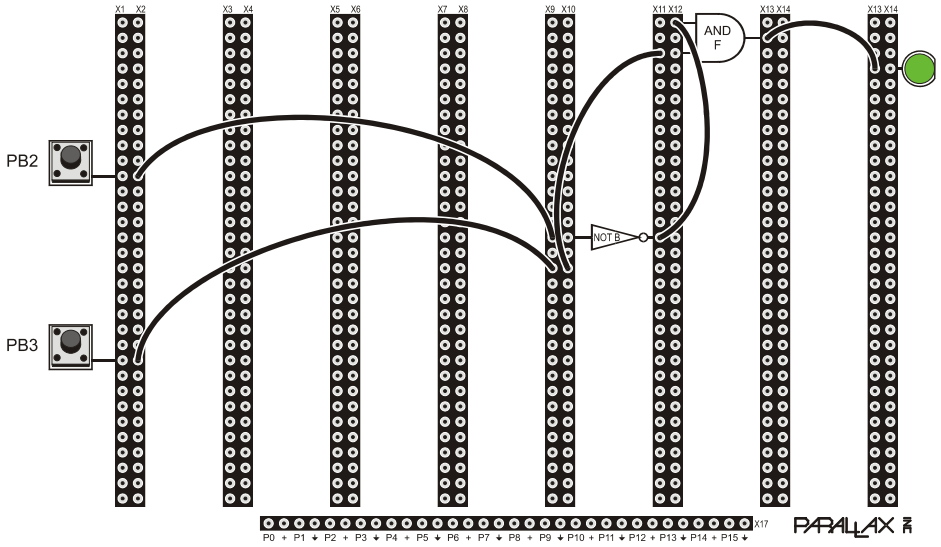

Figure 5-22: Implementation of the second sub-function of the derived XOR Gate.

Now that both sub-functions have been derived, the final step is to combine the sub-functions as described by the Boolean expression: $\overline{A}B + A\overline{B} = Y$ as showed on Figure 5-23.

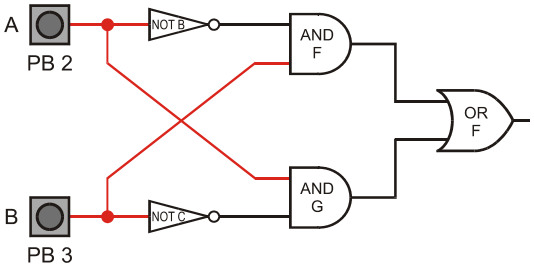

Figure 5-23
Schematic of the derived XOR Gate.

Since the two sub-functions are combined with a '+' sign, it is known that their two outputs will be combined with an OR function. The output of this OR function is the output of the XOR Gate. The wiring is showed on Figure 5-24.

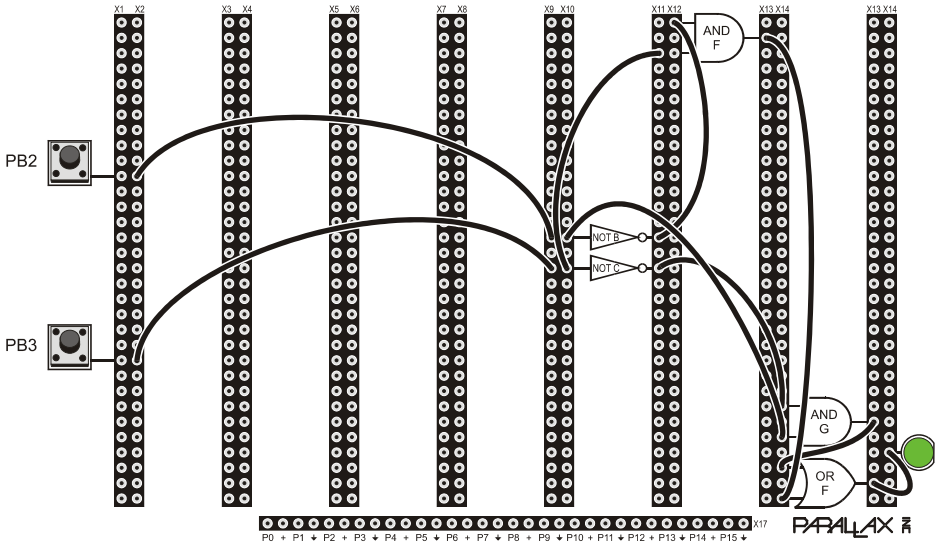

Figure 5-24: Implementation of the derived XOR Gate.

Whereas the symbols for AND, OR, and NOT are x, +, and ~ respectively, the symbol for XOR is $\oplus$. Its typical usage is $A \oplus B = Y$. Like the other gates, the XOR Gate has a few symbols that pertain to it, but in this text, the $\oplus$ will be used.

Exercises

1. Using the same method as above, derive a circuit that satisfies the requirements for the XNOR function.
2. Test the circuit: given all four possible inputs state, does the output match that of the truth table for the XNOR function?

Summary

The Basic Gates, AND, OR, and NOT, are the fundamental building blocks that all logic designs are made of. The Derived Gates, NAND, NOR, XOR, and XNOR were made so that a wider variety of devices would be available to the designer. By combining various gates together, virtually any digital circuit can be made.

Boolean algebra was developed in the 19th century by George Boole. Boolean algebra is a useful tool to the logic designer; it provides the bridge from the description to the

solution. The operations of Boolean algebra adhere to strict laws (listed in Appendix C). Boolean expressions are commutative. Specifically, the variables can be re-arranged without changing the meaning of the expression.

Exercises

1. Given that the truth table for each type of each gate yields a unique combination of four outputs, how many different types of gates are possible?
2. List the four outputs for each of these possible types of gates.
3. Are there any of these possible gates that offer little or no useful function? If so, list which gates and explain why they offer little or no useful function.
4. Using the methodology in Activity #4, derive one of the possible gates not already derived and describe the logic implementation thereof.
5. Using the PDT, implement your XOR function made of NAND Gates and verify its proper operation by recording the resulting truth table and comparing it to the known truth table of the XOR function.
6. Wire up a NOR Gate to one pushbutton and to one LED such that the NOR Gate functions as an inverter.

Extra for experts

1. Using Boolean algebra, derive an XOR function with four NAND Gates.
2. Implement and test the new derived XOR function.
3. In Activity #4, you were directed to choose the Boolean expressions that produced a TRUE output as the equations to combine when deriving the XOR function. Considering this, what would have been the outcome if you were to choose the expressions that produced a false result? Confirm your answer by deriving it algebraically, and by implementing and testing the resulting logic function.

Chapter Review Questions

1. What is the purpose of the power supply section on the PDT?
2. How are the pushbuttons (PB1 - PB4) configured?
3. What is the relationship between the row of ICs near the top of the PDT and the schematic symbols within the work area?
4. What is Boolean algebra used for?
5. Describe the method used to derive the XOR function.
6. Where is the cheat sheet located on the PDT? What information does it contain?

Chapter #6: Combinational Logic

A combinational logic design is one in which the state of its outputs depend solely on the state of its inputs. Essentially, we will be using the basic and derived gates to create and understand larger and more useful blocks of logic using combinational logic design techniques.

ACTIVITIES IN THIS CHAPTER

1. Activity 1: RS Latch
2. Activity 2: Clocked RS Latch
3. Activity 3: D Latch
4. Activity 4: Multiplexer
5. Activity 5: Demultiplexer

By the end of this chapter, you should be able to:

1. Sketch, construct, and discuss issues regarding the RS Latch.
2. Sketch, construct, and discuss issues regarding the Clocked RS Latch.
3. Sketch, construct, and discuss issues regarding the D Latch
4. Understand and discuss race conditions
5. Sketch, construct, and describe the operation of the Multiplexer
6. Sketch, construct, and describe the operation of the Demultiplexer

Parts Required

The parts used in this chapter's activities are:

(1) Parallax Digital Trainer (PDT)
(1) 9 Volt DC power supply with a 2.1mm coaxial power plug (center positive).
(1) Wire kit (Consists of three different colored rolls of 24 gauge solid wire.)
(1) Pair of wire strippers (not included in PDT kit).

Simple Latch

Frequently, it is necessary for a circuit to 'remember' its logical state, even after its input signal has been removed. Generally this is accomplished with the aid of some mechanism of feedback. The simplest form of a feedback circuit is one made of two NOT Gates and is shown in Figure 6-1

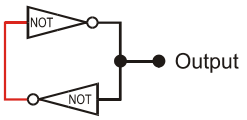

Figure 6-1
Simplest feedback circuit.

Since the output of an inverter is always the opposite of its input, this circuit will attain and retain a logic state. Two problems now exist: it is not possible to predict the state of the output, and there is no way to change the state of this memory circuit. The biggest problem with this circuit is that it has no input and therefore is not controllable. This problem results from the fact that the NOT Gate has but one input. The capabilities of this circuit can be improved by using a different kind of gate instead of the NOT Gate. It stands to reason that if we used gates with two inputs, we would be able to control one of the inputs, while the other was used for the feedback mechanism. In fact, the RS Latch does just that.

RS Latch

The circuit shown in Figure 6-2 is an RS Latch using two NAND Gates. The RS Latch circuit is comprised of two inputs called Set ($\overline{S}$) and Reset ($\overline{R}$), two NAND Gates, and two outputs called Q and $\overline{Q}$ (as spoken: Q and not Q).

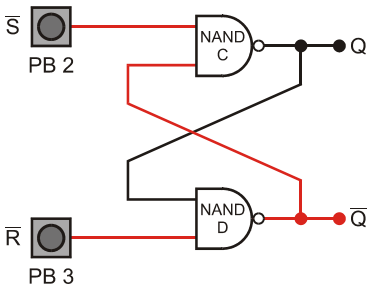

Figure 6-2
Schematic representation of the RS Latch made with NAND Gates.

This circuit requires active-low inputs to function properly. Recall, a '1' is the idle state and a '0' is the pushed (or active) state. Pressing PB2 will force the input to go to '0' and NAND B will switch its output to a '1'. This output is fed back to the input of NAND A, forcing its output to '0'. This state will remain until PB1 is pushed, then a similar action will revert the circuit back to its initial state.

ACTIVITY #1: BUILDING THE RS LATCH

Ensure the PDT is powered down first, and then build the RS Latch on the PDT. Connect the two outputs to nearby LEDs. Verify your wiring with the Figure 6-3 before applying power.

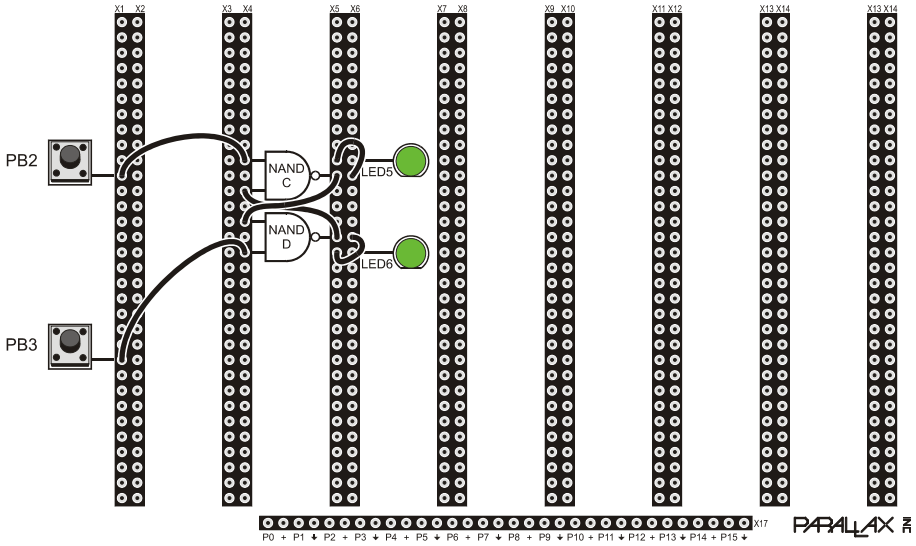

Figure 6-3: Implementation of the RS Latch using NAND Gates.

Apply power and note the state of the outputs Q and $\overline{Q}$ as reflected by the two LEDs. Push each button, one at a time, and note the behavior of the output. This circuit is called a latch because it remembers the last button pressed. Pressing same the button again does not change the output status.

Warning: It is forbidden to press both buttons at the same time. Doing so override the feedback latching action. In this condition, the gate that switched to logic '1' first will lose control while the other gate controls the latch. Note: if both inputs were to go to '1' simultaneously, a race condition would exist, making it impossible for the circuit to come to rest at one state.

Exercises

1. In your own words, describe the function of a latch.
2. What is/are the feedback mechanism(s) in the RS Latch?
3. Write a Boolean expression that represents the RS Latch. (Hint: consider either the top half or the bottom half of the latch when creating the expression.)

The RS Latch is functional, but quite limited in practical application. More often than not it is necessary to use latches in groups, and when doing so, it becomes necessary to control *when* the latches are allowed to change states. To add this functionality, we need to add two more gates and thereby create the Clocked RS Latch.

Clocked RS Latch

By replacing the NAND Gates with NOR Gates and adding an additional pair of NOR Gates to the RS Latch circuit provides a third input - the clock. A byproduct of using NOR Gates to implement this third input is that the R and S inputs are negated such that the Clocked RS Latch requires inputs that are not inverted. The circuit shown in Figure 6-4 is the Clocked RS Latch.

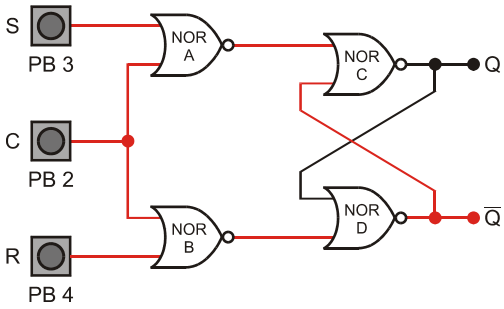

Figure 6-4
Schematic representation of the Clocked RS Latch.

The Clocked RS Latch circuit is similar to the regular RS Latch; the main difference is the clock input. By feeding the clock signal into each NOR Gate, the time during which the Clocked RS Latch may change state can be controlled. Recall that both inputs of a NOR Gate must be high in order for the output to be low. Given this, the state of the Clocked RS Latch may change states only when the clock input is high. Since the R and S signals must pass through the NOR Gate, their polarities are inverted, and therefore the Clocked RS Latch required active-high inputs to function properly.

ACTIVITY #2: BUILDING THE CLOCKED RS LATCH

Ensure the PDT is powered down first, then add the necessary circuitry to convert the previously constructed RS Latch into the Clocked RS Latch on the PDT. Please note that the pushbuttons used on the PDT are for active-low inputs. The Clocked RS Latch requires active-high inputs. Construct the circuit shown in Figure 6-5. Connect the two outputs to nearby LEDs. Verify all wiring before applying power.

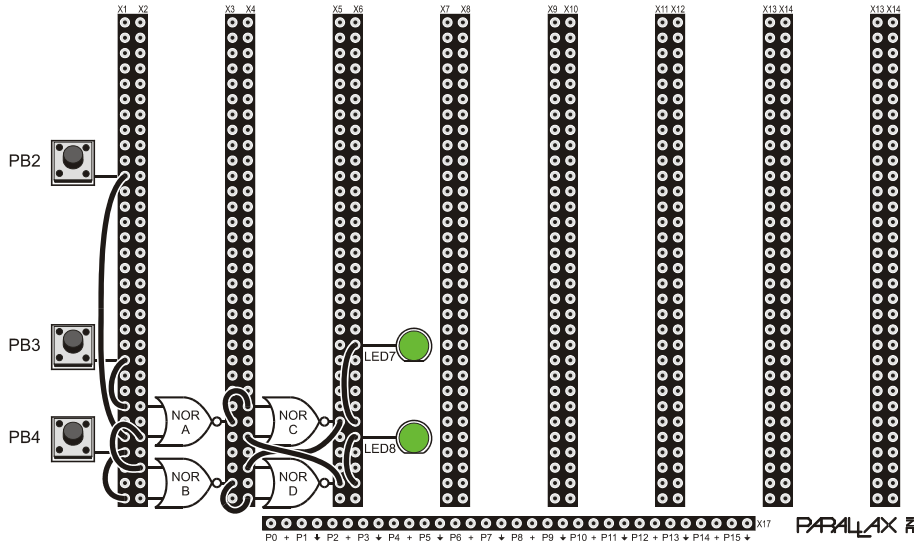

Figure 6-5: Implementation of the Clocked RS Latch.

Apply power and note the state of the outputs Q and $\overline{Q}$. Position the input pushbuttons while cycling the clock input. Try all combinations of inputs to get a feel for how the Clocked RS Latch behaves.

Question? Did the circuit behave as you anticipated for every possible combination of input states? Were the outputs, Q and $\overline{Q}$ always opposite of each other even when both R and S were high when the clock input was stimulated? The answer: no. When both R and S are high, and the clock is stimulated, the latching action is bypassed and both outputs will go to logic 1.

Race Condition is a condition in which the outputs either oscillate or fail to latch in a predictable state due to a forbidden input combination. When the forbidden input combination clears, either Q or $\overline{Q}$ will be driven to a new value before the other, finally coming to rest. Exactly which one will win the race is unknown and indeterminable.

The Clocked RS Latch is a big improvement over the standard RS Latch, but there is still room for further improvement. For instance, it may fail to work as desired when the clock is high for a period of time sufficient to allow the inputs to change more than once. One way to ameliorate that issue is to shorten the duration of the high-side of the clock pulse, but how short should the clock pulse be? Sometimes that is a very difficult to determine. Instead of trying to calculate an ideal amount of time that the clock is active, let's limit the moment of latching to that of an instant. The way to do that is by making the clock input sensitive to transitions (the edges of signals), instead of level sensitive.

Exercises

1. What state must the clock be in order for the latch to ignore both R and S inputs?
2. What advantage does the Clocked RS Latch have over the standard RS Latch?
3. Define combinational logic.
4. Describe the actions of Clocked RS Latch when all inputs are held low.

D Latch

The circuit shown in Figure 6-6, is called the D Latch. The D Latch is comprised of two inputs called Data (D) and Clock (C), four NOR gates, and two outputs called Q and $\overline{Q}$. The D Latch is a made by varying the design of the Clocked RS Latch.

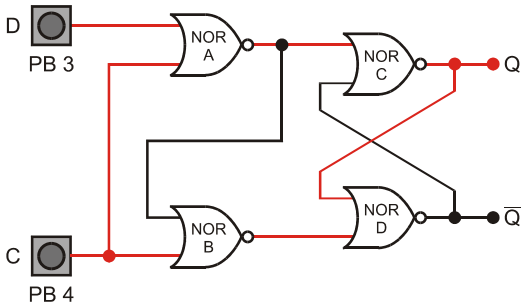

Figure 6-6
Schematic representation of the D Latch.

There are two differences between the D Latch circuit and the Clocked RS Latch: The Set input has been renamed to D for data, and what was the Reset input now comes from the output of NOR A. This rewiring allows NOR A to function as a NOT Gate in that it inverts the D input when the clock is low. By splitting the single D input into two differential signals (signals that are always opposite in state with one another), the D Latch circuit is not subject to the possibility of a race condition. Other than that, the D Latch functions just as the Clocked RS Latch.

ACTIVITY #3: BUILDING THE D-LATCH

Ensure the PDT is powered down first, and then wire-up the D Latch on the PDT. Verify your wiring before applying power with the Figure 6-7.

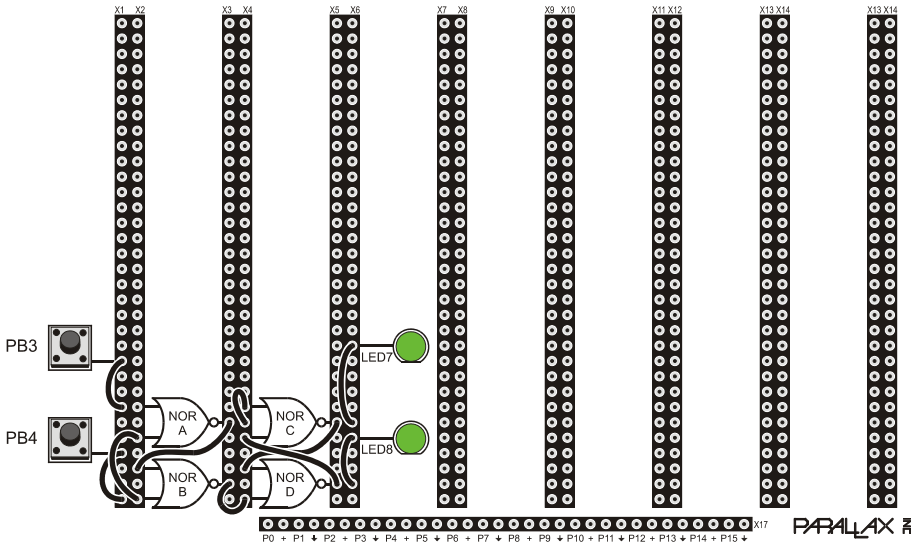

Figure 6-7: Implementation of the D-Latch.

Exercises

1. Apply power, exercise the inputs, note the outputs, create a truth table that describes the behavior of this device.
2. Why was it necessary to invent the D-Latch; wasn't the Clocked RS Latch good enough?
3. What is/are the consequence(s) of holding both C and D low on the D-Latch?

4. What is the great advantage that the D-Latch has over the Clocked RS Latch?

Multiplexer

The circuit shown in Figure 6-8, is called the 2-Input Multiplexer. The 2-Input Multiplexer is comprised of two NOT Gates, two AND Gates, and one OR Gate. These components can be grouped into three functions that work together to perform the switching function.

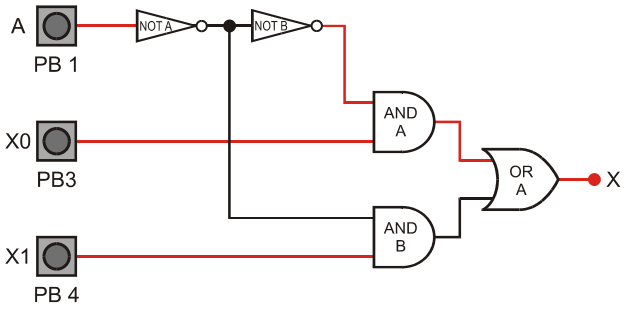

Figure 6-8
Schematic representation of the 2-Input Multiplexer.

The two NOT Gates form the address decoder. The address decoder's function is to enable, based on the A input, either AND A or AND B, but never both. By splitting the address line (A) into two differential signals and feeding the opposing signals into the A and B AND Gates, it is guaranteed that only one AND Gate is "enabled" at any given time. The next section is the switch section which encompasses AND Gates A and B. Enabled or disabled by the decoder section, the switch section passes either X0 or X1 to the summing section. The OR Gate simply combines (sums) the outputs of the AND Gates and passes it to the X output.

ACTIVITY #4: BUILDING THE MULTIPLEXER

Ensure the PDT is powered down first, then remove the previous activities wiring. Add the necessary wiring to build the 2-Input Multiplexer on the PDT. It is recommended that you connect an LED to the output (X). Verify your wiring with Figure 6-9, before applying power.

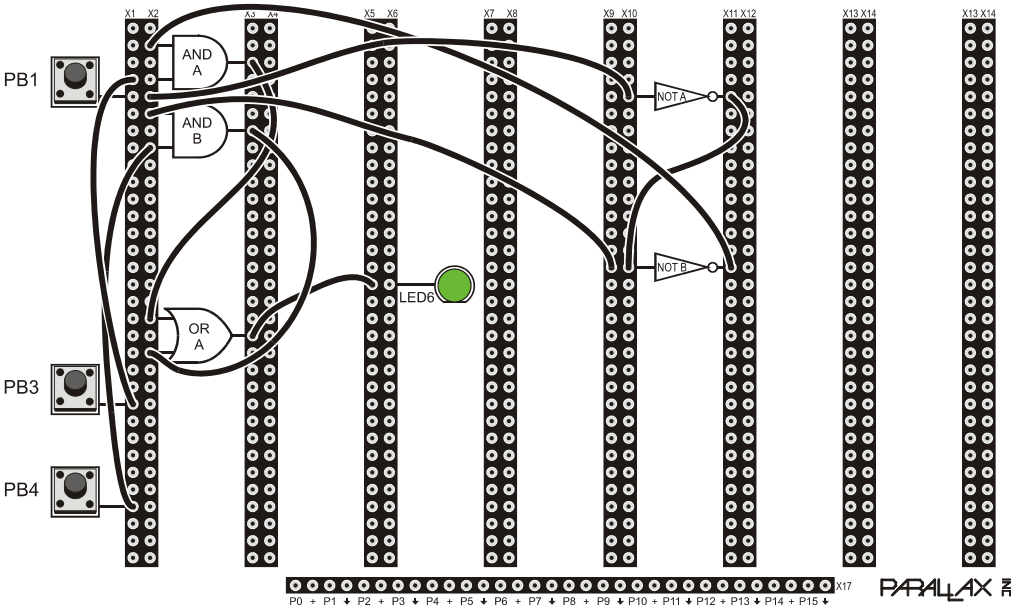

Figure 6-9: Implementation of the 2-Input Multiplexer.

Exercises

1. Apply power, exercise the inputs, note the outputs, and create a truth table that describes the device.
2. What is the overall purpose of the 2-Input Multiplexer?
3. How many inputs does the 2-Input Multiplexer have?
4. How many address inputs would be needed to support a 4-Input Multiplexer?

Demultiplexer

The circuit shown in Figure 6-10, is called the 2-Output Demultiplexer. The 2-Output Demultiplexer is comprised of two NOT Gates and two AND Gates. These components can be grouped into two sub-functions that work together to perform the selecting action.

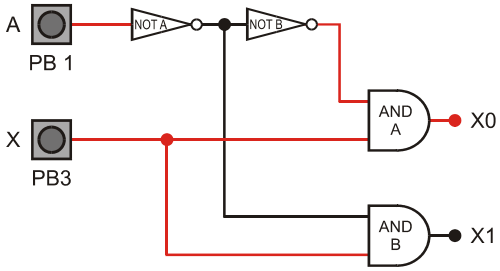

Figure 6-10
Schematic representation of the 2-Output Demultiplexer.

The two NOT Gates form the address decoder. The address decoder's function is to enable, based on the A input, either AND A or AND B, but never both. By splitting the address line (A) into two differential signals and feeding the opposing signals into the A and B AND Gates, it is guaranteed that only one AND Gate is 'enabled' at any given time. The next section is the selector section which encompasses AND Gates A and B. Based on the outputs of the decoder section, either AND A or AND B is 'enabled'. Which ever one is enabled gets to pass its signal to its output, be it X0 or X1.

ACTIVITY #5: BUILDING THE DEMULTIPLEXER

Ensure the PDT is powered down first. Since the demultiplexer is so similar to the multiplexer, you may simply modify the existing wiring to effect the 2-Output Demultiplexer. Verify your wiring with Figure 6-11 before applying power.

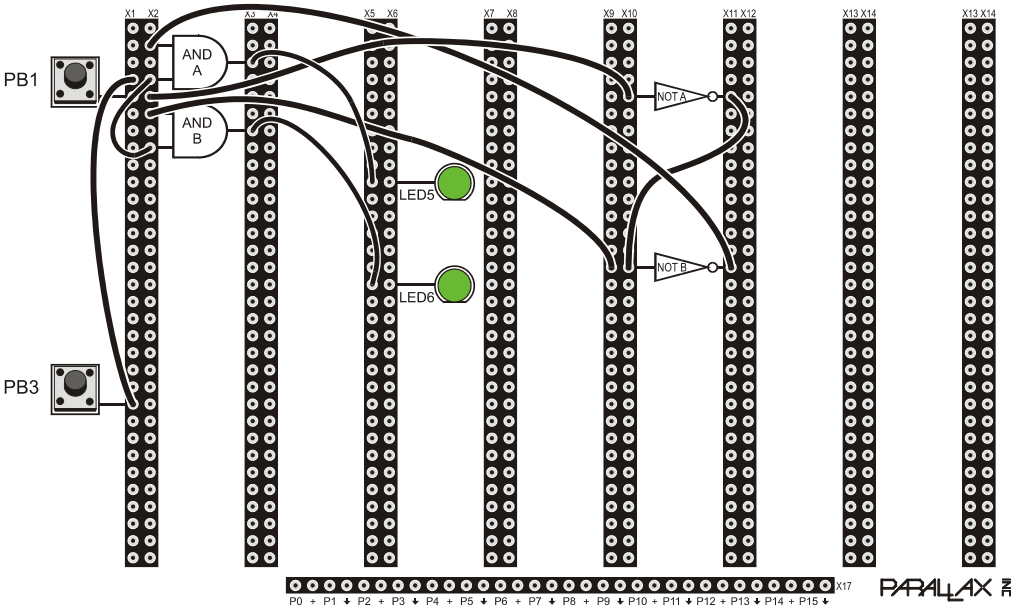

Figure 6-11: Implementation of the 2-Output Demultiplexer.

Exercises

1. Apply power, exercise the inputs, note the outputs, and create a truth table that describes the device.
2. What is the overall purpose of the 2-Output Multiplexer?
3. Think of a way to implement the 2-Output Multiplexer using one less gate. Build the new circuit and test it against the truth table created in Exercise #1.
4. Explain why this device is unidirectional.

Extra for Experts

1. Design, implement, and test a bidirectional multiplexer.

Chapter Review Questions

1. What comprises the most simple latch circuit?
2. What is this latch virtually useless?
3. How does the RS latch improve upon this design?
4. Describe the feedback latching action of the RS Latch.
5. What is a forbidden input?
6. What is a race condition?
7. What is the purpose of the Clocked RS Latch?
8. Describe the shortcomings of the Clocked RS Latch.
9. How does the Edge-Triggered RS Latch fix this problem?
10. Is the Edge-Triggered RS Latch prone to a race condition?
11. How does the D Latch fix this problem?
12. What is a multiplexer used for?
13. What is a demultiplexer used for?

Chapter #7: Sequential Logic

Sequential logic designs depend not only on the states of their inputs, but also rely on the state of memory of past events. In this chapter, we will use combinational circuits and memory circuits together to build even more interesting stuff.

ACTIVITIES IN THIS CHAPTER

1. Activity 1: Edge-Triggered RS Flip-Flop
2. Activity 2: D Flip-Flop
3. Activity 3: Ring Oscillator
4. Activity 4: Binary Counter
5. Activity 5: Frequency Divider

By the end of this chapter, you should be able to:

1. Explain the differences between Combinational and Sequential Logic.
2. Construct and discuss issues regarding the Edge-Triggered RS Latch.
3. Construct and discuss issues regarding the D Flip-Flop.
4. Construct and discuss the Ring Oscillator.
5. Construct and discuss Binary Counters and Frequency Dividers.

Parts Required

The parts used in this chapter's activities are:

(1) Parallax Digital Trainer (PDT)
(1) 9 Volt DC power supply with a 2.1mm coaxial power plug (center positive).
(1) Wire kit (Consists of three different colored rolls of 24 gauge solid wire.)
(1) Pair of wire strippers (not included in PDT kit).

Edge-Triggered RS Flip-Flop

The circuit shown in Figure 7-1, is the Edge-Triggered RS Flip-Flop. The Edge-Triggered RS Flip-Flop circuit is comprised of two inputs called Reset and Set, eight gates, and two outputs called Q and $\overline{Q}$.

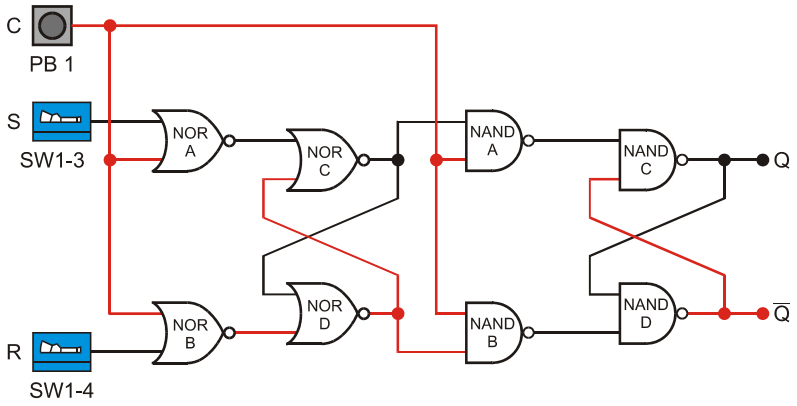

Figure 7-1: Schematic representation of the Edge-Triggered RS Flip-Flop.

The R and S inputs require active-high switches to function properly. The clock input requires an active-low pushbutton. Essentially, this circuit is simply two RS Latches positioned back-to-back. The first section (input side) is made with four NOR Gates. Using NOR Gates in this stage blocks the R and S signals while the clock input is '1'. Only when the clock pushbutton is pressed (producing a '0') will the R and S signals be commuted to the input side of the second stage. The second stage is comprised of NAND Gates. Note that the same clock input is connected to both stages. Using NAND Gates in the second stage blocks the R and S signals while the clock input is '0'.

This two-stage approach causes a ratchet-like action that allows the outputs to change only when there is a complete cycle (high-low-high) on the clock input. Each time the clock changes from high to low, the first stage is loaded with the input D and its opposite state. Every time the clock transitions from low to high, the output from the first stage D Latch is loaded into the inputs of the second stage RS Latch. Using this approach causes the state of the input to be "remembered" when the clock transitions from high to low,

and also causes the outputs states to be updated with what was remembered when the clock then transitions from low to high.

It is interesting to note that if you joined several of these together (input connected to output and all the clocks connected together), you could create a line of bits and march them from the beginning to the end just by toggling the clock several times.

ACTIVITY #1: BUILDING THE EDGE-TRIGGERED RS FLIP-FLOP

Ensure the PDT is powered down first, and then build the Edge-Triggered RS Flip-Flop, as shown on Figure 7-2, on the PDT. It is recommended that you connect LEDs to the outputs of both the first and second stages. Verify your wiring before applying power.

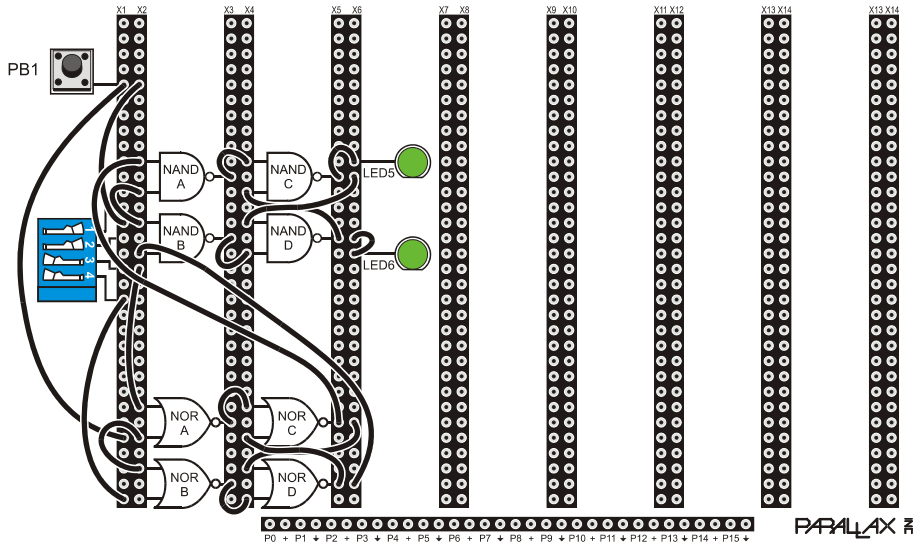

Figure 7-2: Implementation of the Edge-Triggered RS Flip-Flop.

Exercises

1. Apply power, exercise the inputs, note the outputs, and create a truth table that describes the device's operation.
2. Is it possible to throw this device into a race condition? Explain your answer.
3. How could you modify Figure 7-2 to function with eight NAND Gates instead of four NAND and four NOR Gates?

The Edge-Triggered RS Flip-Flop is functional, but still posses a problem: it is subject to a race condition. A slight change to the RS-Latch circuit will yield a latch that is not inherently subject to race conditions: the D Flip-Flop

D Flip-Flop

The circuit shown in Figure 7-3 is the D Flip-Flop. The D Flip-Flop circuit is very similar to the Edge-Triggered RS Flip-Flop. Comprised of two inputs, clock (C) and data (D), eight gates, and two outputs called Q and $\overline{Q}$, the only differences between it and the Edge-Triggered RS Flip-Flop are: The Set input has been renamed as the data (D) input, and the Reset input has been stripped of its named and simply tied to the output of NOR A.

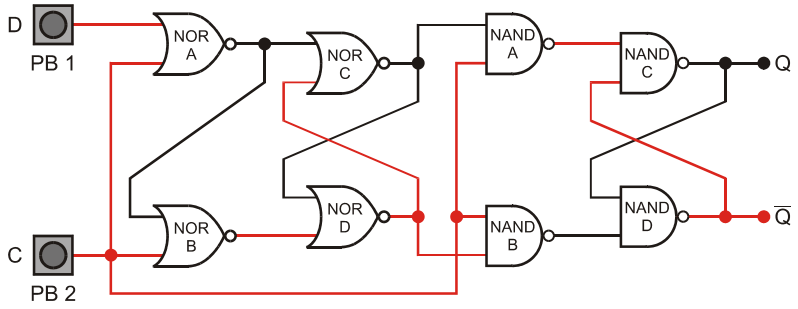

Figure 7-3: Schematic representation of the D Flip-Flop.

The D and C inputs require active-low switches to function properly. Essentially, this circuit is simply a D-Latch abutted to an RS Latch. The first section (input side) is made with four NOR Gates and the second stage is made with four NAND Gates.

Using NOR Gates in the first stage blocks the D signal while the clock input is '1'. Only when the clock pushbutton is pressed (producing a '0') will the D signal (and its counterpart) be commuted to the output of the first stage and therefore the input side of the second stage. Using NAND Gates in the second stage blocks the D signals while the clock input is '0'. This two-stage approach allows the outputs to change only when there is a complete cycle (low-high-low again) on the clock input. Using this approach causes the state of the input to be "remembered" when the clock transitions from high to low, and also causes the outputs states to be updated with what was remembered when the clock then transitions from low to high.

ACTIVITY #2: BUILDING THE D FLIP-FLOP

Ensure the PDT is powered down first, and then build the D Flip-Flop on the PDT, as shown on Figure 7-4. It is recommended that you connect LEDs to the outputs of both the first and second stages. Verify your wiring before applying power.

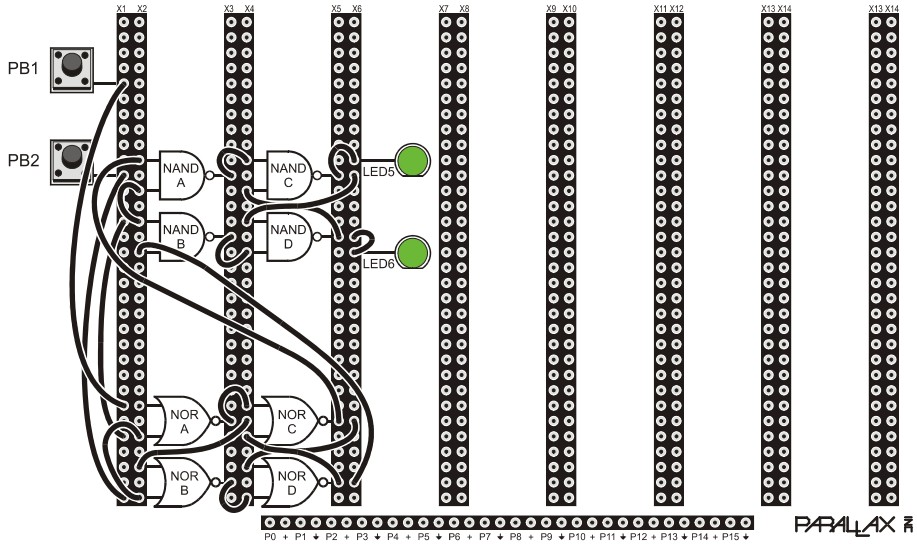

Figure 7-4: Implementation of the D Flip-Flop

Exercises

1. Apply power, exercise the inputs, note the outputs, and create a truth table that describes the device's operation.
2. Is it possible to throw this device into a race condition? Explain your answer.
3. How could you modify Figure 7-4 to function with eight NOR Gates?

The D Flip-Flop is functional and it inherently resists the possibility of obtaining a race condition. Eight of these working together could be used to store a byte of data.

Oscillators

Oscillators are devices in which their outputs automatically vary between high and low at a steady rate. Often in electronics it is necessary to stimulate a circuit with an oscillating signal. The oscillation itself drives the circuit while the period of oscillation (the time between two the beginning of two high points) determines the rate at which circuit operation occurs. One handy example is that of the PC, as shown in Figure 7-5: the microprocessor, the heart of the computer, requires an oscillating input that is generated by the clock oscillator. So, if your PC runs at 1.8 Giga Hertz, that means that your PC's microprocessor is executing 1,800,000,000 instructions per second. [3]

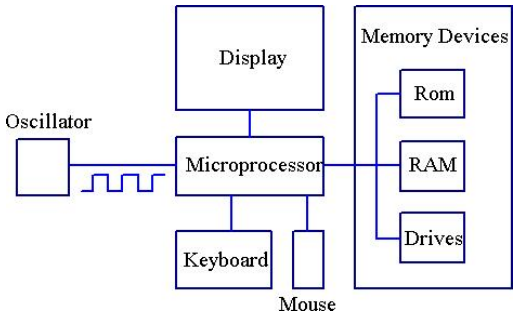

Figure 7-5
Typical PC Block Diagram

The clock drives all and establishes the rate of execution.

Ring Oscillator

Ring oscillators are oscillators that are made by placing a series of NOT Gates in a ring, or closed-circle formation. Recall the beginning of Chapter 6: the Simple Latch circuit. Basically, it was comprised of two NOT Gates in just such a formation, though it is not considered an oscillator, let alone a Ring Oscillator. This is true due to the fact that it cannot oscillate. However, if there were a third NOT Gate in series with the other two, it certainly would have.

Ring configurations that bear an even number of NOT Gates are stable - they do not oscillate. Ring oscillators that bear an odd number of NOT Gates are not stable and will oscillate automatically. The frequency at which a ring oscillator oscillates depends on two things: the number of NOT Gates in the ring, and the time it takes for a NOT Gate to change its output when its input changes. The latter is referred to as the propagation delay

[3] Assuming your microprocessor executes one instruction per clock cycle.

time, and depending on the particular attributes of the NOT Gate, is typically 50 nS. (50 nano-Seconds = 0.00000005 Seconds). Figure 7-6 shows an example of a five-stage ring oscillator using five NOT Gates

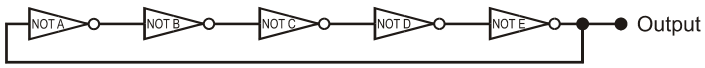

Figure 7-6 Schematic representation of a five-stage ring oscillator.

To understand how this circuit works, you must assume a starting point. It is best to start at the beginning. Assume the input to NOT A is '0'. After a short period of time, (the propagation time of a NOT Gate), NOT A's output will change to a '1'. This change is immediately "felt" on the input of NOT B and will cause NOT B's output to go to a '0'. Which, in turn is "felt" in the input of NOT C thereby changing NOT C's output to go to a '1'. This causes NOT D to change its output to a '0'. Which causes NOT E to change its output to a '1'. The output of NOT E is the output of this ring oscillator and it is also the input to the oscillator as well. So, this '1' is fed back into the input side of NOT A, (which was previously assumed to be a '0'). Since the signal present on the input side of NOT A is now the opposite of what it was, the opposite state is propagated through the entire chain. The sequence of events repeats, endlessly.

ACTIVITY #3: BUILDING THE RING OSCILLATOR

Ensure the PDT is powered down first, remove prior circuits, and build the Ring Oscillator on the PDT shown in Figure 7-7. It is recommended that you connect LEDs to the outputs of both the first and second stages. Verify your wiring before applying power.

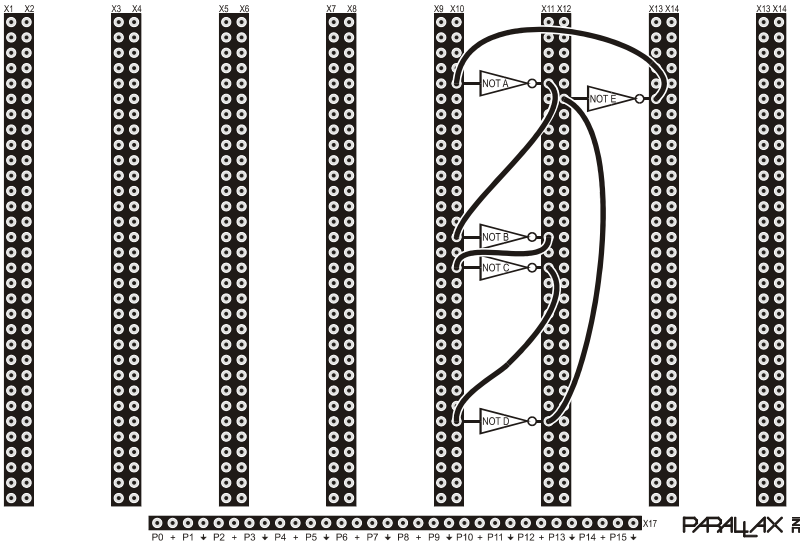

Figure 7-7: Implementation of the Ring Oscillator.

Exercises

1. Apply power and note the output. Is it working properly?
2. Calculate the rate at which this circuit should cycle from high to low.
3. How many cycles per second is this circuit generating?
4. Calculate the rate at which a seven stage ring oscillator would oscillate.
5. Optional - view the output with an oscilloscope.
6. Optional - using the oscilloscope, measure the period of oscillation and calculate the frequency in cycles per second.

Frequency Divider

Within digital systems, it is often necessary that some functions run at a slower rate than others. When designing such systems, the clock is set to run at the fastest rate needed, and a device known as a frequency divider is used to generate clock signal(s) used to run the slower functions. For example, your computer may run at 1.8 Giga-Hertz, but you certainly would not want your keyboard auto-repeat function to run that fast as well. (The keyboard auto-repeat function is responsible for repeating the keystrokes automatically when you hold down a key for a lengthy period of time.)

Recall that a byproduct of a two-stage latch function was a delay (one cycle of the clock input) from the time the input changed until the time that the output reflected that change. We can capitalize on that effect by cascading several two-stage latches together, thereby causing a significant delay. Specifically, we will be using the D Flip-Flops already located on the PDT (instead of hand-wiring the derivations) in designing our frequency divider. Figure 7-8 is a schematic showing a frequency divider made with D Flip-Flops.

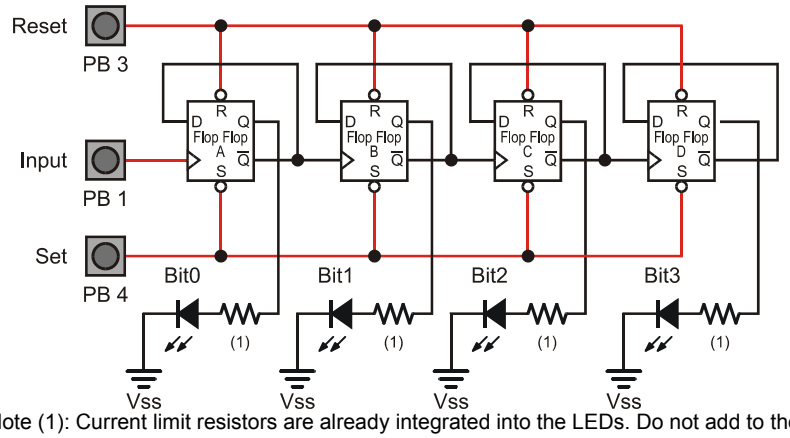

Figure 7-8 Schematic representation of a frequency divider made with D Flip-Flops.

Note (1): Current limit resistors are already integrated into the LEDs. Do not add to them to the circuit.

In this circuit, the input pushbutton is connected to the clock input of Flip-Flop A. The D input is tied to the inverted output, $\overline{Q}$. Every time the clock input pushbutton is pressed and released, the opposite state of D is fed back into D. This causes the output of Flip Flop A to oscillate high and low. Moreover, this circuit causes Flip-Flop A to oscillate at *one-half* the rate of the input. By tying the $\overline{Q}$ output of Flip-Flop A to the C input of Flip-Flop B, and connecting the $\overline{Q}$ output of Flip-Flop B to Flip-Flop B's D input, the

process can be repeated thereby dropping the rate of oscillation by another factor of two. Flip-Flops C and D are also connected in a similar fashion. The LEDs serve to indicate that indeed each successive stage oscillates at exactly half the rate of the previous.

ACTIVITY #4: BUILDING THE FREQUENCY DIVIDER

Ensure the PDT is powered down first, and then build the Frequency divider on the PDT, as shown in Figure 7-9. Verify your wiring before applying power.

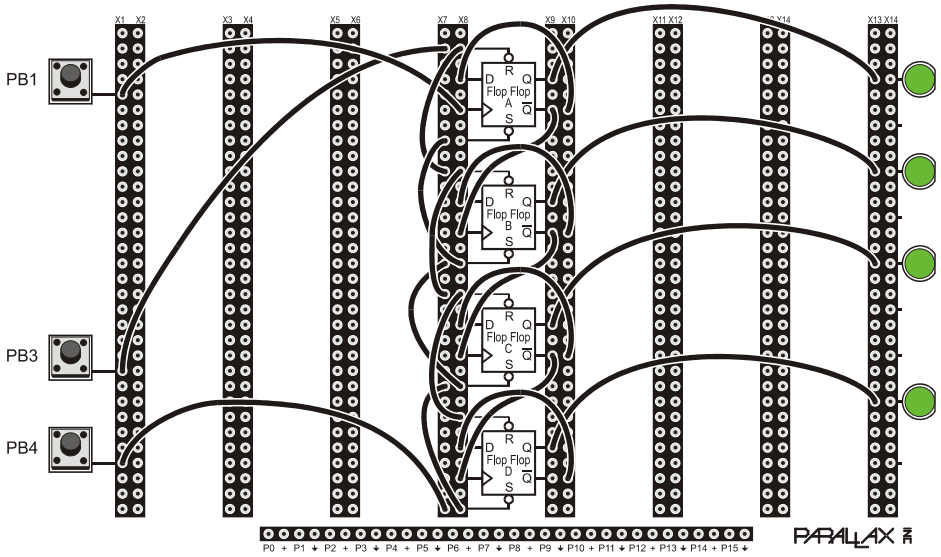

Figure 7-9: Implementation of the Frequency Divider.

Exercises

1. Apply power and note the output. Is it working properly?
2. If the input was oscillating at a rate of eight times per second, what would the rates of oscillation be for each of the stages of the frequency divider?
3. Make a chart that correlates the number of pushes on the pushbutton to the states of the outputs on the LEDs.
4. What, if any, is the pattern that is formed?

Binary Counter

Binary Digital Systems are digital systems that deal with only two numbers: 0 and 1. 0 and 1 are sufficient to define such attributes such as night/day, on/off, and yes/no, but they alone are not sufficient to describe more complex items that we encounter every day. How old are you? Well, you could answer 1, or 10, or 11, or 100, etc. but only on a few select years of your life could you answer accurately. It is by forming a number system with zeroes and ones that we are able to encode binary characters to mean so much more. If you aren't up-to-speed on binary numbers, please read Appendix D before proceeding.

Many applications require that numbers be remembered, compared to, and used for counting to complete the tasks that are required of them. Frequently, *binary counters* are used for just such purposes. By gaining a familiarity with binary counters the student will be enabled to design and implement projects that must use numbers.

Coincidentally, the frequency divider of the previous activity also functions as a binary counter. Again, the schematic for this is shown in Figure 7-10.

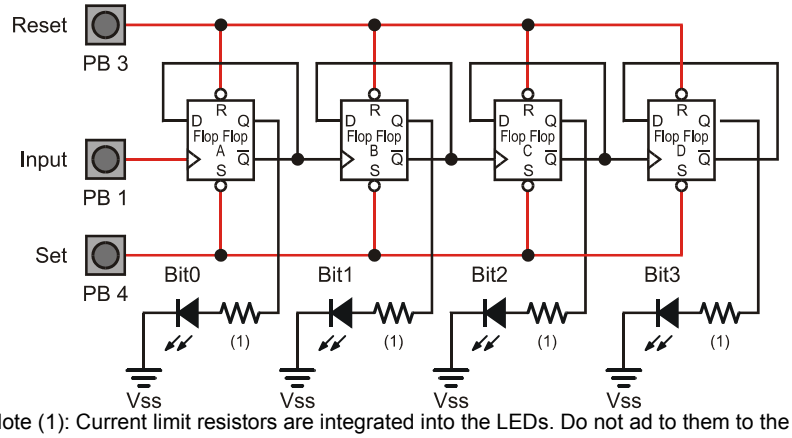

Figure 7-10 Schematic representation of a 4-bit binary counter made with D Flip-Flops.

Note (1): Current limit resistors are integrated into the LEDs. Do not ad to them to the circuit.

First, clear all the Flip-Flops to the reset state (that is, all Q = 0) by pushing PB1. This should cause all four LEDs to extinguish. Pressing the input pushbutton (PB3) will cause the opposite state of Q to be latched in Flip-Flop A. Since all four Q outputs were off, this means that Flip-Flop A will become "set" and the LED attached to it will light. This action also causes the $\overline{Q}$ output (which is the input to Flip-Flop B) to go low. The clock

input on the D flip-flops used herein is of the active-high type, so nothing further will occur at this time.

Pressing PB3 again will cause Flop-Flop A to latch back to the reset state (Q off). Since Q is off, $\overline{Q}$ will be on, and this high signal on the clock input of Flip-Flop B will cause Flip-Flop B to latch the set state (Q on). The net result of this operation: 2 cycles on PB3, and the state of the Q outputs from LSB to MSB: 0,1,0,0 = 2 decimal.

ACTIVITY #5: BUILDING THE BINARY COUNTER

This circuit is identical to the frequency divider built in Activity #3. If you need to rebuild this circuit, ensure the PDT is powered down first, and then rebuild the Binary Counter, as shown in Figure 7-11.

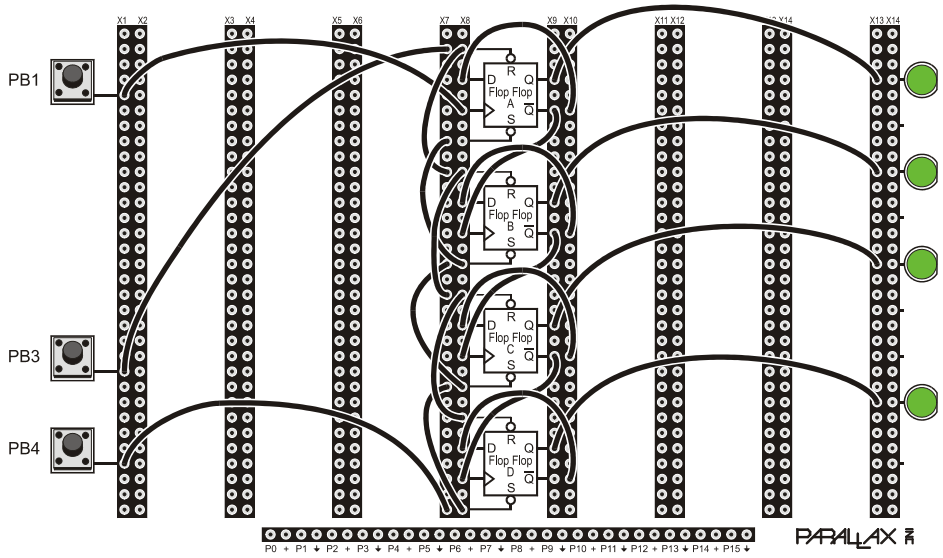

Figure 7-11: Implementation of the Binary Counter.

Exercises

1. Apply power, cycle the set and reset inputs. Is it working properly? Create a truth table for this circuit.
2. Describe what happens when either the set or the reset pushbuttons are held and the input pushbutton is pressed.
3. Is it possible to drive this circuit to a race condition using just the three input pushbuttons? Explain.
4. What is the largest number that this binary counter can count up to?
5. How many different combinations of states can this binary counter hold?
6. What happens to this counter when a clock input occurs when the counter is holding its maximum count?

Extra for Experts

1. Rewire the existing circuit such that it counts backwards, (1111 to 0000).

Chapter Review Questions

1. Describe what it means to be a Combinational Logic Design.
2. Describe what it means to be a Sequential Logic Design.
3. What purpose does the Edge-Triggered RS Latch serve?
4. Is it possible to cause a race condition in an Edge-Triggered RS Latch? How?
5. How does the D Flip-Flop differ from the Edge-Triggered RS Latch?
6. Is it possible to cause a race condition in a D Flip-Flop?
7. How would you calculate the speed of any particular Ring Oscillator?
8. What NOT Gate combinations would cause a Ring Oscillator to work properly?
9. State the differences between a frequency divider and a binary ripple counter.
10. What are frequency dividers used for?

Chapter #8: Handy Circuits

In the near future you will graduate and subsequently evolve into a professional. Once your metamorphosis is complete, you will inevitably be asked to design a solution for a problem. The first step in solving any problem is to clearly define the task to be accomplished. The next step is to break that task into "sub-functions." This process continues until all the sub-functions are reduced to the point where they seem easy to implement. Then, each sub-function is meticulously implemented and individually tested. Once all the sub-functions are complete and work perfectly, they are combined and tested as a whole. Of course, it never works the first time through. And, after the panic wave ripples through the fabric of your soul because your deadline is about to expire, the troubleshooting begins. The trouble, invariably, is caused by a ubiquitous computer bug: the PEBCAC[4].

At this point, it is necessary to turn our attention toward using all of the aforementioned knowledge to build the sub-functions that may be needed to build functional applications in the future, and in Chapter 9.

ACTIVITIES IN THIS CHAPTER

1. Activity 1: Half-Adder
2. Activity 2: Full-Adder
3. Activity 3: Shift Register
4. Activity 4: Casting Logic
5. Activity 5: Short Counts

By the end of this chapter, you should be able to:

1. Design and implement a Half-Adder circuit.
2. Design and implement a Full-Adder circuit.
3. Design and implement a Shift Register.
4. Design and implement short counters.
5. Use gates for multiple purposes.
6. Compute the largest number that can be held by a register or a counter

[4] PEBCAC is an acronym for: Problem Exists Between Chair And Computer

Parts Required

The parts used in this chapter's activities are:

(1) Parallax Digital Trainer (PDT)
(1) 9 Volt DC power supply with a 2.1mm coaxial power plug (center positive).
(1) Wire kit (Consists of three different colored rolls of 24 gauge solid wire.)
(1) Pair of wire strippers (not included in PDT kit).

Binary Addition

Frequently it is necessary for a digital system to be able to perform mathematical functions. More often than not, those mathematical functions are predicated on the ability of digital logic to be able to add two numbers together. The easiest starting point here is to add two bits together. Consider the three possibilities below:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$

Does this pattern look familiar? It should, for it is 3/4 of the truth table for either an OR Gate or a XOR Gate. Either way, that part of this adder should be easy to implement. Consider the fourth and final possibility:

$$1 + 1 = 10$$

Recall that during counting, any carry over results in the implied leading zero to be incremented to 1 while the less significant digit was cleared to zero. Also, the LSB of the answer completes the truth table and it is revealed that indeed an XOR Gate is required to implement at least part of the adder. Consider the digit with the implied 0 over the four possibilities.

$$0 + 0 = 00$$
$$0 + 1 = 01$$
$$1 + 0 = 01$$
$$1 + 1 = 10$$

Viewing the problem this way reveals the two truth tables needed to design a solution. The LSB, (0, 1, 1, 0) can be solved with an XOR Gate. The MSB, (0, 0, 0, 1) can be solved with an AND Gate.

Since it occurs only very infrequently that we need to add only two bits together, the adder really needs to be able to accept and compute an additional input: the carry from a lesser significant bit. Since this adder does but half the job, it is called the Half-Adder.

ACTIVITY #1: HALF-ADDER

Given the following description, design such a circuit.

1. Accept two input bits: A and B
2. Mathematically add the inputs together
3. Yielding two outputs: the sum (S) and a carry (C).
4. Function to behave according to the truth table, in Figure 8-1.

Inputs		Outputs	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Figure 8-1
Desired truth table for the Half-Adder circuit.

At this point, the problem seems adequately defined. The next step is to break the problem down to sub-functions. Figure 8-1shows that the S output's function could be satisfied by using an XOR Gate.

Fitting an XOR Gate to the required function of S, yields the schematic on Figure 8-2.

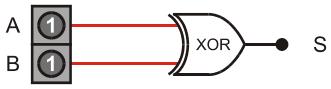

Figure 8-2: The XOR half of the Half-Adder schematic - handles the summing of A and B.

At this point a quick mental check of the circuit reveals that indeed the circuit designed thus far will satisfy the requirements of the sum (S) output. Figure 8-1 also shows that the carry output function (C) could be satisfied by using an AND Gate. Fitting an AND to the required function of (C) yields the schematic in Figure 8-3.

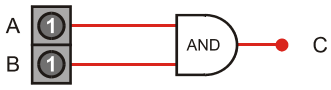

Figure 8-3: The AND half of the Half-Adder schematic - handles the carry aspect.

A verification of the functionality of the AND Gate for the C function confirms that the correct gate has been chosen. Since there are no more sub-functions to design, and the sub-functions have been verified (somewhat) for proper operation, it is time to combine the sub-functions into one function. The combined schematic is in Figure 8-4.

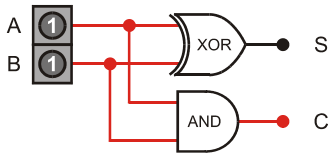

Figure 8-4:
The Half-Adder schematic.

Exercises

1. With the PDT de-energized, remove the wires applied for previous exercises then build the Half-Adder circuit. Remember to connect the outputs S and C to nearby LEDs so you can easily read their states.
2. Confirm your wiring then power-up the PDT. Confirm that your Half-Adder's performance matches that of the truth table listed in Figure 8-1.

Since the Half-Adder is only doing half the job, it is necessary to double the effort and create a more useful circuit: the Full Adder. The Full-Adder functions as the Half-Adder does but can also consider a carry as an additional input. Therefore, eight Full-Adders can be daisy-chained together so that an entire byte (8-bits) can be added to another entire byte concurrently.

ACTIVITY #2: FULL-ADDER

Given the following description, design such a circuit.

1. Accept three input bits: A, B, and an input Carry (C)
2. Mathematically add all of the inputs together
3. Yielding two outputs: the sum (S) and a carry (C).
4. Function to behave according to the truth table on Figure 8-5.

Inputs			Outputs	
A	B	C	S	C
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

Figure 8-5
Desired truth table for the Full-Adder circuit.

Despite the fact that the problem is clearly defined, no immediate solutions appear while studying Figure 8-5. The next step would be to break the problem down to sub-functions, but that would be difficult at this stage. For that reason, a tool borrowed from algebra will be used to help organize the data before continuing. Specifically, the substitution method can be quite helpful in paring down the amount of data that must be considered.

One thing that can be gleaned from studying this table is the fact that the outputs of the Full-Adder behave exactly like those of the Half-Adder when the input carry = 0. From this it can be inferred that the Full-Adder is comprised of a Half-Adder and something else. Following that assumption allows us to replace the A and B inputs of the Full-Adder's truth table with the correlating output of the Half-Adder. This substitution yields the truth table on Figure 8-6:

Inputs		Outputs	
S	C	S	C
0	0	0	0
1	0	1	0
1	0	1	0
0	0	0	1
0	1	1	0
1	1	0	1
1	1	0	1
0	1	1	1

Figure 8-6
Truth table for the Full-Adder circuit reduced by means of the substitution method.

The inputs A and B are no longer shown. In their place is input S which represents the sum of A and B (after the presumed Half-Adder). Now the truth table is a bit easier to read and furthermore, the possible solutions begin to reveal themselves.

It appears that the output sum is the XOR product of the input sum and the input carry, as in Figure 8-7

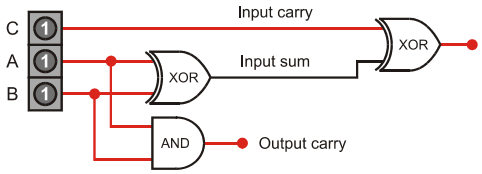

Figure 8-7:
The XOR of the input sum and the input carry. Partial derivation of the Full-Adder.

Recall that the Half-Adder will produce a carry if both inputs are 1. Considering this, and the data within Figure 8-6, the nature of the C output can be described as follows: if the Half-Adder produces a carry or if a carry is produced by ANDing input S and input C, the output C will be set. This implies two operations. First is the AND function regarding input S and input C, as shown in Figure 8-8

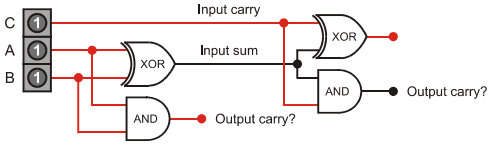

Figure 8-8:
The AND function of the input sum and the input carry. Partial derivation of the Full-Adder depicting the possible carry outputs.

The second operation implied by the aforementioned statement is that these two possible output carries are OR'ed together, as in Figure 8-9

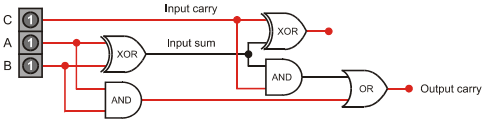

Figure 8-9:
The two output carries are OR'ed together. Final derivation of the Full-Adder.

Exercises

1. With the PDT de-energized, add the necessary connections build the Full-Adder circuit. Remember to connect the outputs S and C to nearby LEDs so you can easily read their states.
2. Double check your wiring then power-up the PDT. Confirm that your Full-Adder's performance matches that of the truth table listed in Figure 8-6.

Extra for Experts

1. Build two Full Adders on the PDT and connect them such that two 2-bit numbers are added together.
2. Create a truth table that describes the operation of the 2-bit Full-Adder.

Shift Registers

A register is a group of storage elements that are read and/or written to as a unit. There are many different types of registers, typically distinguished by size and function. This activity is concerned with the 8-bit shift register. A shift register is a register whose elements can exchange their states (and therefore their information). Generally this exchange of information occurs in only one direction. Consider the shift register made from eight flip-flops in the drawing in Figure 8-10

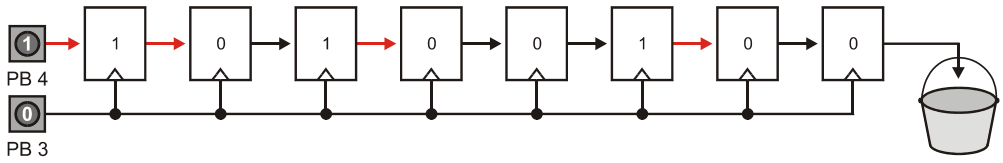

Figure 8-10: Generalized shift register - arbitrary initial state.

This simplified rendition of a shift register omits the reset inputs (for clarity) and shows all of the clock inputs connected together along with a pushbutton. When a shift register powers-up, some of the outputs flip and others flop. There is no rhyme or reason to it, it is quite arbitrary and may not even be consistent. The initial value of each stage of the shift register (each individual flip-flop) is represented within the stage with either a 1 or a 0. As long as the clock input is steady, the shift register will not change. If the clock was cycled once, in Figure 8-11 is what would happen.

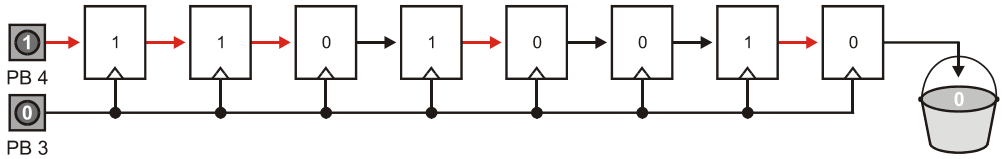

Figure 8-11: Generalized shift register - after one clock cycle.

When the clock goes high, (this clock is an active-high input), each stage will sample its input (input is on the left of each block), and update its output. The net result is that each stage will shift its data one stage to the right. The right-most shift register's contents are lost (depicted here with the proverbial bit-bucket) and the left-most shift register's input now holds the state of the pushbutton input.

ACTIVITY #3: SHIFT REGISTER

Construct an 8-bit shift register on the PDT using Flip-Flops E-L. Follow the schematic in Figure 8-12.

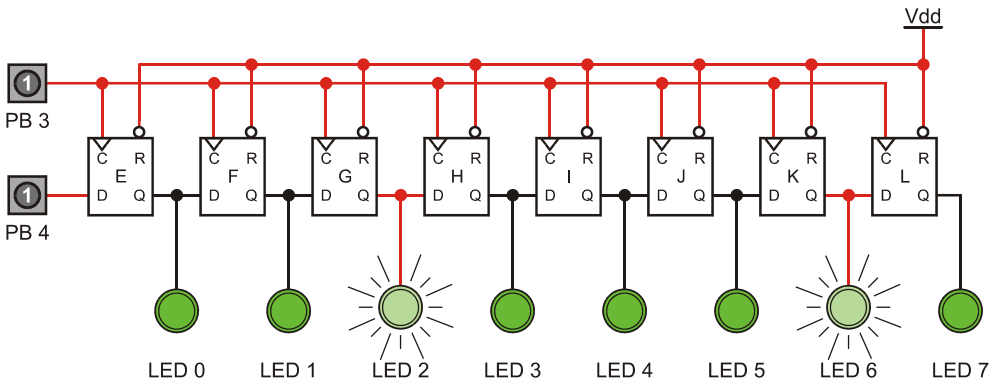

Figure 8-12: Shift Register - 8-bit

Exercises

1. Confirm your circuit is wired properly, then apply power to the PDT and note the state of the LEDs.
2. Push PB3 repeatedly (at least eight times). Describe the behavior of the shift register.
3. While PB4 is held, push PB3 at least eight times. Describe the behavior of the shift register.
4. What happened to the data that is shifted into the "bit bucket"? Is it retrievable?
5. Describe the pushbutton sequence necessary to write, "00001111" to the shift register.
6. Describe the pushbutton sequence necessary to write, "01010101" to the shift register.

Extra for Experts

1. Add the necessary components that cause the shift register to clear itself automatically when the number "99" (in decimal) is loaded into it.

Design Optimization

For a variety of reasons it becomes necessary from time to time to use one type of gate as another. Sometimes this is possible, sometimes not. For instance, you can use a NOR Gate as an inverter by tying one input low (as was done in previous exercises). Why would this be necessary in the real world? The answer may surprise you.

In the real world when you finish a design and that design works perfectly, that is the moment when the hard work just begins. After all, any two-bit crack-pot-wire-twister can slap a few ICs together and make a cell phone that works, but in order for that phone to be competitive and survive in the marketplace, it needs to offer a bazillion functions, play video games, cost next to nothing, and be so small that customers really want it, even after they lose it and must buy another one. That's where the work begins.

A good engineer will squeeze every iota of functionality out of each and every IC and gate within the design (within practical limits). Also, many circuits will do "double-duty", (performing different functions at various times) which takes a great deal of forethought and planning. Employing these practices will cut down on the number of parts required to make your product, and therefore reduce the cost of production, increase the product's functionality, and increase the likelihood of that product becoming a successful competitor in the marketplace.

Within the scope of this course, we will be exploring a few alternative ways in which we can utilize logic elements. In software, the practice of using one type of variable as another type is referred to as casting. In the movies, a person named George could be cast as a character named Everett; he is still George, but is simply emulating Everett for the interim. Therefore, it also seems appropriate that we apply the same term here.

ACTIVITY #4: CASTING LOGIC

Many integrated circuits (ICs) in the logic family are actually clusters of functions. For instance, U3 of the PDT is one IC that is comprised of four NAND Gates. If your design required three NAND Gates and one inverter, you could use three of the NAND Gates as usual, and cast the last NAND Gate as the inverter. This approach has many benefits: eliminate the need to for a separate NOT Gate, make the design less expensive, and the overall design would be physically smaller.

Design optimization often necessitates the casting of logic, but for those of you who go on to IC design, you will find it absolutely necessary for a surprising reason: there are no AND Gates and no OR Gates, only NAND, NOR, and NOT Gates are available at the silicon level. AND an OR gates are really just mathematical functions. So, get used to casting logic, you will most likely use it in the future.

Coincidentally, the easiest place to start here is with the inverter, or NOT Gate. Since the NAND Gate has within itself a NOT Gate, it should be easy to view the truth table and configure the gate to emulate an inverter, as shown in Figure 8-13.

Input A	Input B	Output Y
0	0	1
0	1	1
1	0	1
1	1	0

Figure 8-13
Truth table for the NAND Gate.

Using the above truth table, design and implement a circuit that casts a NAND Gate as a NOT Gate. The answer is in the truth table. If input A is tied low, output Y will always be high, regardless of the state of input B. However, if input A is tied high instead of high, output Y will vary inversely with input B. This is what is desired. The schematic for this is given in Figure 8-14

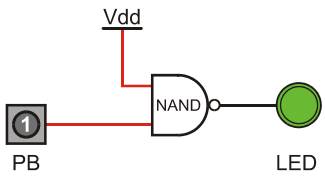

Figure 8-14
The NAND Gate, cast to function as an inverter (NOT Gate).

Feel free to wire this circuit on the PDT and try it out if you wish. A similar function can be achieved with the NOR Gate. First, revisit the truth table for the NOR Gate, in Figure 8-15.

Input A	Input B	Output Y
0	0	1
0	1	0
1	0	0
1	1	0

Figure 8-15
Truth table for the NOR Gate.

If the A input was tied high, the output Y would remain low, regardless of the state of input B. However, if input A was tied low, output Y would vary inversely with input B, which is the operation desired. The schematic in Figure 8-16, depicts the NOR Gate cast as a NOT Gate

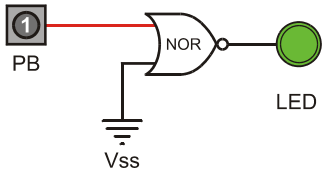

Figure 8-16
The NOR Gate, cast to function as an inverter.

Exercises

1. Create and test a circuit that uses an XOR Gate cast as a NOT Gate.
2. What other ways can you configure an XOR Gate to act as an inverter?
3. Create and test a circuit that uses an XNOR Gate cast as a NOT Gate.
4. What other ways can you configure an XNOR Gate to act as an inverter?
5. Create and test a circuit that uses a MUX cast as a NOT Gate.
6. What other ways can you configure a MUX Gate to act as an inverter?
7. Create and test a circuit that uses a Flip-Flop cast as a NOT Gate.
8. What other ways can you configure a Flip-Flop Gate to act as an inverter?

Extra for Experts

1. Can you cast a Flip-Flop to act as an XOR Gate?
2. Cast a D Flip-Flop into a T Flip-Flop. (A T Flip-Flop Toggles its state every time an input pulse is received.)
3. Can you cast an XOR Gate into an OR Gate?
4. Cast an XOR Gate to be a buffer. (A buffer is like an inverter, but performs no inversion; essentially, a YES Gate.)

ACTIVITY #5: SHORT COUNTS

Counters are required in many applications. The bit-width of a counter defines the maximum number it may hold. A two bit counter can hold any number 0-3 inclusive. A 3-bit counter can hold any number between 0-7 inclusive. Similarly, a 4-bit counter can hold any number between 0-15 inclusive.

In fact, there is a formula that gives the maximum number that can be held, given the bit width of the counter.

n = 2x - 1 where **n** is the maximum number that the binary counter can hold, and **x** is the bit width of the binary counter.

Running a few numbers through this equation yields some peculiar numbers: 3, 7, 15, 31, 63, 127, etc. Many tasks required do not need counts of those increments but need counts like 5, 10, 100; after all, designs require the use of binary numbers to accommodate a decimal based consumer. That necessitates the use of short counts.

A short count is a configuration in which a counter is wired to reset itself after receiving a number of counts that is less than its maximum. By applying a little logic in the right spot, a counter's count can be adjusted to any value less than its maximum. For example, if a counter were required to count from 0 to 9, then reset to 0 on the next pulse, the reset logic would reset the counter when the counter output equaled 10 decimal or1010 binary. The right tool for the right job.

First, describe the problem:

Given four inputs, and one output, make the output true when the four inputs are 1010.

Further define the problem:

When Q4 AND Q2 are 1, AND Q3 AND Q1 are 0, then the output is 1.

Translate to Boolean expressions:

When (Q4 AND Q2) AND (Q3 AND Q1)', then output is 1.

Simplify using the laws and axioms of Boolean algebra:

...after applying DeMorgan's Theorem on the second term...
(Q4 AND Q2) AND (Q3 NOR Q1) = 1

Implement the logic:

As in Figure 8-17

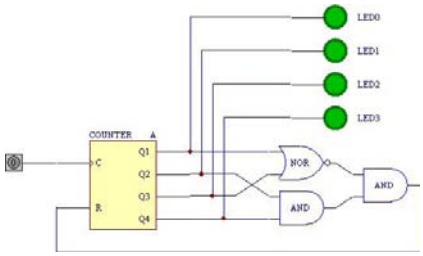

Figure 8-17
A counter configured to count short from 0 to 9.

Exercises

1. Design and implement a counter that counts short to 11 decimal.
2. Design and implement a counter that counts short to 13 decimal.

Extra for Experts

1. Design a counter than counts from 0 - 6, then counts back down to 0.
2. Design and implement a counter that counts by twos.

Chapter Review Questions

1. Describe the process of adding two binary digits together.
2. Discuss the differences between the "carry in" and "carry out" bits?
3. What is a register?
4. What is a shift register?
5. What is meant by a short count?
6. What is the right tool for the right job?
7. How does one calculate the largest number that can be held in a counter?
8. Does this same equation hold true for registers?
9. What is meant by "casting" logic?
10. Why would one want to cast logic?

Chapter #9: Logic Projects

This section is quite different than its predecessors in that, after each project is designed and implemented using hardware, the project will then be designed and implemented using a software approach with the BASIC Stamp 2. Additionally, you will then be asked to compare and contrast the pros and cons of each approach. Specifically, each activity is split into three or four sub-components: the "A" section is the familiar hardware approach, the "B" section is the new software approach, the "C" section is the critique on both previous approaches.

This unique approach will illuminate the strong and weak points of each discipline. Additionally, each activity may be concluded with a hybrid project (section "D") involving both hardware and software working together - a powerful combination.

If learned, these skills will serve you well. Often, it is not a matter of *if* we can achieve something, but *how* a particular task should be achieved. It is the implementation that decides the projects limits, cost, ability to be upgraded, etc. So, now that you know the what, let's examine the how by doing a few interesting projects.

ACTIVITIES IN THIS CHAPTER

1. Activity 1: LFSR (Random Number Generator)
2. Activity 2: 7-Segment LED Decoder
3. Activity 3: Traffic Light Controller

By the end of this chapter, you should be able to:

1. Compute the largest number that can be held by a register or a counter.
2. Understand the nature of random numbers
3. Design and implement solutions for simple problems
4. Write simple programs using the Stamp
5. Identify the strong and weak points of hardware and software.

Parts Required

The parts used in this chapter's activities are:

(1) Parallax Digital Trainer (PDT)
(1) 9 Volt DC power supply with a 2.1mm coaxial power plug (center positive).
(1) Wire kit (Consists of three different colored rolls of 24 gauge solid wire.)
(1) Pair of wire strippers (not included in PDT kit).
(2) Red LEDs
(2) Yellow LEDs
(2) Green LEDs
(6) 470 Ohm resistors

Random Numbers

The first rule about random numbers in computers is: there is no such thing. That is because that all numbers in computers are *pseudo-random*, and are not truly random.

Pseudo-random numbers are actually a sequence of numbers generated by either a list or a numerical calculation.

Truly random numbers are just that: truly random, with no rhyme or reason as to their nature. The perfect random number sequences never repeat their patterns, ever. One good example of this is are the digits of PI. PI is a constant ratio that is used in circular equations. Example: The area of a circle is equal to the diameter of that circle multiplied by PI. If you were to very accurately measure a circle's area and diameter, you could calculate PI. Doing so would reveal that the division would never end. One might wonder why something so irrational would be needed within a digital binary system. The answer is easy, but perhaps not obvious: the answer is that human beings who use these computers are irrational, and for that reason desire some element of irrationality in the computer based systems that they use.

Consider the average home CD player. It has a random function that is nice to use from time to time. After all, playing the songs in a different order sometimes sounds more interesting. Does the random play function playback the songs in the same order every time? Of course not. It could hardly be referred to as random at all in that case. So, how then does the random function work on one's CD player, or in a video poker machine for

that matter? It certainly appears to be truly random. By exploring the nature of random numbers and the implementation thereof in computer-based systems, you will learn it all.

ACTIVITY #1A: LFSR-HARDWARE

The type of random number generator that will be explored is the linear feedback shift register, or LFSR. An LFSR may be used whenever an element of randomness is needed and security is not an issue. I mention the security issue because random numbers are one means of encryption used to transceive secure messages and data. Simple LFSR generated sequences of numbers are easy to reverse engineer.

At the heart of every LFSR is a shift register, much like the one explored in Chapter 8. The shift register used in Chapter 8 merely shifted the data from right to left without affecting the data at all. The pushbutton was the input device that fed the shift register. Also recall that the shift register automatically assumed some value when it was first powered up. The same thing would happen here as well. The difference is the input device: instead of a pushbutton, we generate the input signal with an XOR Gate. The output of the XOR Gate will be the input of the shift register. This XOR Gate functions as a device called the tap. The tap, in this case, is on bit 6 and bit 7. Study Figure 9-1.

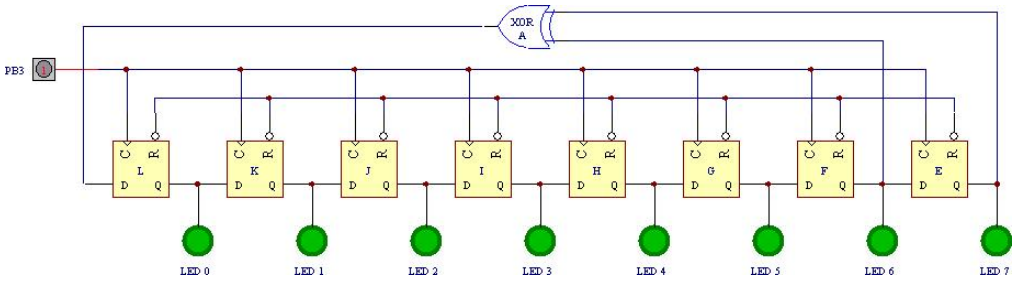

Figure 9-1: Simplified Linear Feedback Shift Register.

Data is shifted through the shift register with every cycle of the clock. The data present in bits 6 and 7 are fed to an XOR Gate. The XOR Gate functions predictably and its output is fed to bit 0 of the LFSR.

Exercises

1. Build the simplified LFSR. Verify your wiring before powering up the unit.
2. Cycle the clock input, describe the behavior of the LEDs.
3. Does the cycle ever repeat? If so, how often?

Remove the tie to Vdd on the reset line and instead connect the reset bus to PB4. Check your additions against the schematic in Figure 9-2 before powering-up and proceeding.

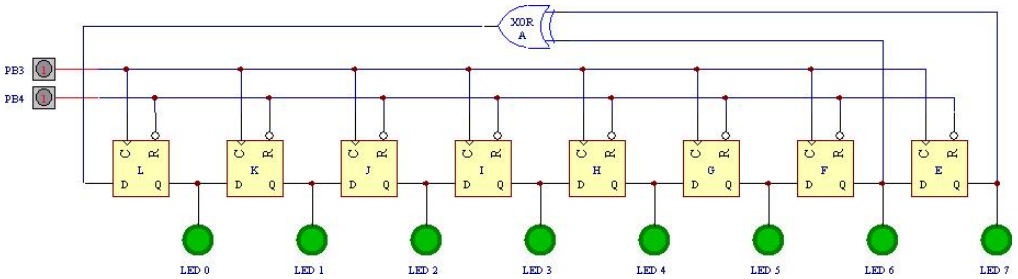

Figure 9-2: Single Tap LFSR with the reset input connected.

4. Clear the LFSR by pushing PB4. Cycle the clock input, describe the behavior of the LEDs.
5. Does the LFSR ever restart? Describe why or why not?

Taps

The characteristics of any LFSR are governed by two things: the bit-width of the LFSR, and the number and placement of the taps. Generally there are "good spots" for taps and there are "bad spots" for taps. "Good spots" are tap configurations that produce numeric sequences that can run for a long time without repeating themselves. Conversely, the "bad spots" are tap configurations that either have relatively short repeat periods, or that "short circuit" and die out instead of continuing on. Calculating exactly the nature of each tap combination is a bit beyond the scope of this course, but if you use only one or two taps, you can usually find a "good spot". The mechanism of every tap does not necessarily have to be an XOR Gate either. It could be just about any function.

6. Using one tap, change the tap location to find the combination that yields the shortest sequence. What is this "bad spot" and how many cycles did it last before repeating or dying out?
7. Using one tap, change the tap location to find the combination that yields the longest sequence. What is this "good spot" and how many cycles did it last before repeating itself.
8. Repeat steps 6 and 7 with an XNOR Gate for the tap function instead of the XOR Gate.

The BASIC Stamp

If you have never worked with the BASIC Stamp before, you are encouraged to read Appendix B: PBASIC Quick Reference Guide. This appendix serves as both a primer and a reference for PBASIC issues within this text. You may download the complete BASIC Stamp manual free of charge from: www.parallax.com.

While working through this chapter, you will be asked to write small PBASIC programs. Generally, you will be given a program as an example, and you will have to modify it to attain the desired results. Every instruction that you are likely to use is explained in Appendix B. Pending your instructor's approval, you are certainly free to use other PBASIC commands. In that case, be sure to download the BASIC Stamp Manual from Parallax's website.

Hello World

The best first program to write is the proverbial "Hello World" program. This program is small and simple, and getting it to work proves the software is installed properly and all is well with your PC, the PDT, and your stamp.

Launch the BASIC Stamp Editor program. Ensure that the PDT is connected to your PC via a serial cable and the BASIC Stamp (a/k/a Stamp) is plugged into its socket aboard the PDT before powering up the PDT. Within the IDE (integrated development environment) write the following:

```
' Elements of Digital Logic - EODL910.bs2        Program Listing 9-1-0
' Proverbial first program.
' {$STAMP BS2}
' {$PBASIC 2.5}

  DEBUG "Hello World!"
  END
```

Lines that are preceded with an apostrophe are comments. Each stamp program should start with a couple comments to tell what the program is called and what it does. Some comments direct the IDE; they are called directives. The directive tells the IDE what kind of Stamp to look for and what version of Stamp Editor to use.

Directive: a text string, denoted as a comment, that controls a function within the IDE.

The first line of code uses the Stamp's internal debugger to send a message to the monitor of the PC. Run this program by hitting "Ctrl-R", or clicking Run->Run, or clicking on the green triangle on the toolbar. Note: when you run a program, it is automatically compiled[5] first. If all is well you should see a new window, the Debug Window, pop up and display your greeting, "Hello World".

Just to be certain that its working, change the text message within the double quotes, and run the new program. Note that any text within the double quotation marks will be displayed.

If the desired results are not achieved, please ask your instructor or lab assistant for help. If the difficulty persists, please call Parallax Technical Support: (916) 624-8333.
Now, on to the fun stuff.

ACTIVITY #1B: LFSR-SOFTWARE

The first step to make a random number generator is to create a shift register. This is quite easy to do with a Stamp. Simply type:

```
' Elements of Digital Logic - EODL911.bs2        Program Listing 9-1-1
' Creating an alias for a variable.
' {$STAMP BS2}
' {$PBASIC 2.5}

shiftReg         VAR    Word                     ' Alias "shiftReg" is
                                                 ' assigned to the 1st
                                                 ' 16-bit register.
Main:
  END
```

[5] The Stamp software actually "tokenizes" the user program, (reduces the user program to its smallest possible size). In this text, the words compile and tokenize (and their derivations) are used interchangeably.

This statement after the comments and directives is a declaration. It doesn't really "do" anything. However, it does associate the first register of memory with the name "shiftReg". Within our program, anytime we need to access this byte of memory, we can do so with its alias, "shiftReg". Bear in mind that simply naming the register "shiftReg" does not make it a shift register. Naming registers with useful names helps keep the program readily understandable. The nice thing about the registers within the Stamp is that all of the bits within a register are already tacitly connected together. In fact, the Stamp has built in commands that perform the shift for us.

```
' Elements of Digital Logic - EODL912.bs2        Program Listing 9-1-2
' Using a variable in an expression.
' {$STAMP BS2}
' {$PBASIC 2.5}
shiftReg         VAR    Word                     ' Alias "shiftReg" is
                                                 ' assigned to the 1st
                                                 ' 16-bit register.
Main:
  shiftReg = shiftReg * 2                        ' Shift the data left
  END                                            ' one position.
```

The multiply function is invoked by the "*", while the assignment function is invoked by the "=" sign. Multiplying a binary number by two causes all of the bits to shift to the left one place. Conversely, dividing a binary number by two causes all the bits to shift to the right. To confirm that the data is shifting as it should, a few numbers will be run through this program. The Stamp's on-board debugger will confirm or deny the fact that it is operating properly.

The Stamp automatically clears all registers to 0 when it powers up. So, an initial value is needed to seed the shiftReg register. Modify your code to reflect the program in Listing 9-1-3.

```
' Elements of Digital Logic - EODL913.bs2        Program Listing 9-1-3
' Initializing a variable.
' {$STAMP BS2}
' {$PBASIC 2.5}

shiftReg         VAR    Word                     ' Alias "shiftReg" is
                                                 ' assigned to the 1st
                                                 ' 16-bit register.
Init:
  shiftReg = 1                                   ' Seed shiftReg with
                                                 ' a value of 1 first.
Main:
  shiftReg = shiftReg * 2                        ' Shift the data left
  END                                            ' one position.
```

If this program were to run, shiftReg is loaded with the literal value "1", and then it is multiplied by a constant 2. The assignment operator places the resulting value into shiftReg. Adding a few more instructions can really bring this example to life.

```
' Elements of Digital Logic - EODL914.bs2        Program Listing 9-1-4
' A simple shift register.
' {$STAMP BS2}
' {$PBASIC 2.5}

shiftReg         VAR    Word                     ' Alias "shiftReg" is
                                                 ' assigned to the 1st
                                                 ' 16-bit register.
Init:
  shiftReg = 1                                   ' Seed shiftReg with
                                                 ' a value of 1 first.
Main:
  shiftReg = shiftReg * 2                        ' Shift the data left
                                                 ' one position.
  DEBUG BIN8 shiftReg, CR                        ' Show shiftReg (bin)
  PAUSE 1000                                     ' Wait 1 second
  GOTO Main                                      ' Repeat forever
  END
```

By adding a debug command, a delay, and the necessary commands so that the program loops forever, you can visibly see the shifting bits

Exercises

1. Carefully type the code in Listing 9-1-4 into the Stamp IDE, (Stampw.exe)
2. Once entered correctly, click on the run icon (triangle).
3. Record the contents of the debug window and describe the behavior of the shift register.
4. Change the seed value that is initially loaded into shiftReg, run the program and note the output.

The next step is to convert the shift register into an LFSR. Since we wish to mimic the hardware as closely as possible, examining it reveals that the tap action really counts only immediately after the shift action does. Knowing this, we can simply add one line of code to effect the feedback for the LFSR.

```
' Elements of Digital Logic - EODL915.bs2        Program Listing 9-1-5
' Simple LFSR.
' {$STAMP BS2}
' {$PBASIC 2.5}

shiftReg         VAR    Word                     ' Alias "shiftReg" is
                                                 ' assigned to the 1st
```

```
                                                ' 16-bit register.
Init:
  shiftReg = 1                                  ' Seed shiftReg with
                                                ' a value of 1 first.
Main:
  shiftReg = shiftReg * 2                       ' Shift the data left
                                                ' one position.
  shiftReg.BIT0 = shiftReg.BIT6 ^ shiftReg.BIT7  ' Bit6 and bit7 are
                                                ' the tap inputs, and
                                                ' bit0 is the output.
  DEBUG BIN8 shiftReg, CR                       ' Show shiftReg (bin)
  PAUSE 1000                                    ' Wait 1 second
  GOTO Main                                     ' Repeat forever
  END
```

The added line of code simply XORs bits 6 and 7 of the shift register, and places the result in bit0 of the shift register, just as it was done in Activity 1A, Figure 9-2.

5. Modify your code to reflect the new line of code in Listing 9-1-5.
6. Run the program and note the output.
7. Note the first few output values, count the number of iterations the LFSR goes through before the cycle repeats itself.

ACTIVITY #1C: LFSR-CRITIQUE

Now that you've completed the same experiment in two mediums, you can compare and contrast the pros and cons of each approach. By critiquing the two approaches, the strong and weak points of each solution will become apparent.

Exercises

For the following exercises, please limit the scope of your answers to exclude the time you spent learning, but to simply include the time planning, working, and debugging.

1. Describe how would you rate the relative difficulty of each approach?
2. How much time did you spend individually on Activity 1A and Activity 1B?
3. Once the designs were done, how would you rate the relative ease in which design and parameter changes were made?
4. How capable is each method of approach? Is one approach a better fit for this particular task?
5. Given the Stamp costs roughly ten times more than that of the 74 series ICs used on the PDT, which would you prefer to use to create a product that used an LFSR?

6. Which approach do you think works best overall? Why?

ACTIVITY #1D: LFSR-HYBRID

What good is a PC without any software to run it? What good is the latest game without a nice and fast PC is run it on? The fact is that hardware and software need each other. Furthermore, hardware and software must work *with* each other. That means that each must utilize their own strengths and help compensate for the weaknesses of the other.

You've probably concluded that software is really the way to go if you are going to make an LFSR. For this last activity, both hardware and software will join forces to create a much more interesting experiment. Coupling software to the hardware will add a dimension of flexibility that would be difficult to achieve otherwise.

Exercises

1. On the PDT, build an 8-bit LFSR using Flip-Flops E-L (L is the LSB) by connecting the Q output of L to the D input of K, etc.
2. Connect the Q outputs of the Flip-Flops to the adjacent LEDs (0-7) and to P0 through P7 respectively on the X19 jumper wire socket strip.
3. Add an XOR tap connected to Flip-Flops E and F (on the Q outputs).
4. Connect Flop-Flop L's clock and reset line to P8 and P9 respectively on jumper wire socket strip X17. (Actually, you are connecting the clock and reset line for all eight Flip-Flops when you do this.)
5. Connect P10 to an input of OR E. Connect OR E's other input to the output of the XOR tap.
6. Connect the output of OR E to the Data input of Flip-Flop L.
7. Connect Flip-Flop L's clock to LED 8.
8. Connect P15 on the X19 jumper wire socket strip to the speaker.
9. Enter the code in program listing 9-1-6 into the Stamp Windows Editor.

```
' Elements of Digital Logic - EODL916.bs2        Program Listing 9-1-6
' Simple LFSR - Hybrid design.
' {$STAMP BS2}
' {$PBASIC 2.5}

Clk              PIN    8                       ' Define aliases for
Rst              PIN    9                       ' I/O pins used.
Seed             PIN    10

tally            VAR    Word                    ' Counts the number
                                                ' of iterations
Init:
```

```
  DIRC = %0111                                    ' P11 = input, P8-10
                                                  ' are outputs
  OUTC = %0010                                    ' Set Rst high, Clk
                                                  ' and seed low.
Main:
  HIGH Clk                                        ' Make LFSR shift
                                                  ' Show the count and
  DEBUG "Count:",DEC4 tally," LFSR:",DEC4 INL,CR  ' the LFSR contents
  LOW Clk                                         ' Hold the LFSR idle
  LOW Seed                                        ' Turn off seed bit
  FREQOUT 15,250,INL*13+350                       ' Calculate f in the
                                                  ' audible range.
  tally = tally + 1                               ' Increment the count
  IF INL = 0 THEN HIGH Seed                       ' If shiftReg stalls
  GOTO Main                                       ' then seed it.
  END                                             ' Repeat forever.
```

If all is well, power up the PDT and run the program. The LFSR should be working together producing a range of random tunes. You will need to know a few things about how the program works before you can complete this activity. The comments within the program are adequate for what they are, but some general knowledge about the concepts employed could help clarify the operation of the program.

Every time the Clk goes high, the LFSR shifts. As mentioned before, an LFSR can hit a bad spot and stall. If the output of the LFSR ever is 0, the Seed is brought high and places a 1 on the data pin of Flip-Flop L. This action "seeds" the LFSR with a non-zero value. The output of the Seed is mixed with the output of the XOR tap courtesy of OR E. LED 8 should blink each time the LFSR is shifted.

The elaborate expression for the freqout command simply maps the output of the LFSR (which is 0-255) to a range that is within the speaker's performance (~350Hz to 3.6 kHz). The 250 number in the freqout expression controls the duration each frequency is played.

At the rate of 1/4 second per tone, it takes a while before the sequence repeats. In fact, it takes so long that it may be easier to see when the LFSR repeats rather than by listening. By reducing this rate, the repeat time can be reduced to fit within the "attention span" of the human ear.

10. Slowly, progressively, reduce the tone duration until your ear can detect the point when the LFSR repeats.
11. Restart the LFSR by pushing the Reset button. Does the LFSR always start up the same way? Explain why that is.

12. Systematically change the tap gate to each different kind of gate. Use the debugger window to view the Tally register to discern how many times the LFSR shifts before it repeats. Tabulate all of the answers
13. Which gate(s) produced the longest repeat sequence?
14. Which gate(s) produced the shortest repeat sequence?
15. Using the gate that produced the longest sequence, move the tap position and note the lengths of the repeat sequences.
16. Using the gate that produced the longest sequence, and the tap configuration that produced the longest sequence, how many iterations does the LFSR go through before repeating itself?
17. Add a second tap gate (any kind) and configure it to lengthen the repeat sequence. How many iterations did it go before repeating?

Extra for Experts

1. Add a pushbutton input to the Stamp and amend the program so that a new random number is generated every time the button is pushed.
2. In the breadboard area, add six LEDs arranged as 2 columns of three. Connect these LEDs to Stamp I/O pins[6].
3. Using the Stamp, limit the output of the LFSR such that the numbers 1 through 6 are randomly generated. Map this output to the LEDs so that they resemble an electronic die. Roll the die and show it off to friends. If you get this done, you're ready for an internship at Parallax! Good job!

ACTIVITY #2A: LED DECODER-HARDWARE

Displays are everywhere. They constitute half of virtually every user interface in the world. There are many types of displays: CRT (like your monitor), VFD (vacuum fluorescent display), LCD (liquid crystal display), LED, and still others. The easiest one to build a driver for is the LED.

Driver is a common term used to refer to sub-system, (can be made of hardware, software, or both), that makes the use of a device (like an LED display) easier.

[6] The Stamp can supply enough current to drive 5 LEDs safely. The sixth LED will have to be driven indirectly using a gate. If you use a gate that inverts the signal, remember to invert that signal in your program as well.

Within each genus of display, there are many species. The LED display that will be used in the following activities is the lowest level. This implies that there is no "smart" driver on-board. Therefore, each element of the display must be controlled discreetly. Figure 9-3 depicts the segment names and the pin definitions of the display.

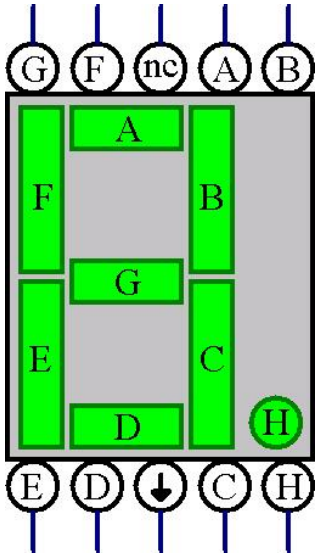

Figure 9-3
7-Segment LED Display

Segment designators and pin locations shown.

Recall that a discreet LED has two legs (wires). Previously, we referred to them as simply the positive side or the negative side. The more correct terms for these are anode and cathode respectively. This distinction is necessary to make at this time in order to accurately describe and understand this LED display. LED displays like this one come in two flavors: common anode and common cathode. It is important to know which type of display you have in advance, so that you may wire it up properly. Figure 9-4 shows the common cathode variety, which is the type that you will be using.

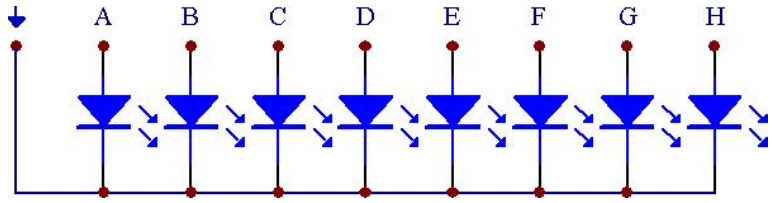

Figure 9-4: Common Cathode 7-Segment LED Display Schematic

As the name implies, all of the cathodes are tied together, in common. Since the cathode is the negative side of the LED, it makes sense to connect it to ground. The remaining connections will be tied, *via 470 Ohm resistors*, to the circuitry that will control them.

WARNING: All of the LEDs on the PDT have current limit resistors integrated into the LEDs, which has allowed you to forego using external resistors. The 7-Segement LED Display has no such resistors, so you MUST add a 470 Ohm resistor in series with each segment that will be energized. Please follow the schematics carefully to avoid damaging the PDT and your 7-Segment LED Display.

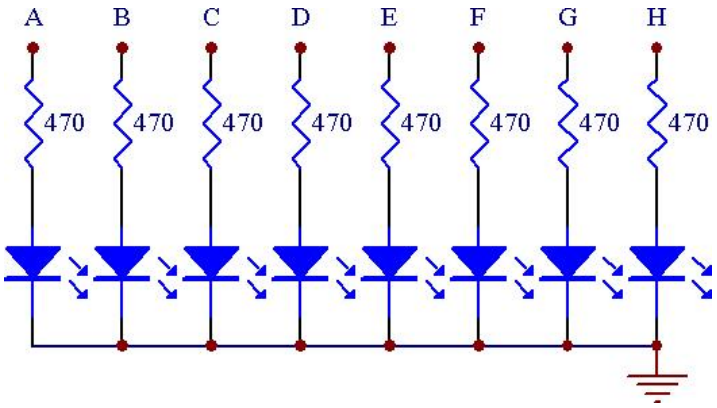

Figure 9-5: Common Cathode 7-Segment Display - partial schematic

Figure 9-5 depicts the common cathode display with its series resistors, ready to be connected to the decoder circuit.

The decoder will be designed using combinational logic techniques, since there are no transitional inputs required - the state of the outputs rely solely on the static state of the inputs. *The right tool for the right job.* That's right! Boolean algebra will guide us through the design process. The first step is to thoroughly describe the problem.

Based on two inputs, drive a 7-Segment LED to reflect the decimal equivalent of the binary value present on the inputs. There are so many variables and parameters, a table would be a good way to correlate the data.

Inputs		Segments							Num
B	A	G	F	E	D	C	B	A	
0	0	0	1	1	1	1	1	1	0
0	1	0	0	0	0	1	1	0	1
1	0	1	0	1	1	0	1	1	2
1	1	1	0	0	1	1	1	1	3

Figure 9-6: Input/Output correlation table for the 7-Segment LED Display[7]

In this case, it is a very good way since this table very succinctly describes the behavior of each segment for all cases of the two inputs. From Figure 9-6, we can extract one correlation for each segment, then use Boolean algebra to define the logic required. Example:

Segment B is ON regardless of the inputs, therefore:

B = 1. (on all of the time)

Segment G is ON whenever input B is ON, therefore:

G = B

Two segments (albeit the easiest ones) are now represented with a Boolean expression. One more example will be done for you, the F segment.

Example:

Segment F is ON only when both A and B are OFF, therefore:

F = (A + B)' (this would be satisfied by a NOR Gate.)

Now, it's your turn.

Exercises

1. Derive Boolean expressions for the remaining segments, (forego the H segment, since the decimal point segment will not be used).
2. For all seven segments, design the logic necessary on paper. Present this to your instructor for review and approval prior to proceeding.
3. Implement your logic design. Use PB3 and PB4 for the two inputs. Use NOR A and NOR B to invert PB4 and PB3 respectively.

[7] The H segment (the decimal point segment) has been omitted since it will not be used here.

4. After checking your wiring, power up the PDT and confirm its proper operation. Should you encounter problems, troubleshoot each segment individually. Do not proceed until this project works properly.

Extra for Experts

1. Modify the existing design to support one extra number so that it can display 0-4.

ACTIVITY #2B: LED DECODER-SOFTWARE

Coding an LED Decoder in software starts off much the same way that it did in hardware. The problem needs to be clearly stated, then defined. So, again, the correlation table of Figure 9-7:

Inputs		Segments							Num
B	A	G	F	E	D	C	B	A	
0	0	0	1	1	1	1	1	1	0
0	1	0	0	0	0	1	1	0	1
1	0	1	0	1	1	0	1	1	2
1	1	1	0	0	1	1	1	1	3

Figure 9-7: Input/Output correlation table for the 7-Segment LED Display[8]

The Stamp has built-in support for look-up tables. Each element in the table must be one byte wide (8-bits wide). When this table was used to design the hardware, the data was grouped in a columnar fashion to accommodate logic gates. We could use the exact same approach in software, but software allows us to use data in larger groups, so we will group the data by row. To convert a 7-bit pattern into an 8-bit byte you must add what is called a "don't care" bit, usually denoted by an "X". The bit patterns then become:

Pattern to display number: 0 X0111111
Pattern to display number: 1 X0000110
Pattern to display number: 2 X1011011
Pattern to display number: 3 X1001111

[8] The H segment (the decimal point segment) has been omitted since it will not be used here.

These patterns presume that you will connect the segment LEDs to certain I/O pins on the Stamp. Figure 9-8 depicts this.

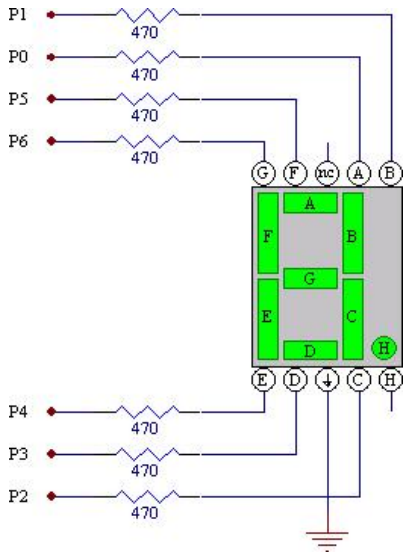

Figure 9-8
7-Segment LED Display connections to the Stamp.

Note: these LED segments require current limit resistors as shown..

9

Exercises

1. Remove the previous activity's wiring from the PDT.
2. Wire up the 7-Segment LED Display as shown in Figure 9-8.
3. Write a small program that confirms the proper operation of the circuit thus far. (Just use the HI and LOW commands to turn on and off the segments to confirm two things: that they work, and that the right I/O pin is connected to the correct segment.
4. Type in the program from Listing 9-2-1. Run it, and confirm that all is well.

```
' Elements of Digital Logic - EODL921.bs2        Program Listing 9-2-1
' Simple number generator.
' {$STAMP BS2}
' {$PBASIC 2.5}

Number          VAR         Nib                  ' Num is 4-bits wide

Main:
  FOR Number = 0 TO 3                            ' For each number
    DEBUG ?Number                                ' 0 - 3, show the
```

```
  PAUSE 1000                                    ' number, wait 1S.
NEXT                                            ' When done, wait
PAUSE 3000                                      ' 3S. then repeat.
GOTO Main
END
```

5. Using the LOOKUP function and the given patterns (to display number x), amend the code in Listing 9-2-1 to decode Num, and drive the 7-Segment LED Display accordingly.

Extra for Experts

1. As it stands, the program now counts and displays the numbers 0-3. Expand the functionality of the program to count and display the numbers 0-4.
2. Expand the program to count and display numbers 0-15. Note: you will have to program the unit to count using hexadecimal numbers.

ACTIVITY #2C: LED DECODER-CRITIQUE

You have solved a problem two different ways. Compare and contrast the pros and cons of each approach.

Exercises

For the following exercises, please limit the scope of your answers to exclude the time you spent learning, but to simply include the time planning, working, and debugging.

1. Describe how would you rate the relative difficulty of each approach?
2. How much time did you spend individually on Activity 2A and Activity 2B?
3. Once the designs were done, how would you rate the relative ease in which design and parameter changes were made?
4. How capable is each method of approach? Is one approach a better fit for this particular task?
5. Given the Stamp costs roughly ten times more than that of the 74 series ICs used on the PDT, which would you prefer to use to create a product that had to drive a 7-Segment LED display? What about a device that had three of these 7-Segment LED Displays to drive?
6. Which approach do you think works best overall? Why?

ACTIVITY #2D: LED_DECODER-HYBRID

The hybrid LED Decoder will employ a few items not previously covered simply because these add-ons will make the project more complete. First, you will need to remove all previously wired circuitry on the PDT - start fresh.

Exercises

1. Connect PB3 and PB4 to Stamp I/O pins P6 and P5 respectively.
2. Connect the outputs of Counter A to four LEDs (0 - 3)
3. Connect Counter A's reset input to P9 and its clock input to P8 and also the LED5.

```
' Elements of Digital Logic - EODL922.bs2        Program Listing 9-2-2
' Simple number generator.
' {$STAMP BS2}
' {$PBASIC 2.5}

i        VAR     Nib
OnOff    VAR     Bit

PB3      VAR     IN6
PB4      VAR     IN7

Init:
  DIRC = 3
Main:
  GOSUB ReadPBs
  IF OnOff = 1 THEN TOGGLE 8
  GOTO Main

ReadPBs:
  FOR i = 1 TO 10
    IF PB3 & PB4 THEN SkipIt
      OnOff = PB3 ^ PB4 & PB4
      i = 9
    SkipIt:
    PAUSE 25
  NEXT
  RETURN
```

4. Enter the program code from Listing 9-2-2
5. Write a detailed description of what this program is doing.
6. Comment your code to summarize the program operation.
7. Run the program in Listing 9-2-2. Test the operation and feel on PB3 and PB4. Did it work the way you thought it would?

8. Add the necessary logic to limit the count to 0 - 9. Verify the proper operation before continuing.
9. The outputs of the counter are currently connected to four LEDs. Now, connect them to Stamp I/O pins P0 - P3 as well.
10. Modify the your program to read P0 - P3, decode those inputs into 7-Segment LED bit patterns (for digits 0-9), and (using series resistors) drive the 7-Segment LED Display.
11. Amend your program to support hexadecimal digits 0-F. Remove the count limit circuitry. Test the program, verify your binary to hexadecimal converter works.

ACTIVITY #3A: TLC-HARDWARE

All that you have learned thus far will be required to bring the Traffic Light Controller project to fruition. In addition, one concept that has been only flirted with previously will be brought into the spotlight: the finite state machine.

Finite State Machines

One of the most important types of circuits in digital logic is the finite state machine, or FSM. The FSM is so named because its logic and output(s) can only be in a fixed number of possible states. The FSM is a sequential logic circuit comprised of combinational logic whose next state relies on the current state of the logic and the state of the next input. A counter is an example of a simple state machine. Simple in that the next state of a counter always the same, i.e. 4 always follows 3, etc. In an FSM, the choice of the next state depends on the value of the input, and therefore the FSM can exhibit more complex behavior.

A simple FSM is the Traffic Light Controller or TLC. The TLC that you will work with has been simplified. It contains two red lights, two green lights, two yellow lights, and two "demand" buttons. Each set (red, yellow, green) of LEDs constitutes a signal. Each signal controls traffic on a one-way street. These two signals work together to control traffic at the intersection of two one-way streets. Figure 9-9 depicts the intersection.

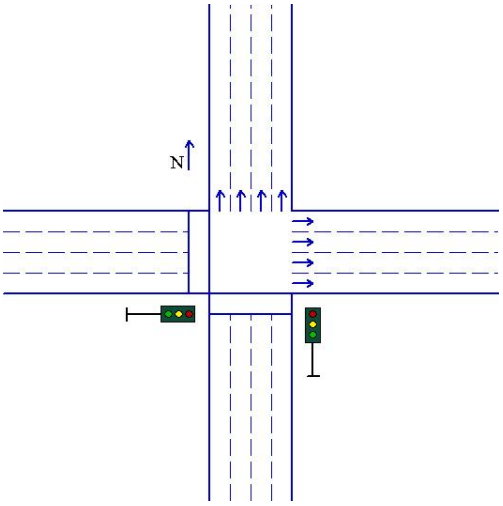

Figure 9-9
Intersection of two one-way streets.

It is recommended that you group the LEDs by signal so that, when they operate, they look familiar and resemble actual traffic lights, Figure 9-10. Doing so will make it easier to determine whether or not your circuit is operating properly.

Figure 9-10
Discreet LED with current limit resistor in series.

Warning! When using discreet LEDs, you MUST use a current limit resistor in series with each LED as shown in Figure 9-11. It doesn't matter whether the resistor is on the anode or the cathode side; it will work equally well either way. Figure 9-11 shows one way to wire the LEDs.

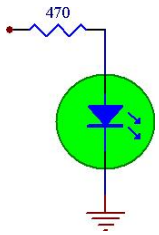

Figure 9-11
Discreet LED with current limit resistor in series.

Exercises

1. Identify the four possible states that the traffic lights could possibly be in.
2. Based on these four states, design a decoder using discreet gates on the PDT to drive the LEDs accordingly. *(Use the right tool for the right job and show your work.)*
3. Implement and test decoder using two pushbuttons as the inputs to test all four possible states.

Extra for Experts

1. Add the control logic to read the demand inputs and sequence the lights through their cycles. Use a pushbutton to simulate a clock so you can control the rate at which the circuit switches by hand. Set the counter to count three cycles on the yellow states.

ACTIVITY #3B: TLC-SOFTWARE

Exercises

1. Disturbing your existing circuit as little as possible, disconnect the LEDs from the hardware decoder and re-connect them to the Stamp's I/O pins of your choice. (Remember to use the current limit resistors - they're still needed)
2. Design and code a software based state machine that, based on the inputs of two pushbuttons, cycles the LEDs through the same sequence as activity 3A.

ACTIVITY #3C: TLC-CRITIQUE

You have solved a problem two different ways. Compare and contrast the pros and cons of each approach.

Exercises

1. Describe how would you rate the relative difficulty of each approach?
2. How much time did you spend individually on Activity 3A and Activity 3B?
3. Once the designs were done, how would you rate the relative ease in which design and parameter changes were made?
4. How capable is each method of approach? Is one approach a better fit for this particular task?
5. Given the Stamp costs roughly ten times more than that of the 74 series ICs used on the PDT, which would you prefer to use to create a product that had to drive a 7-Segment LED display? What about a device that had three of these 7-Segment LED Displays to drive?

ACTIVITY #3D: TLC-HYBRID

1. Add the control code necessary to read the demand inputs and sequence the lights through their cycles. Use a pause command to set the time that the yellow cycles run.
2. Remove the state machine from your program and connect the output of your software control logic to the input of your hardware state machine.

Extra for Experts

1. Use the two extra push-buttons, four LEDs on the PDT, any additional decode gates, and the necessary code to support the WALK/DON'T WALK pedestrian crossing demands and displays.

Chapter Review Questions

1. What is the largest number a 10-bit register can hold? Show your work.
2. Within computers, are random numbers truly random? Explain.
3. Explain each step used when designing a solution in a digital binary system.
4. Explain how one pushbutton can be used for both on switch and an off switch in the same program program.
5. Explain how to use the LOOKUP function.
6. Compile a list of the advantages that hardware has over software.
7. Compile a list of the advantages that software has over hardware.

A

Appendix A: Ohm's Law

This text presumes that electrons flow, as opposed to another theory called "Hole Flow". In a general sense, electricity is the flow of electrons possessing the capacity to do work. A battery is a device that converts chemical energy into electrical energy. Given a path, electrons flow from the negative lead of a battery to the positive lead. Electricity is not like water that can jump from the end of the hose; electricity is bound to the wires, so it requires a path.

Current refers to the *flow* of electrons. The amount of current can be quantified; the unit of measurement is called the Ampere (or Amp) and its symbol is A. Voltage is the *pressure* that causes the electrons to flow. This pressure can be quantified; the unit of measurement is called the Volt and its symbol is V. Electrical components that resist the flow of electricity are called resistors. The amount of resistance of a resistor is quantifiable; the unit of measurement is called the Ohm and its symbol is Ω (pronounced Ohm). Resistors are available in many different resistance values and are designated with color bands to denote their values.

There is a mathematical relationship between Volts, Ohms, and Amps called Ohm's Law given below:

Ohm's Law $E = IR$

Where E is voltage in volts, I is current in Amps, and R is resistance in Ohms.
Using simple algebra, you can solve for any variable.

$$I = V/R, R = V/I, V = IR$$

Example:
If you place 5 Volts across a 5 Ω resister, 1 Amp of current will flow. 1 = 5/5.

Given that the numbers frequently used in electronics range from the very small to the very large, prefixes are employed to help keep the physical size of the numbers

reasonable. For large numbers, the prefixes Mega (M) and Kilo (K) are used. A 1M Ω resistor is actually 1,000,000 Ω. A 4.7K Ω resistor is actually 4,700 Ω. For small numbers, milli (m) and micro (μ - pronounced as mew) are used. A current of 1 mA is really 0.001 Amps. A current of 1 μA is really 0.000001 Amps. Now you are ready to calculate a value depicted in Figure A-1.

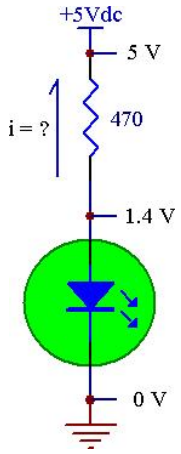

Figure A-1
Voltage and current analysis of a resistor and LED in a simple DC circuit.

Given that an LED and a resistor are in series, and that 5 VDC is applied across the circuit as shown, calculate the amount of current that flows through the circuit using Ohm's Law.

$$I = V/R$$

First, realize that the amount of current flowing through the LED must be the same as the current flowing through the resistor. We don't know what the resistance is of the LED, but we do know that the resistor is 470 Ohms. Since our LED (as do most others) requires 1.4 VDC to operate, we can take the difference to find the amount of voltage felt across the resistor. 5V - 1.4V = 3.6V. So now the voltage and the resistance for one component in a series circuit are known, so the current can be calculated thusly:

$$I = 3.6V/470\ \Omega = 0.00766 \text{ Amps (or 7.6 mA)}$$

LEDs typically require 10-15 mA to attain full brightness. If 15mA were required to flow through this circuit, what size resistor would you use?

Note: Stamp I/O pins can supply 20mA of current per pin (max of 40mA total of all pins). Never choose a resistor that will allow more than 20mA to flow, lest you fry your stamp. Actually, the Stamp's voltage regulator would limit the amount of current flowing through its regulator to 50mA. If the circuit should attempt to draw more current than that, then the Stamp's regulator would go into "current limit" mode and shut itself down. Once it cooled off, it would then allow current to flow. If excessive current would start to flow again, the stamp would shutdown again. The result is a pulsing circuit.

The amount of power can be calculated with either of the following formulae:

$$P = IE \quad \text{or} \quad P = I^2R$$

P is power in Watts, I is current in Amps, E is voltage is Volts, R is resistance in Ohms. The resistors supplied with the PDT are 1/4Watt resistors. Using the above formula, we can calculate the maximum amount of power allowed through these 470 ohm resistors: 0.25 W = I x 3.6V. After a applying a little algebra: I = 0.25/3.6 = about 69mA. Since the LEDs should never be subjected to more than about 30mA, these 1/4W resistors should be just fine.

Appendix B: PBASIC Primer & Reference Guide

Note: this is a very brief overview. If you cannot find the answer within this appendix, please download the BASIC Stamp Manual from our website: www.parallax.com.

The BASIC Stamp (Stamp for short) is a miniature single board programmable computer. It is called the BASIC Stamp because it is programmed with a language that is very similar to BASIC that we call PBASIC (short for Parallax BASIC), and because is it very small – about the size of a postage stamp.

The Stamp is comprised of a microcontroller, user memory, and all the other circuitry a microcontroller requires for proper operation. Collectively, the Stamp is a single board computer. The microcontroller is pre-programmed at the factory to fetch, decode, and execute PBASIC tokens from the user memory. The user memory is an EEPROM (electrically erasable, programmable, read only memory), so it may be reprogrammed over and over again.

Parallax has created software for the Stamp called Stampw.exe. This program looks and works like a typical word processor. It allows users to write PBASIC programs, tokenize them, (a type of compilation) and download the resulting tokens via a serial cable to the Stamp. Once programmed, the Stamp simply fetches and executes every instruction as long as power is applied. The Stamp can then be disconnected from the PC and run completely independently.

The Stamp executes the instructions, one at a time, starting at the top of the program. Consider the small PBASIC program below.

```
Start:
  PAUSE 500          ' Wait 500 mSec. (1/2 second).
  HIGH 0             ' Make P0 an output and drive it high.
  PAUSE 500          ' Wait 500 mSec. (1/2 second).
  LOW 0              ' Make P0 an output and drive it low.
GOTO Start           ' Return to the top and repeat forever.
```

The left-most column is where "labels" are placed. Labels are addresses, they flag a particular location within the program. The central column contains the instructions of the program. The right-most column contain the comments, demarked by the apostrophe. Comments merely explain what the program is doing and have absolutely no effect on the program itself. The Stamp syntax is case insensitive.

The Stamp interfaces with the outside world via its I/O pins (I/O = Input or Output). The I/O pins are labeled P0 through P15. External access to these pins is provided courtesy of X17 and X19 on the PDT.

There is one pushbutton that has a fixed function, and as a result, a fixed name: Reset. Pushing this pushbutton causes the Stamp to reset and start running the program from the beginning.

The PBASIC language is comprised of both high-level and low-level commands. Low-level commands perform very simple tasks such as switching on or off and I/O pin. High-level commands perform more impressive tasks such as generating sound. One of the most useful high-level commands is the **DEBUG** command. The **DEBUG** command echoes information within the Stamp to a **DEBUG** window within the Stamp software so that we may view what is going on inside the Stamp while it is running.

Variables

The Stamp's architecture is based on 16-bit integers. Specifically, a "Word" is 16-bits wide. Other bit width variables are available: A "Byte" is 8-bits wide, a "Nib" is 4-bits wide, and a "Bit", by definition, is 1-bit wide. The various bit widths are there to allow the user to minimize RAM usage as RAM memory is the most precious resource on the Stamp.

Aliases

An alias is simply a more convenient name for a variable. The Stamp software allows for declaring variables with the **VAR** directive. Declare your variables near the top of the program. Examples below:

```
' {$STAMP BS2}                ' Listing B-2-1
' {$PBASIC 2.5}               ' PBASIC version 2.5

lucky        VAR    Nib
tabby        VAR    Word
cat          VAR    Byte
```

I/O Pins

The Stamp has 16 I/O pins that are completely under user control. The names of the individual I/O pins are P0 through P15. Given the aforementioned bit widths allowed in

the Stamp's architecture, the I/O pins may be addressed individually, and in groups of various bit widths. When addressing I/O pins as groups, it is easiest to use their predefined names despite the fact that you must consider this when assigning I/O pin functions at design time.

There are three dedicated 16-bit registers associated with I/O pin function. The register `OUTS` is written to when any pin that is an output is to be changed. The register `INS` is read when the state of the I/O pins needs to be read. The register DIRS is written to when the direction (i.e. input or output) of the I/O pins needs to be changed. All three of these registers are 16-bits wide. All three have constituent registers that allow you to write to groups of I/O pins smaller than that of the 16 total. The drawing in Figure: B-2 shows all of predefined names associated with the I/O pins.

		Predefined Names associated with I/O pins											
		Output registers				Input registers				Direction registers			
Names	16-bit	8-bit	4-bit	1-bit	16-bit	8-bit	4-bit	1-bit	16-bit	8-bit	4-bit	1-bit	
P0	OUTS	OUTL	OUTA	OUT0	INS	INL	INA	IN0	DIRS	DIRL	DIRA	DIR0	
P1				OUT1				IN1				DIR1	
P2				OUT2				IN2				DIR2	
P3				OUT3				IN3				DIR3	
P4			OUTB	OUT4			INB	IN4			DIRB	DIR4	
P5				OUT5				IN5				DIR5	
P6				OUT6				IN6				DIR6	
P7				OUT7				IN7				DIR7	
P8		OUTH	OUTC	OUT8		IN8	INC	IN8		DIRH	DIRC	DIR8	
P9				OUT9				IN9				DIR8	
P10				OUT10				IN10				DIR10	
P11				OUT11				IN11				DIR11	
P12			OUTD	OUT12			IND	IN12			DIRD	DIR12	
P13				OUT13				IN13				DIR13	
P14				OUT14				IN14				DIR14	
P15				OUT15				IN15				DIR15	

Figure: B-2

The drawing in Figure B-3, shows how a typical (P0 in it this case) is organized.

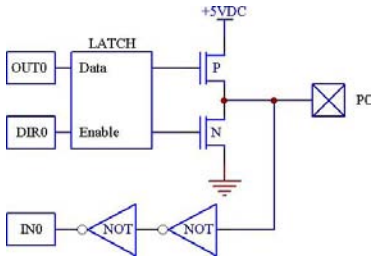

Figure B-3
Construction of P0.

Note that you can always read the I/O pin whether it's an input or an output.

Note: All I/O pins are inputs (not latched) upon power up.

Math Operations

The Stamp supports the following math operators and operations: **+** addition, **-** subtraction, ***** multiplication, **/** division, **<<** shift left, **>>** shift right, **&** logical AND, | logical OR, ^ logical XOR, **=** assignment.

Commands

Below you will find a description of the commands you are likely to use while working through this text.

```
DEBUG "The temperature is: ", DEC3 temp," degrees C", CR
```

This command sends a message to a discreet window on the PC. The Stamp Editor program must be running and a debug window must be open and enabled. You may send both fixed text and variable data to the debug window with the debug command. Any text within the double quotes will be printed literally in the debug window. Variables will be cast as directed (i.e. **BIN**, **HEX**, **DEC**) to the number of places specified. There are a few predefined constants that you may find useful when using debug:

SYMBOL	VALUE	ACTION
CLS	0	clears the debug window
HOME	1	Cursor to the top-left corner of the debug window
BELL	7	Beeps the PC speaker
BKSP	8	Cursor left one space
TAB	9	Tab to the next column (multiples of 8 spaces)
CR	13	Carriage return and line-feed

```
HIGH 0        ' Make P0 an output and drive it high.
```

This command makes the pin specified, (in this case P0), an output if it was not an output already, and drives the pin high, (5VDC).


```
LOW 0         ' Make P0 an output and drive it low.
```

This command makes the pin specified, (in this case P0), an output if it was not an output already, and drives the pin low, (0 VDC).

```
TOGGLE 1      ' Change the state of P1
```

The toggle command inverts the state of the specified pin, and also writes to the specified "dir" register, thereby making it an output.

```
PAUSE 2000    ' Delay program execution for #of mS specified
FREQOUT 0, 1000, 800, 1200
```

The **FREQOUT** command plays one tone, or two tones simultaneously, on the selected I/O pin, for the duration specified. In this example, P0 is the pin, 1000 mS (or one second) is the duration, and the two tones are 800 Hertz and 1200 Hertz.

GOTO *address*

The **GOTO** command vectors the program to a new address or location. The address must be specified by the user. The **GOTO** command is used to loop the program, or skip over code, instead of executing it. Example below:

```
Start               ' Address label
  TOGGLE LED        ' Blink the LED
  PAUSE 100         ' at 0.2 second intervals
GOTO Start          ' Go to the beginning and repeat forever.
```

GOSUB *address*

RETURN

The **GOSUB** command works like the **GOTO** command in that it vectors the program to a new address. The difference is that when a **GOSUB** instruction is executed, the location of the **GOSUB** is memorized. Upon executing a **RETURN**, the program recalls that memorized

address and returns to the very next instruction after the **GOSUB**. The **GOSUB** and **RETURN** instructions are generally used together to effect subroutines. Example below:

```
counter       VAR    Byte   'Declare a variable named Counter

DO
  GOSUB Increment          ' Call the Increment subroutine
  DEBUG ?counter           ' Display the current Counter
LOOP                       ' Repeat forever

Increment                  ' Increment subroutine
  counter = counter + 1    ' Add one to Counter
RETURN                     ' and return
```

FOR *variable* = *expression* TO *expression*

NEXT

The **FOR - NEXT** instruction pair allows the user to code a loop for a specified number of iterations. The variable must be a RAM variable, while the expressions can be either constants or expressions. Example below:

```
j      VAR    Nib          ' Loop counter
TOP    CON    12           ' Top constant defined

DO
  FOR j = 0 TO TOP - 1     ' for all but 1 of array elements
    DEBUG ?j               ' print the element number
  NEST
LOOP                       ' Repeat forever.
```

IF *condition* THEN GOTO *label*

The **IF-THEN** instruction is one of the conditional branches allowed in PBASIC programming. At runtime, if the condition specified evaluates to be true, then the label specified will be vectored to. If the condition evaluates to be false, the label instruction will be ignored and instruction execution will continue with the next instruction following the **IF-THEN** instruction.

Bit Operations and Manipulation

PBASIC language allows bit-wise operations. Each bit position is given a predefined name that you may use to refer to specific bits at runtime. For instance, if you wanted to

move a data bit located in bit position 7 (of a byte size variable) into bit 3's location, you would use the following assignment:

```
temp          VAR    Byte
temp.BIT3 = temp.BIT7
```


Other operations of a similar type are permitted too. For example if you wanted to swap the two lower nibbles of two bytes, you could do so with the following assignments.

```
temp1         VAR    Byte
temp2         VAR    Byte
temp3         VAR    Nib
temp3 = temp1.LOWNIB
temp1.LOWNIB = temp2.LOWNIB
temp2.LOWNIB = temp3
```

Like the nibble operations mentioned, byte-wise operations are also permitted. The designators are: **HIGHBYTE**, and **LOWBYTE**.

LOOKUP ***index, (val0, val1, val2,…valn], result***

The **LOOKUP** command will, based on the value of the "index" parameter, select one element of a table of elements and load it into the "result" parameter. If the index register held a value of 0, then val0 would be loaded into the register result. This command is good for correlating numbers when there is no easy way to calculate one number based on another.

Appendix C: Boolean Laws

Boolean Rules

A x 1 = A
A x 0 = 0
A x A = A
A x A' = 0

A + 1 = 1
A + 0 = A
A + A = A
A + A' = 1

Commutative Law

A x B = B x A
A + B = B + A

Distributive Law

A x (B + C) = (A x B) + (A x C)
A + (B x C) = (A + B) x (A + C)

Associative Law

(A x B) x C = A x (B x C) = A x B x C
(A + B) + C = A + (B + C) = A + B + C

DeMorgan's Theorem

(A x B)' = A' + B'
(A +B)' = A' x B'

Appendix D: Number Systems

Due to the advent of the computer, there are several number systems in use today. Chiefly, they are: binary, octal, decimal, and hexadecimal. For years it was case that using any number system other than the decimal number system, (our familiar number system), was left for mathematicians and other academics, or amputees. The decimal number system is easily understood by the human mind, but computers can handle numbers of other types of number systems much more efficiently.

Decimal

In the number system that we are all familiar with, (base 10), ten characters are used, hence the name decimal number system. The "dec" prefix means 10. These ten characters are ordered from the lowest magnitude, 0, to the greatest magnitude, 9, thusly: 0,1,2,3,4,5,6,7,8,9. To count, we increment the number in this certain order. It is implied that there is a 0 proceeding, (but not changing the value of), the digit, example: 09.

When we need to express the number that is one greater that 09, we restart the counting in the least digit 9 to 0, and increment the implied 0 to the next its next greater digit, 1. It may seem silly to point out the tedium involved with the way we count our ever so familiar decimal numbers, until you see that that is the key to understanding *any* number system.

One of the nice things about the decimal number system is how easy it is to multiply and divide - as long as the number is multiplied or divided by 10d. Even an untidy number like 81,370d is easily divided by 10d. Simply shift the decimal point to the left one digit, and presto: 8,137d. If you wish to divide the answer by 10d again, simply repeat the process, and presto: 813.7d. This is true because the decimal number system is based on ten characters, so operations dealing with the number 10d are usually quite easy. The converse is true for multiplying by ten.

Hexadecimal

Consider the base 16 number system we call Hexadecimal. It is called Hexadecimal because, instead of just ten characters used, there are 16. Like the decimal number system, the first ten characters are: 0,1,2,3,4,5,6,7,8,9. Six more characters we required to complete a hexadecimal number system, so we borrowed the first six characters of the alphabet: A,B,C,D,E,F. So, the sixteen characters that comprise the hexadecimal number

system ordered with regard to their respective magnitudes from least to greatest are: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Counting in hexadecimal starts exactly the same way as it does in decimal. The difference comes when we get to the digit 09. Since we have yet to run out of characters to use, we simply keep on counting. In decimal, the next number would be 10. In hexadecimal, the next character is 0A. This progression continues until we get to 0F, (which is equal to decimal 15), then we must restart the counting of the right-most digit from F to 0, and increment the implied 0 to a 1. The result is 10 (hex, short for hexadecimal) which is equal to 16 (dec, short for decimal)

One of the nice things about the hexadecimal number system is how easy it is to multiply and divide - as long as the number is multiplied or divided by 10h (16d). Even an untidy number like 3170h is easily divided by 10h (16d). Simply shift the decimal point to the left one digit and you get, 317h. If you wish to divide the answer by 10h (16d) again, simply repeat the process and presto: 31h. This is true because the hexadecimal number system is based on sixteen characters, so operations dealing with the number 10h (16d) are usually quite easy. The converse is true for multiplying by sixteen.

One of the other nice things about the hexadecimal number system is that math operations involving the number two are also quite easy.

Binary

In the binary number system, we have but two characters to use: 0, 1. This number system, like the others, utilizes the same algorithm as the other number systems. We start counting 0,1; then what? As with the other number systems, there is an implied 0 before the one: 01. Also, as with the other number systems, when we ran out of characters to use, we reset the right-most digit to 0 and incremented the implied 0 to a 1, thereby yielding 10b.

The strange thing about this 10b is that it is equal to 2 in decimal, not ten. Now that the right-most digit has been reset to 0, we can count one more higher to 11. Now what? Well, if we consider the number 99 in the decimal number system: how do we count one higher than 99? Answer: we reset the two right-most digits to 00 and increment the implied 0 to 1, which gives us 100. Apply this same technique to our binary 11, and when we increment 11 we get: 100b (coincidence not intended). Again, this number is not one - hundred as it would be in decimal, but it is "one-zero-zero" in binary which is equal to 4 in decimal. The converse is true for multiplying by two.

One of the nice things about the binary number system is how easy it is to multiply and divide - as long as the number is multiplied or divided by 10b (2d). Even an untidy number like 10101101b is easily divided by 10b (2d). Simply shift the decimal point to the left one digit and voila 1010110b. If you wish to divide the answer by 10b (2d) again, simply repeat the process and voila 101011b. This is true because the binary number system is based on two characters, so operations dealing with the number 10b (2d) are usually quite easy.

Converting

At some point you may need to convert numbers of one system to that of another. This is rather easy to do once the nature of number systems is understood. First consider the decimal number system, so named because it uses ten characters to count with. Another property of the decimal number system is that each digit's value is a power of 10. The ones digit has that value because it is the zero-eth digit $10^0 = 1$. The next digit to the left of the ones digit is the tens digit: $10^1 = 10$. Predictably, the next digit is the hundreds digit: $10^2 = 100$. The same is true for other number systems too, but the *base* changes to accommodate the number system.

In the binary system, $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$. So, in the binary system, each digit's value progresses exponentially from 1,2,4,8,16,32,64,128,256,512,… In the hexadecimal number system, $16^0 = 1$, $16^1 = 16$, $16^2 = 256$, $2^3 = 4096$, $2^4 = 65{,}535$. Notice the coincidence between binary and hexadecimal. 0,1,16,256, etc. This makes the transition from binary to hex and vice-versa quite easy.

Hexadecimal:	FE21	F	E	2	1
Binary:	1111111000100001	1111	1110	0010	0001

Simply isolate each hex digit and decode it to binary using the chart (at the end of this appendix) as if were a sole digit. Then concatenate the four digit clusters (respecting the order) to make a complete binary. The reverse is true when converting from binary to hex.

Binary:	1010100010001010	0101	0100	1000	1010
Hexadecimal:	548A	5	4	8	A

Converting from decimal to binary is pretty easy too. Start by finding the largest power of 2 that is smaller than or equal to the number you are working with.

Example: convert 73 decimal to binary, then into hex.

Since 64 is the largest power of two that is less than 73 and $64 = 2^6$, simply subtract 64 from 73 and place a 1 in digit 6 of the binary answer. Doing so leaves us with 9 and 01xxxxxx. The x digits are digits we have not yet calculated. Repeating the process, we find that 8 is now the largest power of 2 that is less than or equal to 9. Subtracting 8 from 9 leaves us with 1, and consequently since $8 = 2^3$, we must place a 1 in the 3rd digit. That leaves us with 01001xxx and 1. Since 1 is the largest power of two that is less than or equal to 1, and $1 = 2^0$, simply subtract 1 from 1, and place a 1 in the zero digit of the answer. Doing so leaves us with: 01001001 for the binary answer. You can easily convert this to hexadecimal by grouping it into 4 bit clusters and converting each cluster:
0100 1001 = 49h. So, 49h = 73d = 01001001b.

The following chart should help you with your conversions.

Binary	Hexadecmal	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Appendix E: 74' Series Mini-Datasheets

The following diagrams depict most of the logic ICs used on the PDT. The inputs will tolerate 0-5V signals (TTL levels), and higher voltage levels, like CMOS and HSC CMOS. For more information regarding other logic levels, consult your instructor or the manufacturer's data book.

The outputs are capable of driving 10 TTL loads or about 25mA each. If you need to drive more loads, simply run the signal through a couple of inverters so you have multiple chips driving your signal.

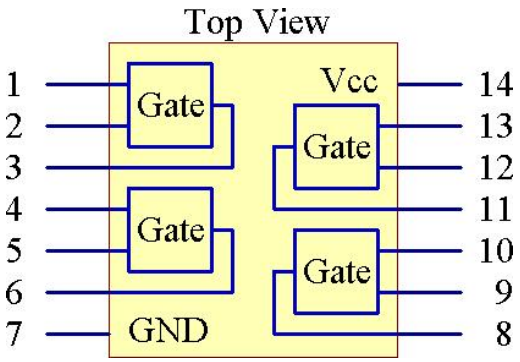

Figure E-1
Pin legend for the following ICs:
74C00 - Quad NAND Gate
74C08 - Quad AND Gate
74C32 - Quad OR Gate
74C86 - Quad XOR Gate

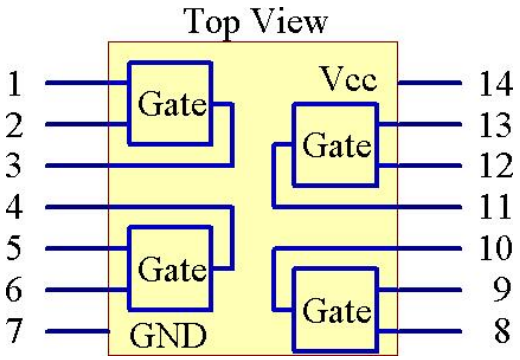

Figure E-2
Pin legend for the following ICs:
74C7266 - Quad XNOR Gate

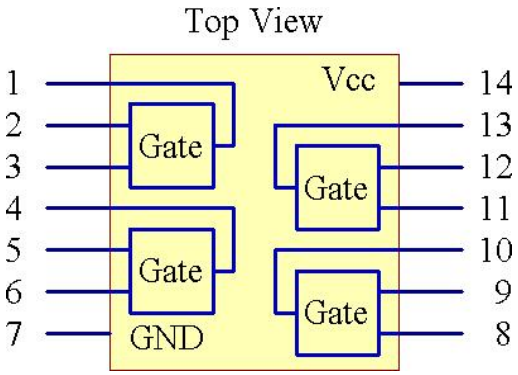

Figure E-3
Pin legend for the following ICs:
74C02 - Quad NOR Gate

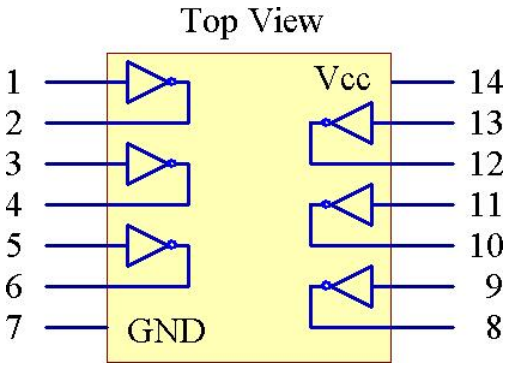

Figure E-4
Pin legend for the following ICs:
74C14 Hex Schmitt-Trigger Inverter

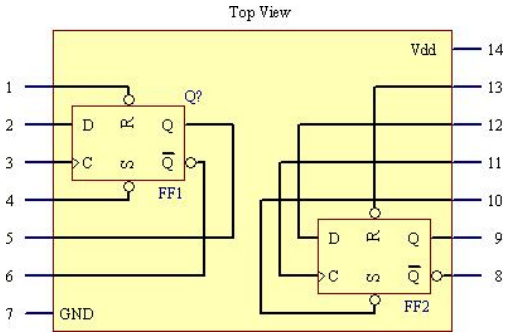

Figure E-5
Pin legend for the 74C74 Dual D Flip-Flop

Appendix F: Maintenance, Troubleshooting and Repair

Reasonable treatment and care should assure a long productive lifespan of your PDT. Since accidents and extraordinary events can and do occur, the PDT has been designed with ease of maintenance in mind.

On the back of every PDT is a list of the ICs used, correlated to the particular elements of the PDT that they are associated with. Troubleshooting the board is best accomplished by empirically testing each gate, one at a time. When a logic element is found to be non-functional, note the gate designator. Correlated the gate number on the IC Replacement chart on the back. The correlation should reveal an IC designator. Then, with the PDT powered down, replace that particular IC with a part of the same part number. Replacement parts will be least expensive if you obtain them from your own sources, but are conveniently available from Parallax, Inc.

The sockets used to wire-up circuits on the PDT were designed to use 24 gauge wire. 22 gauge wire will fit if forced a bit, but doing so will permanently distort the sockets. The distortion of the sockets is irreversible and will most likely reduce the useable lifespan of the PDT. Parallax, Inc. maintains large amounts of 24 gauge wire on-hand that are available at very reasonable rates. Please do not use 22 gauge wire on your PDT. Doing so will void the warrantee.

The breadboard used on the PDT possess very good quality contacts and is designed for heavy (frequent) usage. The trouble comes when a wire breaks an leaves a fragment in a socket. It can be removed with the point of an X-acto knife, but great care must be exercised since these knifes are incredibly sharp and dangerous, as any hobbyist knows.

For severely damaged breadboards, replacements can be obtained from Parallax for a modest fee. They are attached to the PDT with an adhesive tape that relinquishes its hold with some effort.

Catastrophic failures should be sent to Parallax for refurbishing. Please call customer support first, to obtain an RMA number prior to shipping.

Appendix G: Handling Static Sensitive Devices

Many electronic devices are sensitive to static electrical discharge. That is to that they may be damaged if subjected to a static discharge. Since the PDT contains static sensitive devices, (SSDs), it makes sense to discuss just how to handle SSDs to prevent such damage. First, a brief look at just what static energy is.

Static energy is the buildup of electrical charge in an object. For the purposes of this discussion, this buildup of charge usually results from friction between two dissimilar objects, like your shoes and the carpet. As you walk, sliding friction strips electrons away from one surface, (the carpet), and stores them on another, (you). The more you walk on the carpet, the more charge builds up on you. Eventually, you will touch a doorknob or a friend and ZAP! Why does this happen?

The expression, "opposites attract" applies here. In the previous paragraph, there was a large negative charge building up on you, and a large positive charge building up on the floor. These charges are opposite in polarity and naturally tend to be mutually attractive for the express purpose of neutralizing each other. When the opportunity presents itself, a spark will fly, electrons will flow, participating parties will yell, and thereby neutralize the charge. What opportunity presented itself? A path.

There is a whole industry devoted to the elimination of static energy. In manufacturing facilities, it is absolutely necessary. Since all this anti-stat equipment is rather pricey, you may not have it in your classroom or in your home. Assuming this is the case, there is little that can be done to prevent the generation of static energy, but there is a great deal that you can do to determine the path of the static discharge.

It all boils down to what you do immediately before picking up, putting down, or handing an SSD to a friend. Follow these simple precepts and your time spent reading this appendix will not be ill spent.

When picking up an SSD from a table. Touch the table first, then grab the device.

If you are carrying any charge, it will first be discharged to the table. Moreover, your fingers will be at the same electrostatic charge level as the SSD, so when you touch the SSD no current will flow through the SSD to the table.

When putting SSD down on a table. Touch the table first with your finger, then place the SSD on the table.

Again, doing this will neutralize any charge you may have on your person thereby protecting the SSD from the wrath of the path.

When handing an SSD to another person OR receiving one in a similar manner from a person. Touch that person's hand first, then grab the device.

Again, doing this will neutralize any charge that either of you may have, and therefore keep the SSD out of harm's way. By discharging the charge and diverting the path away from SSDs, you can handle SSDs in a reasonably safe manner.

Index