Basic Analog and Digital

Student Guide

VERSION 1.4

PARALLAX

INTERNET 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:

- <u>Propeller Chip</u> – This forum is for users of the multiprocessing Parallax Propeller chip.
- <u>BASIC Stamp</u> – This forum is widely utilized by engineers, hobbyists and students who share their BASIC Stamp projects and ask questions.
- <u>SX Microcontrollers and SX-Key</u> – Discussion of programming the SX microcontroller with Parallax assembly language SX – Key® tools and 3rd party BASIC and C compilers.
- <u>Stamps in Class</u>® – Created for educators and students, subscribers discuss the use of the Stamps in Class series in their courses. Students, educators and hobbyists are welcome to participate.
- <u>Javelin Stamp</u> – Discussion of application and design using the Javelin Stamp, a Parallax module that is programmed using a subset of Sun Microsystems' Java® programming language.
- <u>Robotics</u> – 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.
- <u>HYDRA</u> – A place for enthusiasts of the Propeller-based HYDRA game development system.
- <u>Parallax Educators</u> – A private forum exclusively for educators and those who contribute to the development of Stamps in Class. Parallax created this forum for educators to provide feed back and to obtain, develop, and share teaching materials.
-

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 personal computer brought in a whole new era of electronic sophistication. With it, we have immense amounts of digital computing power located right at our desk. Computers work well when they are connected to each other, and digital data can be transferred quite reliably from machine to machine.

However, the minute you wish to connect a digital computer to some "real world" device (such as a wind speed indicator or fuel level sensor) you need to design a circuit that interfaces an analog device to the digital computer. In many cases, this involves the conversion from an analog voltage to a digital representation of that voltage.

This set of Stamps in Class experiments will explore many of the basic principles of interfacing analog devices to digital microcontrollers. Many times this involves the use of easy-to-use commands built right into the BASIC Stamp, and at other times requires the use of a an "analog to digital converter".

Why should we be interested in converting from analog to digital? Many different aspects of our lives are dependent upon this conversion process. Some are not too critical to our survival like compact disc players, telephone systems, and music. Others, however, might be critical. Medical equipment and sensors often require analog to digital and digital to analog conversion.

The Basic Analog and Digital text will be revised and updated continually based on feedback from students and educators. Version 1.4 was edited to change from a 100 kΩ to a 10 kΩ potentiometer due to supply issues. Additionally, program listings PL3_1R0.bs2 and PL4_1R0.bs2 were updated to accommodate modification for several ADC0831 chips connected to the same CLK (clock) and D0 (data) lines. Pagination remains similar if not identical to Version 1.3.

AUDIENCE AND TEACHERS GUIDE

The audience for this text is ages 17 and above. This guide can be used as a complete book to introduce analog and digital concepts in a class, or as a reference to obtain detailed explanations about hardware and techniques used in other Stamps in Class Student Guides.

The answers to these experiments present no impossible or very difficult technical hurdles, and could be solved with a bit of patience. Instructors could participate in an Educator's Forum to obtain support or Teacher's Guides if they are available.

CONDITIONS OF DUPLICATION

Parallax grants the user a conditional right to download, duplicate, and distribute this text without Parallax's permission. This right is based on the following conditions: the text, or any portion thereof, may not be duplicated for commercial use; it may be duplicated only for educational purposes when used solely in conjunction with Parallax products, and the user may recover from the student only the cost of duplication.

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.

FOREIGN TRANSLATIONS

Parallax educational texts may be translated to other languages with our permission (e-mail stampsinclass@parallax.com). If you plan on doing any translations please contact us so we can provide the correctly-formatted MS Word documents, images, etc. We also maintain a discussion group for Parallax translators which you may join. This will ensure that you are kept current on our frequent text revisions.

SPECIAL CONTRIBUTORS

Version 1.0 of this text was written mostly by Andy Lindsay based on a manuscript submitted by Matt Gilliland, original author of *What's a Microcontroller?* and the ever-popular *Microcontroller Application Cookbooks.* Andy wrote this text during his senior year at California State University, Sacramento, where he studied Electrical and Electronic Engineering. This was his first of many Stamps in Class texts that he has revised and/or authored. Andy is also is a contributing author of several papers that address the topic of microcontrollers in pre-engineering curricula. When he's not writing educational material, Andy does product engineering for Parallax.

Chapter #1: Analog Voltage and Binary States

This series of experiments introduces analog and digital electronics. What does that mean? In *What's a Microcontroller?* we learned that analog is a "continuously variable value". Another way to think about it is that analog electronics is analogous to nature.

There are lots of continuously variable values in nature, such as motion, light level, and sound. The position of a door as it swings open is a good example of a continuously variable value. As a door swings from all the way closed to all the way open, it visits every value in between. At one instant during its travel, it is 1/3 of the way open. At another instant, it is 1/2 way open, and so on.

INTRODUCTION TO ANALOG AND DIGITAL

Digital simply means represented by digits. Think about how many times each day you encounter analog values that are represented with digits. The temperature is 79.8 degrees. The speed limit is 45 miles per hour, etc. Not surprisingly, digital electronics represents values with digits.

The term digital is also used when referring to binary devices such as the circuitry that makes a calculator work, the microprocessor in a computer, and the BASIC Stamp microcontroller. It's true - they are all digital devices. Binary devices are digital devices using two digits, 0 and 1.

The experiments in *What's a Microcontroller?* introduced a variety of techniques for interfacing with the outside world and other devices. These interfaces were mostly binary. This series of experiments extends the capabilities of interfacing by introducing several analog component interfaces and more component interface techniques.

In this first experiment, we'll build a circuit that produces an analog voltage at its output. Remember that analog voltage is continuously variable. The circuit will be adjustable so that it can produce an output anywhere between 0 and 5 volts. We'll also build a circuit called a voltage follower that uses this analog voltage to drive an LED circuit.

> **Volt/Voltage:** The volt is a fundamental unit of electrical measurement named after 18th century physicist Allesandro Volta, and a measurement in volts is referred to as voltage.
>
> Most of us encounter this unit of measurement when buying batteries such as the 9 volt (DC) battery that can be used to power the Board of Education or BASIC Stamp HomeWork Board. Inside a battery there are two chemical reactions, which are separated from each other by a barrier. One of the reactions creates a surplus or electrons and the other creates a shortage of them.
>
> The electron surplus and shortage sides of the barrier are connected to the negative and positive terminals of the battery respectively. If given a pathway around the barrier, the electrons have the potential to do work to get from the negative to the positive terminal. The volt is a measure of this potential to do work. The volt is also referred to as a unit of electric potential.

The analog voltage will also be connected to one of the BASIC Stamp I/O pins set to input mode. This binary input can actually be used to measure small variations in the analog voltage. PBASIC will be used to program the BASIC Stamp to drive a binary LED circuit, which will indicate when these variations have been detected.

The Debug Terminal is also a useful tool for displaying data the BASIC Stamp collects and sends. It will be used to monitor the binary value that the input pin receives as the analog voltage is varied.

Parts Required

For each experiment you will need a BASIC Stamp 2 microcontroller module on a Board of Education platform or a BASIC Stamp HomeWork Board. Your board needs to be connected to an IBM-compatible PC with Windows 2K/XP/Vista. You will need to install the BASIC Stamp Editor v 2.4 or higher, which is available for free download from www.parallax.com. (If you are using a USB board or a Parallax USB to Serial Adapter, you will need the FTDI USB VCP driver software installed; it installs automatically with the BASIC Stamp Editor v 2.4 or higher.) In addition, you'll need the following parts for this experiment:

(2) 470 Ω resistors
(2) Red LEDs
(1) 10 kΩ potentiometer
(6) Jumper wires
(1) LM358 op-amp

Throughout this series of experiments, we will build circuits from circuit schematics. One of the keys to learning how to read circuit schematics is learning what each symbol on the schematic means. It's also important to learn how to connect a part from the Analog and Digital Parts Kit to the prototyping area (also called a breadboard) of your Board of Education or HomeWork Board based on its circuit symbol in a schematic.

> **?** **Circuit Schematic:** Often referred to as a schematic, a circuit schematic is a map that uses symbols to show the components in a circuit and how they are connected. The components are represented by symbols such as the one that represents the LED in Figure 1-1.

Figure 1-1 shows the circuit symbol for an LED on the left and a drawing of an LED from the parts kit on the right. It also shows how the pins on an LED correspond to the terminals on the circuit symbol.

Flat spot on plastic part of LED indicating the cathode.
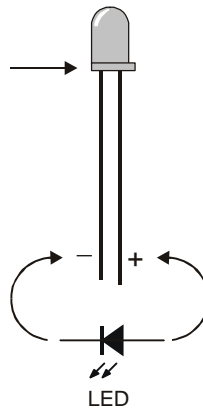
Figure 1-1
LED Circuit Symbol Compared to the Component

LED

Figure 1-2 shows a drawing of a resistor below its circuit symbol. The circuit symbol typically has the resistance value written below or next to it. The colored stripes on the part drawn below the symbol indicate its value, which is measured in ohms. The omega symbol (Ω) is used to denote the ohm. You can use Appendix B to convert the color codes on the resistor to resistance values.

470 Ω

Figure 1-2
Resistor Circuit Symbol and Corresponding Component

Current/Amp: Current happens when electrons travel from point A to point B. Direct current is what happens when you give the surplus electrons in the negative terminal of a battery a pathway to get to the positive terminal. The amp is the measurement of how many electrons per second are traveling through the pathway.

Resistance/Ohm: Resistance is a property of a material in the pathway the electrons travel through. The more difficult it is for the electrons to get from one end of the pathway to the other, the higher the resistance. A resistor is just such a pathway, and its resistance is measured in ohms (Ω).

Ohm's Law: When a resistor is used to provide a pathway between the negative and positive terminals of a battery, you have an electric circuit with voltage, resistance, and current. Ohm's Law relates the three quantities as follows:

V = I x R

V is the voltage measured in volts, I is the current measured in amps, and R is the resistance measured in ohms.

The Other Guys: Ever wonder where the words volt, amp, and ohm come from? They are all named after some of the people who made significant discoveries about electricity. We already know who the volt is named for; what about the other guys? The amp, also called the ampere, is named after 18th century physicist André Marie Ampère. The ohm is named after 19th century physicist Georg Simon Ohm.

The Potentiometer - A Source of Variable Voltage

The potentiometer (pot) has 3 pins on its underside that get plugged into the breadboard. On the topside, it has a knob you can twist to adjust it. In this experiment, we will use variable resistance to get a variable voltage output. Figure 1-3 shows how the pins on the underside of the pot from the parts kit correspond to the circuit symbol.

Pot
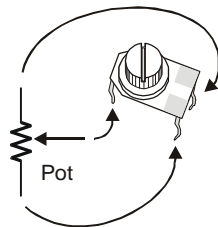
Figure 1-3
Potentiometer Circuit
Symbol and Component

Figure 1-4 shows what happens inside the pot as it is adjusted. The jagged line represents a resistive element, typically made of carbon. One end of the resistive element is wired to Vdd on your board, and the other end is wired to Vss. The middle of the three terminals is connected to the "wiper", and it's where the variable output voltage is measured. The wiper stays in contact with the carbon element as it moves.

As the wiper gets closer to Vdd, the voltage measured at the wiper terminal will approach the value of Vdd, which is 5 volts. Likewise, when the wiper is closer to Vss, the voltage at the wiper terminal will be closer to Vss, which is 0 volts. As the wiper terminal travels between Vdd and Vss, the output measured at the wiper terminal varies between these two values in a manner analogous to a door as it opens and closes.

Vdd

+

0-10 kΩ
Pot

‒

Vss
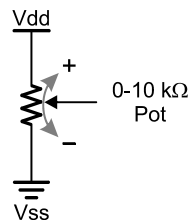
Figure 1-4
Potentiometer Wiper

*Showing how the wiper
in a potentiometer
travels along the surface
of the resistive element
as it's adjusted.*

The LM358 Op-amp

An op-amp (operational amplifier) is a building block commonly used in analog circuits. Figure 1-5 shows the circuit symbol and block diagram for the LM358 op-amp used in this experiment. The op-amp circuit used in this experiment is called a voltage follower because the same voltage comes out as goes in. In other words, the voltage at the output "follows" the voltage at the input. The reason it's used in the circuit in this experiment is to electrically separate a potentiometer circuit from an LED circuit. We'll learn more about the usefulness of a voltage follower in Chapter #4.

Figure 1-5
LM358 op-amp
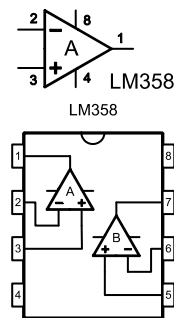
The circuit symbol has numbers on each of its terminals that correspond to the numbers on the block diagram.

The block diagram is a top-view of the part from your parts kit with the circuit symbols for the two op-amps in the part drawn in.

Make sure to note the location of pin 1 (top left) and the index mark when you place the LM358 on the breadboard. Improper wiring can damage an op-amp.

> **IMPORTANT:** Always disconnect your board's power source while you build or modify circuits.

The Board of Education and HomeWork Board Prototyping Area

Figure 1-6 shows the remaining circuit symbols used in the first experiment and where to find them on your board's prototyping area. The symbol for Vdd is the positive 5 volt supply for the BASIC Stamp and board. There are 4 sockets along the top side of the breadboard to the left for making connections to Vdd. Three sockets are for Vin, which is the direct voltage from your battery or power supply.

Next, the ground symbol is used for Vss. This is the reference terminal for taking measurements, and it's considered to be 0 volts compared to all other voltages on your board. The four sockets for connecting jumper wires to Vss are along the top of the breadboard to the right.

There is a row of sixteen sockets along the left side of the breadboard for connecting to the BASIC Stamp I/O pins. Each I/O pin has a label. I/O pin P0 is connected to the bottom left socket. Pin P1 is the next socket up, and above that socket is the connection to pin P2, and so on through pin P15 at the top left.
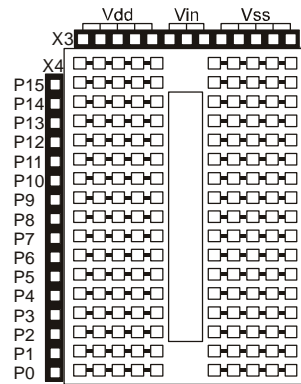
Figure 1-6
Prototyping Area on the Board of
Education and HomeWork Board

*Note the power header X3 provides
connections to Vdd (+5V), Vin (direct
battery or power supply voltage) and
Vss (0 V, ground.) Also shown is
how each row of 5 sockets on the
breadboard is electrically connected
underneath. BASIC Stamp I/O pins
are accessed via the X4 header.*

Figure 1-6 also shows some samples of 5-socket-wide rows that are electrically connected underneath the breadboard. There are 34 of these 5-socket-wide rows arranged in the two columns on the breadboard. If you want to connect two jumper wires to each other, you can just plug them into the same row of 5. Then the wires are electrically connected. Likewise, if you want to connect one or more wires to the terminal of a part, just plug them into the same row on the breadboard and they'll be connected.

Building the Analog and Digital Comparator

Build the circuit according to the schematic in Figure 1-7. This schematic is like a list of connections between circuit symbols. Try to use this list to build the circuit. Here is a partial list of the connections shown in the schematic:

- √ The wiper terminal of the 10 kΩ pot is connected to pin 3 of the LM358 op-amp.
- √ Pin 2 of the LM358 is connected to pin 1 of the LM358.
- √ Pin P7 of the BASIC Stamp is connected to the wiper terminal of the pot.
- √ Pin 8 of the LM358 is connected to Vdd on your board.
- √ Pin 4 of the LM358 is connected to Vss on your board.

Keep following the schematic like a list and you'll have the circuit built in no time.

Pot with
analog
output from
wiper
terminal

Op-Amp
voltage
follower circuit

Analog LED
output

Binary
LED
output.
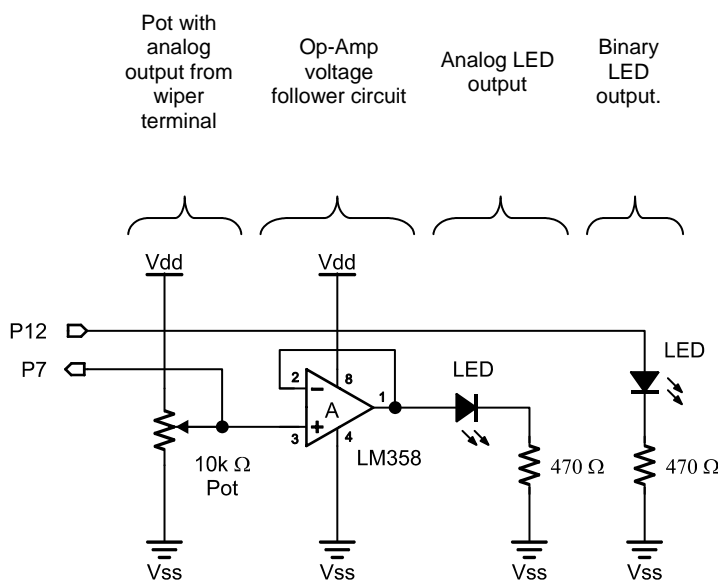
Figure 1-7
Circuit Schematic.

*Remember to treat
this schematic like
a list of
connections for
building your
circuit.*

*Although this circuit
only has a few
parts, it actually
has 4 separate sub
circuits, and each
has a different
function as shown*

Vdd   Vdd

P12

P7

LED

LED

2   8

A

3   4

1

10k Ω
Pot

LM358

470 Ω   470 Ω

Vss   Vss   Vss   Vss

The potentiometer is what makes the analog output. The op-amp is wired to function as a voltage follower. The voltage follower drives the analog LED output. Then there's a separate circuit that uses a BASIC Stamp I/O pin to drive an LED.

Figure 1-8 shows a breadboard example of the schematic from Figure 1-7. For extra tips on building circuits on the breadboard, consult *What's a Microcontroller?*

1
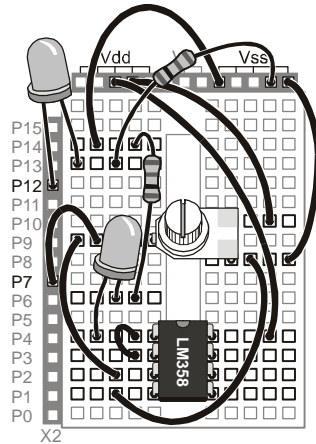
Figure 1-8
Breadboard Example

Compare this breadboard example to the schematic from Figure 1-7. Is the LM358 connected right? Does Vdd go to pin 8 and does Vss go to pin 4?

It turns out that the answer to both questions is yes. Vss is connected by a wire to the lower right terminal on the pot. Then another wire connects from that pot terminal to pin 4 on the LM358.

Since you can follow a wire all the way from pin 4 on the LM358 to the Vss terminal, that means it is in fact connected directly to Vss.

> IMPORTANT: Pay careful attention to placing the LM358 so that the index mark (half-circle notch) is to the top, as shown on this breadboard example. If you place it in reverse, the op-amp will be ruined after the battery or power supply is connected to your board.

Programming the Project

The program State_of_P7.bs2 below shows how PBASIC can be used to instruct the BASIC Stamp to do several tasks. First, the BASIC Stamp monitors the state its I/O pin P7, which is set to function as an input. Remember, P7 is connected to the wiper terminal of the pot.

Depending on the analog voltage level at pin P7, the BASIC Stamp interprets the input as low or high (binary 0 or 1). As soon as the input to P7 receives the high signal, the BASIC Stamp sends a high signal to the LED circuit via pin P12. When the input is low, a low signal is sent to P12. The Debug Terminal is also used to monitor the state of pin P7.

Enter Program State_of_P7.bs2 into the BASIC Stamp Editor, and save it under a convenient name, such as State_of_P7_1_1R0.bs2. This stands for Program Listing 1.1 Revision 0. Make sure the programming cable is properly connected to your board, and to the serial or USB port on your computer. Also make sure that a battery or power supply is properly connected, then run the program by clicking the BASIC Stamp Editor's Run button "►", or Ctrl-R.

```
' Basic Analog and Digital - State_of_P7.bs2
' Check the state of P7 and show it on the Debug Terminal.
' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG CLS
INPUT 7
OUTPUT 12

DO
  OUT12 = IN7
  DEBUG HOME, "The state of P7 is ", BIN IN7
LOOP
```

About the Code

The first lines start with an apostrophe. This means that they're comments and not PBASIC commands. The first line reminds you of the book and file name, for future reference.

```
' Basic Analog and Digital - State_of_P7.bs2
```

The second comment is a description of the program. What does the program do?

```
' Check the state of P7 and show it on the Debug Terminal.
```

The next two lines are special comments. We call them compiler directives and they are intended to identify the BASIC Stamp and the PBASIC version we are using. For example if you're following this manual with a BASIC Stamp 2 SX, you might replace the compiler directive "**'{$STAMP BS2}**" with "**'{$STAMP BS2SX}**"**. (Hint: use the toolbar buttons to insert these compiler directives.)

```
' {$STAMP BS2}
' {$PBASIC 2.5}
```

It's good to start the Debug Terminal and clear it before using it to display data. This way you will avoid inadvertently displaying outputs from previous programs that were in the BASIC Stamp module's memory. The Debug Terminal starts automatically the first time it encounters the **DEBUG** command in a PBASIC program. This **DEBUG** command clears the Debug Terminal after it is opened:

```
DEBUG CLS
```

The BASIC Stamp needs to be told how to treat the I/O pins connected to the circuit. They can either be set to function as inputs or outputs.

This PBASIC command sets BASIC Stamp I/O pin P7 to function as an input pin:

```
INPUT 7
```

Likewise, I/O pin P12 functions as output with this command:

```
OUTPUT 12
```

The rest of the program should be done over and over again, so this is a good place to put a **DO…LOOP** loop. So, at the point we want to start to repeat the code we put:

```
DO
```

Later in the program, the command **LOOP** is entered. Each time the program gets to the **LOOP** command, it returns to **DO** and starts executing instructions all over again.

The next task is to make the LED connected to pin P12 light up when the voltage at P7 is high enough to qualify as a binary high signal. In other words, if the input value measured at P7 is a binary-1, then the output at P12 should be set to binary-1. Although there are several ways to accomplish this, the easiest way is to set the binary output value of pin P12 equal to the binary input value of pin P7.

```
OUT12 = IN7
```

The **DEBUG** command can be used to display the signal levels received by an I/O pin functioning as an input in the Debug Terminal. The **DEBUG** command below prints three different items. When printing more than one item with a **DEBUG** command, always separate each of the items with commas.

```
DEBUG HOME, "The state of P7 is ", BIN IN7
```

The first item listed using the **DEBUG** command is **HOME**, and it sends the cursor to the top-left "home" position in the Debug Terminal. Note how **HOME** is followed by a comma to separate it from the next item. The next item is a message in quotes: **"The state of pin P7 is "**. Whenever you want to display a text message in the Debug Terminal, use quotes. The third item is **BIN IN7**, which tells the Debug Terminal to display the binary input value measured at pin P7.

We want the BASIC Stamp keep checking the voltage at P7 over and over again. We also want the BASIC Stamp to automatically update the LED and the Debug Terminal with the latest information from P7. This is accomplished by repeatedly sending the program back to the **DO** command we created earlier.

To send the program back to the DO statement to start the process all over again, use the command:

```
LOOP
```

Troubleshooting

Here are a few tips on things to check if your program doesn't work as expected.

- It often takes a few tries to catch all of the wiring mistakes and programming errors.

- The most common mistake is a program entry error. In some cases the BASIC Stamp Editor will tell you there is a mistake. For example if you misspelled a command, the BASIC Stamp Editor will not understand it, and it will tell you which term it didn't understand by highlighting it and displaying a brief message.

- In other cases the program will run even though a line of code was typed incorrectly. For example you might have typed 13 when you meant to type 12. With a mistake like this, when you try to run the program, the LED connected to pin P12 won't light up when you expect it to because the high signal gets sent to pin P13 instead.

- Another common mistake is to plug a wire into the wrong socket on the breadboard. If an LED is not turning on and off the way it should, and there are no programming errors, check the wiring. Also check to make sure the cathode and anode of the LED are connected to the proper places. When an LED is hooked up backwards it won't emit light.

- If the information in the Debug Terminal appears garbled or nonsensical, try closing the Debug Terminal and running the program again.

1

The Output

As you adjust the potentiometer knob, note how the LED at the voltage follower's analog output varies in brightness. Meanwhile the LED circuit driven by P12 either turns on or off. This characterizes the difference between analog voltage and digital (binary) voltage.

The output displayed in the Debug Terminal should resemble the output shown in Figure 1-9. The state of P7 will either be 0 or 1. Correspondingly, the LED circuit driven by P12 will either be off or on.
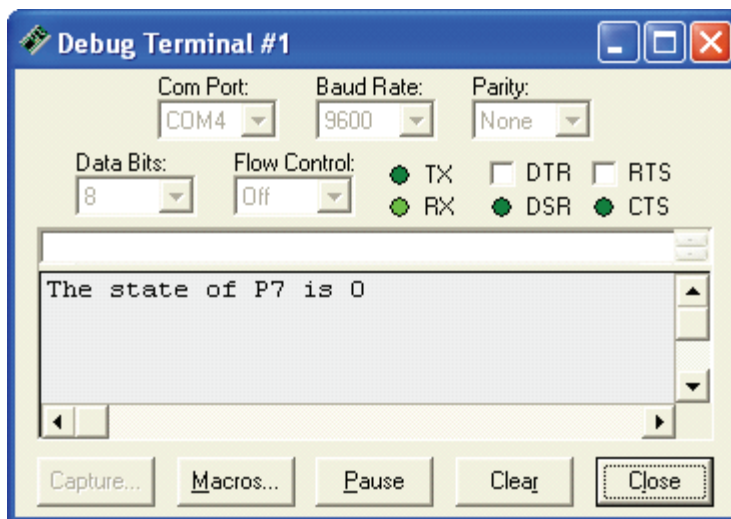
Figure 1-9
Debug Terminal
Output for Program

Adjust the potentiometer until you have found the threshold voltage. You will know when you've found it because the Debug Terminal will indicate that the state of P7 skips back and forth between 0 and 1 with just the slightest adjustment of the pot. Note the position of the potentiometer. After we build a DC voltmeter in Chapter #3, we can find out how close we are to 1.4 volts, which is the actual threshold voltage of a BASIC Stamp I/O pin when it's functioning as an input.

About the Comparator

Using PBASIC, we programmed the BASIC Stamp to function as a comparator. A comparator is so named because it's a circuit that compares its input voltage to a particular voltage, also called the threshold voltage. If the input voltage is higher than the threshold voltage, the comparator sends a high signal at its output. If the input to the comparator is below the threshold voltage, it sends a low signal.

In the case of our circuit and program, when the analog voltage at pin P7 is below 1.4 volts, the BASIC Stamp sends a low signal (0 volts) at pin P12. When the analog voltage at pin P7 is above 1.4 volts a high signal (5 volts) is sent from pin P12. As you can see in the Debug Terminal, the BASIC Stamp interprets analog input below 1.4 volts as low (binary 0) and input above 1.4 volts as high (binary 1).

Operating a comparator near the threshold voltage is interesting because you can make a small change in voltage at a BASIC Stamp input pin, say from 1.3 to 1.5 volts, and it results in a fairly large change, from 0 to 5 volts at the output.

1

What have I learned?

On the lines below, insert the appropriate words from the list on the left.

carbon op-amp analog threshold output schematic breadboard	A _____ shows circuit symbols connected to each other by lines. Each circuit symbol corresponds to a component and the lines connecting the symbols work like a list of connections that can be used to guide the construction of a working circuit on a _____. An _____ is an analog building block that was used in this experiment as a voltage follower. In this experiment, the potentiometer was used as a source of _____ voltage. The voltage at the wiper terminal of the potentiometer can be made to vary depending on where the wiper is in contact with the _____ element. A comparator is a device that sends a binary output that depends on whether its analog input is above or below a certain _____ voltage. A comparator can react to a small change in input voltage with a comparatively large change in _____ voltage.

Questions

1. Circle the word that makes the sentence true: The input for a voltage follower is at the (inverting / non-inverting) terminal of the op-amp in this experiment.

2. How do you tell the difference between the cathode and the anode on an LED from your parts kit?

3. If the threshold for a comparator is 2.5 volts, and the input is at 1.5 volts, what's the output going to be?

4. Explain what the command **DEBUG HOME** does. What has to be done to display more than one item using a single **DEBUG** command?

5. What command would you use to set pin P8 to function as an input?

Challenge!

1. Add another LED to the circuit to your board and use pin P11 to drive it in an inverted state. In other words, when one LED is on, the other off. Hint: Add 1 to the output value at P11 to invert it.

2. Modify the code from Program State_of_P7.bs2 so that the LED output flashes on and off when the potentiometer output is above the threshold voltage of the BASIC Stamp input pin. Hint: You can use the command pause 500 to make the program pause for half a second.

3. Modify State_of_P7.bs2 to cause one LED to light up when the input voltage at pin P7 is above the threshold voltage and the other LED to light up when the voltage at pin P7 is below threshold voltage.

Why did I learn it?

In this experiment, we compared a binary LED output to an analog LED output. Aside from knowing that the voltage at the wiper terminal had just crossed the threshold voltage, there was no way of indicating the potentiometer's position or the LEDs brightness. On the other hand, in the neighborhood of the threshold voltage, fine variations in analog voltage could be detected.

Even with the limited amount of analog information provided by a binary input, we were able to develop a device called a comparator, which has many applications in electronics design. As we'll discover in a later experiment, the 555 timer chip in you're A & D parts kit can do some pretty amazing things. This is due in part to two microscopic comparators in the chip.

How can I apply this?

In later experiments, we'll use the threshold voltage to measure the frequency of sound for record and playback purposes. We can also build another type of analog to digital converter using a very simple circuit, the BASIC Stamp and the concept of threshold voltage. We'll use this technique to measure light intensity as well as values for capacitors from the Analog and Digital parts kit.

Chapter #2: Introduction to Bit Crunching

An important step in learning how to make the BASIC Stamp process analog data is learning how to make it send and receive binary numbers. It's also important to understand how binary numbers work, and how to convert from a binary number to a decimal number.

BASIC COMMUNICATION

This experiment introduces some techniques for transmitting and receiving binary numbers using the BASIC Stamp. In this experiment, we'll make a binary keypad for transmitting binary numbers to the BASIC Stamp. The BASIC Stamp will also be programmed to process and display the binary numbers it receives. The binary numbers will be displayed by LEDs as well as by the Debug Terminal. The Debug Terminal will also come in handy for monitoring and displaying the binary numbers in decimal form.

In *What's a Microcontroller?*, we learned that binary is the number system used by microcontrollers, and that it works with two digits, 0 and 1. The BASIC Stamp is part of an entire class of digital electronic devices that can interpret 0 volts as binary-0 and 5 volts as binary-1.

Binary is useful for describing both states and numbers. In terms of states, the two digits in the binary number system (0 and 1) can be used to describe off/on, closed/open, no/yes, etc. Combinations of binary digits can be used to describe numbers. For example, the binary numbers 101, 110, and 111 describe the decimal numbers 5, 6, and 7. These numbers can in turn be used to describe analog information, such the position of a door as it swings open and closed.

Parts Required

(2) 470 Ω resistors
(2) 220 Ω resistors
(2) 10 kΩ resistors
(2) Pushbutton switches
(2) Red LEDs
(misc.) Jumper wires

The Pushbutton

There's just one new part and circuit symbol to introduce for this experiment, the pushbutton in Figure 2-1. Note how each terminal on the circuit symbol corresponds to two pins on the part. If you want to connect to a particular terminal shown on the symbol, you can connect to either (or both) of the two corresponding pins on the part.

The open space in the circuit symbol indicates that the switch is normally open. When the two terminals of a switch are not connected, the switch is referred to as an open circuit. Under normal circumstances (when the pushbutton is not pressed), the circuit is open, thus the name normally open.
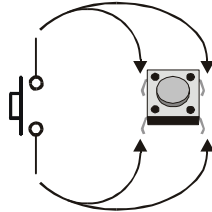
Figure 2-1
Pushbutton Circuit
Symbol Compared to
the Component

Building the Circuit

Figure 2-2 shows the schematic for this experiment. Remember to think of a schematic as a list of components and connections. For example, the anode of the right LED is connected to the BASIC Stamp I/O pin P5 socket on your board. The cathode is connected to one terminal of a 470 Ω resistor. The other pin on that same resistor is connected to the Vss terminal on your board, and so on. Follow schematics faithfully when constructing circuits.
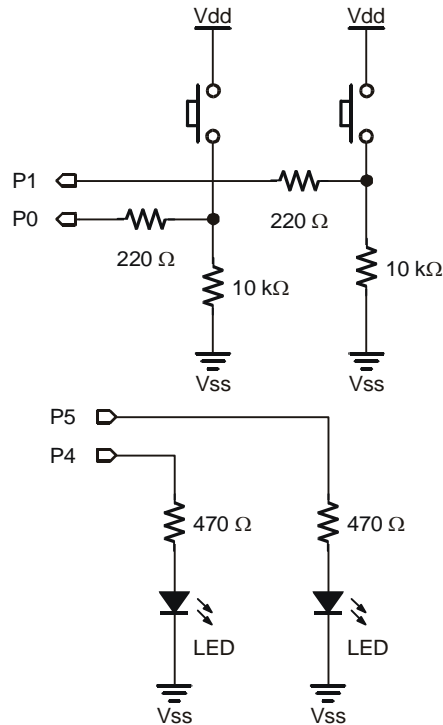
Figure 2-2
Schematic featuring two
pushbutton circuits and
two LED circuits.

Before making a PBASIC program telling the BASIC Stamp how to interface with this circuit, it's essential to understand how the circuit works. The LEDs are pretty straightforward. Set P4 high and the LED lights up; set P4 low and the LED goes dark again. The LED circuit connected to P5 works the same way.

Now, what about the pushbuttons? Let's look at what pin P0 sees when the pushbutton is pressed, then not pressed. When the pushbutton is pressed, P0 gets connected directly to Vdd, which is 5 volts. P0 sees a high signal. When the pushbutton is not pressed, P0 is connected to Vss (0 volts) through the 10 kΩ resistor. Then P0 sees a low signal. This concept applies to both pushbuttons shown in Figure 2-2.

Figure 2-3 shows a breadboard example of the circuit schematic. Of the two BASIC Stamp I/O pins used for the pushbuttons, the lower pin (P0) is connected to the right pushbutton. Likewise, the right pin (P1) is connected to the lower pushbutton. The reason

the wires for the pushbuttons cross relates to the way binary numbers are written, which will be explained later in this experiment.
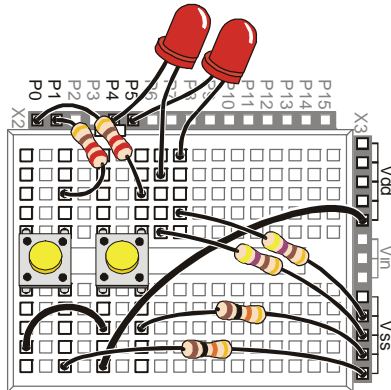
Figure 2-3
Breadboard Example

Entering binary numbers on the pushbuttons will be easiest if you orient your board as shown.

Note that the left button is connected to pin P1, and the right button is connected to P0.

Program Listing 2.1 makes the left LED in Figure 2-3 light up when the left pushbutton is pressed. Likewise, the right LED lights up when the right pushbutton is pressed. The program also displays the activity of the pushbuttons in the Debug Terminal.

Programming the Project

Here is a more precise description of the program specifications for the pushbuttons and LEDs.

- When P0 receives a low signal, P5 should send a low signal.
- When P0 receives a high signal, P5 should send a high signal.
- When P1 receives a low signal, P4 should send a low signal.
- When P1 receives a high signal, P4 should send a high signal.

The Debug Terminal can be used to display what the BASIC Stamp receives at pins P0 and P1. **DEBUG** commands are used to display the binary values the BASIC Stamp receives as well as their decimal equivalents in the Debug Terminal.

Let's see how this can be done using PBASIC. Enter the Program Listing 2.1 into the BASIC Stamp Editor, and save it as PL2_1R0.bs2. This stands for Program Listing 2.1 Revision 0. Make sure that your board has power and the programming cable is properly connected, then run the program.

```
' Basic Analog and Digital - PL2_1R0.bs2
' Program Listing 2.1 Revision 0.
' {$STAMP BS2}
' {$PBASIC 2.5}

a        VAR     Bit
b        VAR     Bit
d        VAR     Nib

INPUT 0
INPUT 1
OUTPUT 4
OUTPUT 5

DEBUG CLS

DO
  a = IN0
  b = IN1
  OUT4 = b
  OUT5 = a
  d = (2*b) + (1*a)
  DEBUG HOME, "State of pin P0 is ", BIN a, CR
  DEBUG "State of pin P1 is ", BIN B, CR, CR
  DEBUG "2-bit binary number: ", CR
  DEBUG "P1   P0", CR
  DEBUG " ", BIN b, "    ", BIN a, CR, CR
  DEBUG "Decimal equivalent: ", DEC1 d, CR
LOOP
```

The Output

Here's how the program should work. When no pushbuttons are pressed, the Debug Terminal output should match Figure 2-4, and both LEDs should be off. Try pressing the right button (in Figure 2-3). Did the right LED light up? Did the state of P0 in the Debug Terminal change to 1? Is the decimal equivalent 1? If so, it looks like your circuit and program are working well so far.
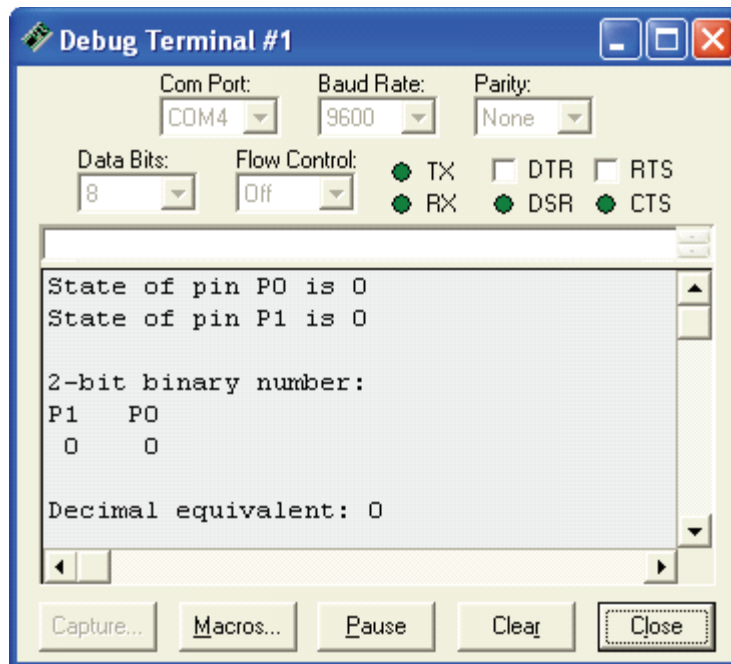
Figure 2-4
Debug Terminal
Output for Program
Listing 2.1

So how do you count from decimal-0 to decimal-3 using the binary pushbuttons? The two-bit binary equivalent of decimal-0 is 00. When you don't press either of the pushbuttons, the decimal output is 0 in the Debug Terminal. When you press the right button, you get 01, which has a decimal equivalent of 1. When you press the left button, you get 10, which has a decimal equivalent of 2. When you press both pushbuttons, you get 11, which has a decimal equivalent of 3.

About the Code

As with the program of the previous chapter, the first lines start with an apostrophe, so they are comments and compiler directives, and the BASIC Stamp ignores them.

```
' Basic Analog and Digital - PL2_1R0.bs2
' Program Listing 2.1 Revision 0.
' {$STAMP BS2}
' {$PBASIC 2.5}
```

Next, three variables are defined. Variables can be used to store values while the program is running. The letters **a** and **b** are defined as variables that store 1-bit each. So, the variable a can store a single binary digit, likewise with the variable **b**. The letter **d** is defined to be a variable that stores a "nibble" of binary information.

```
a       VAR     Bit
b       VAR     Bit
d       VAR     Nib
```

> **Memory and Food:** A *bit* of memory can store one binary digit, either a 0 or a 1.
>
> A *nibble* of memory stores 4 bits.
>
> A *byte* (pronounced bite) stores 8 bits.
>
> A *word* stores 16 bits.
>
> Since a bit, a nibble, and a byte, all sound like references to eating food, perhaps a better name for 16 bits might have been "dinner".

This segment of code uses commands introduced in the last chapter. First, two I/O pins are declared inputs and two more pins are declared outputs. Then the Debug Terminal is opened and cleared.

```
INPUT 0
INPUT 1
OUTPUT 4
OUTPUT 5
DEBUG CLS
```

We want the BASIC Stamp keep checking the inputs over and over again. We also want the BASIC Stamp to automatically update the LEDs and the Debug Terminal with the latest information on the pushbuttons. The way to accomplish this is to keep repeating the program inside a **DO…LOOP** loop. To define the start point of the loop we use **DO** and to send the program back to this point we'll use the **LOOP** command.

```
DO
```

Next, we need to check the state of the pushbuttons by checking the input at pins P0 and P1. The first of these two commands sets the bit variable **a** equal to the state measured at pin P0. The second command sets the bit variable **b** equal to the state measured at pin P1.

```
a = IN0
b = IN1
```

Next, we need to set the output at pin P4 equal to the input taken at pin P1. The left LED which is connected to P4 will light up when the left button, which is connected to P1, is pressed. Likewise, we need to set the output at pin P5 equal to the input measured at pin P0.

Since the input values were set to the variables, **a** and **b**, we can use **a** and **b** to dictate the output values at pins P4 and P5.

```
OUT4 = b
OUT5 = a
```

We could just as easily have used the commands **OUT4=IN1** and **OUT5=IN0**; however, using variables to store the values in memory has advantages as the programs get more complicated. In the next experiment, it will be necessary to use variables to store values.

The reason we used variables in this program is because they can be manipulated arithmetically, and the next task is to convert from binary to decimal. To do this, multiply the variable **b** by 2 and the variable **a** by 1 and add them together. The nibble variable **d** is used to store this new value. This is the method for converting a 2-bit binary number to a decimal number. The next section shows how to do this for a binary number of any size.

```
d = (2*b) + (1*a)
```

> **BASIC Stamp Memory:**
>
> **RAM**: The BASIC Stamp has 26 bytes of RAM (random access memory) that can be used for storing variable values. Another 6 bytes of RAM is used to interface the BASIC Stamp with its I/O pins.
>
> **EEPROM**: Short for Electrically Erasable Programmable Read Only Memory, EEPROM is used mainly to store the PBASIC programs. EEPROM can also be used to store data values that do not change frequently.

In the calculation we just did using PBASIC, the parentheses are necessary to maintain the normal algebraic order of operation. This is because the BASIC Stamp performs its math beginning at the left. Then, it performs each operation it encounters while checking the line from left to right.

Without the parentheses, **d** would be set equal to the value $((2 \times b + 1) \times a)$ because that's the order in which the operators (+, -,*, /, etc) are encountered. When parentheses are used, the BASIC Stamp completes operations within parentheses first, and then it does its sweep of operations from left to right.

Six **DEBUG** commands are used to display all the measured states and the calculated binary values in the Debug Terminal. The first **DEBUG** command below displays four different items. Remember, each item in a single **DEBUG** command must be separated by a comma.

The **DEBUG HOME** command sends the cursor to the top-left "home" position in the Debug Terminal. Note that it is followed by a comma to separate it from the next item. The next item is a message in quotes: **"State of pin P1 is "**.

Whenever you want to print a text message to the Debug Terminal, use quotes. The third item is **BIN a**, which tells the Debug Terminal to print the binary value of the variable **a.** The fourth item is **CR**, which makes the Debug Terminal print a carriage return.

```
DEBUG HOME, "State of pin P0 is ", BIN a, CR
```

A similar message is printed for the variable **b**, without the **HOME** command. The **HOME** command works well when it's used once per loop. Remember that **DEBUG HOME** sends the cursor to the top-left corner of the Debug Terminal. If we used **HOME** more than once in the loop, the information displayed after the first **HOME** command would be overwritten by the information following the second **HOME** command.

```
DEBUG "State of pin P1 is ", BIN B, CR, CR
```

Next, two **DEBUG** commands are used. Each prints a message in quotes followed by two carriage returns.

```
DEBUG "2-bit binary number: ", CR
```

Next, another message in quotes is printed followed by a single carriage return.

```
DEBUG "P1   P0", CR
```

In this next command, the quotes contain spaces. The first pair of quotes just contains one space (the space bar on the keyboard was pressed once). Then the binary value of **b** is printed, followed by another two spaces in quotes, followed by the binary value of **a**, then two more carriage returns.

```
DEBUG " ", BIN b, "  ", BIN a, CR, CR
```

Here's something new. The modifier **DEC** was used to print the decimal value of the variable **d.** Because this is the last instruction we want to repeat, it's followed by the **LOOP** command.

```
     DEBUG "Decimal equivalent: ", DEC1 d, CR
LOOP
```

Counting in Binary

Table 2-1 shows how to count from 0 to 3 using 2-bit binary numbers and how to count from 0 to 7 using 3-bit binary numbers.

Note that four numbers (decimal 0 through 3) can be represented with a 2-bit binary number. Eight numbers (0 through 7) can be represented with a 3-bit binary number. 4 bits can describe 16 different numbers, 5 bits can describe 32 different numbers and so on.

Table 2-1:Measured voltages during charge cycle		
Decimal number	**2-bit binary representation**	**3-bit binary representation**
0	00	000
1	01	001
2	10	010
3	11	011
4		100
5		101
6		110
7		111

You can always determine how many counting numbers (combinations of 0s and 1s) can come from a given number of bits by using this formula:

$$\text{combinations} = 2^{\text{bits}}$$

This means the number of combinations equals two raised to the power of the number of bits. For 2 bits, the number of combinations is $2^2 = 4$. For 3 bits, the number of combinations is $2^3 = 8$, and so on.

Converting from binary to decimal takes two steps. The first step is to multiply each bit by its power of two. Table 2-2 shows the powers of two for up to 8 bits. When you multiply each bit by its value from Table 2-2, you end up with a series of decimal values. The second step is to add up all the decimal values.

Table 2-2: Bit Multipliers for an 8-bit Binary Number								
Bit	7	6	5	4	3	2	1	0
Multiplier	128	64	32	16	8	4	2	1

Bit multipliers and Powers of Two: Bit-0 is the least significant bit (LSB) and bit-7 is the most significant bit (MSB). That's because bit-0 makes the smallest contribution to the number and bit-7 makes the largest contribution. Thinking about a binary number as starting on the left with bit-7 and ending on the right with bit-0 is useful because these numbers indicate the power of 2 for each digit.

Examples:

The multiplier for bit-0 is 1, which equals 2^0.

The multiplier for bit-1 is 2, which equals 2^1.

The multiplier for bit-7 is 128, which equals 2^7.

Note: You can use powers of two to extend Table 2-2 to any number of bits!

As an example, let's convert binary-1011 to decimal. First, multiply each bit by its power of two from Table 2-2.

```
8 × 1 = 8
4 × 0 = 0
2 × 1 = 2
1 × 1 = 1
```

Second, add all 4 of the decimal values:

```
8 + 0 + 2 + 1 = 11
```

Now we know the binary number 1011 is equal to the decimal number 11.

Parallel and Serial Transmission

Program Listing 2.1 repeats the entire check and report on the pushbutton states routine over and over again. Because the BASIC Stamp checks for input over and over again without waiting for some kind of signal that the data is ready, we are sending the binary numbers to the BASIC Stamp asynchronously.

> **Asynchronous** means not synchronized. In the case of our binary keypad, it means that we change the binary values whenever we want to without waiting for permission from the BASIC Stamp to do so. Likewise, the BASIC Stamp checks the signals at P0 and P1 as fast as it can without waiting for a signal from us that says the data is ready to be checked.

We are also sending the binary bits across two separate data lines at the same time. This means we are sending our data bits to the BASIC Stamp in parallel.

The BASIC Stamp has a 16 I/O pins. We could actually send a word-size binary number to the BASIC Stamp in parallel. The problem is that we wouldn't have any pins left for outgoing signals or other input data. When dealing with larger binary numbers, sending serial data instead of parallel data can be useful because it reduces the number of BASIC Stamp I/O pins used to receive data.

When sending serial data, there has to be some way of letting the BASIC Stamp know when each new bit is ready. The BASIC Stamp has built-in functions for sending asynchronous as well as synchronous serial data.

In this next example, the same two pushbuttons are used to send the BASIC Stamp a nibble (4 bits) of serial, synchronous data. The result is displayed in the Debug Terminal.

> **Parallel** means the data bits are sent across more than one data line at the same time. We just finished using the pushbuttons to send two parallel bits.
>
> **Serial:** Instead of sending data in parallel along multiple data lines, a single data line can be used and the data bits can be sent one after another.
>
> **Synchronous:** Sending data synchronously means we are sending the data in a time-coordinated manner (in sync). Technically, it means that the sender and receiver of the data bits do so according to signals from the same clock

Reprogramming to Receive Serial Data

Enter Program Listing 2.2 into the BASIC Stamp Editor, and save it under the name PL2_2R0.bs2.

2

```
' Basic Analog and Digital - PL2_2R0.bs2
' Program Listing 2.2 Revision 0.
' {$STAMP BS2}
' {$PBASIC 2.5}

n       VAR     Nib
d       VAR     Nib

INPUT 0
INPUT 1

FOR n = 1 TO 4

  DO
    'Wait for high
  LOOP UNTIL IN1=1

  DO
    'Wait for low
  LOOP UNTIL IN1=0

  d = d << 1
  d = d + IN0
  DEBUG HOME, "Shifting in bits: ", BIN4 d
NEXT

DEBUG CR, CR, "Done shifting.", CR, CR
DEBUG "Decimal value: ", DEC2 d, CR, CR
```

The Output

The Debug Terminal will initially show a blank screen when you run the program. Follow these instructions carefully to send the synchronous, serial data. First, press and hold the right button. Then press and release the left button. The output should look like Figure 2-5 below.
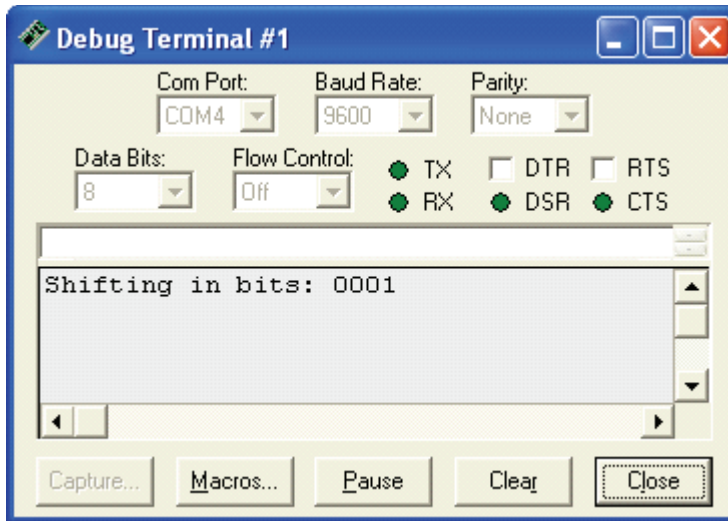
Figure 2-5
Debug Terminal
Output for Program
Listing 2.2.

Next, release the right button, and press and release the left button again. The output should change so that it looks like Figure 2-6.
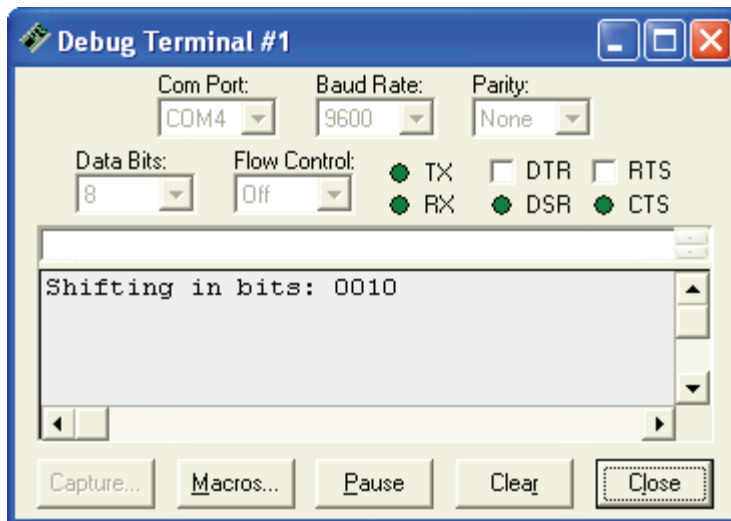
Figure 2-6
Debug Terminal
Output for Program
Listing 2.2.

Press and hold the right button, then press and release the left button twice. Then the output should look like Figure 2-7.
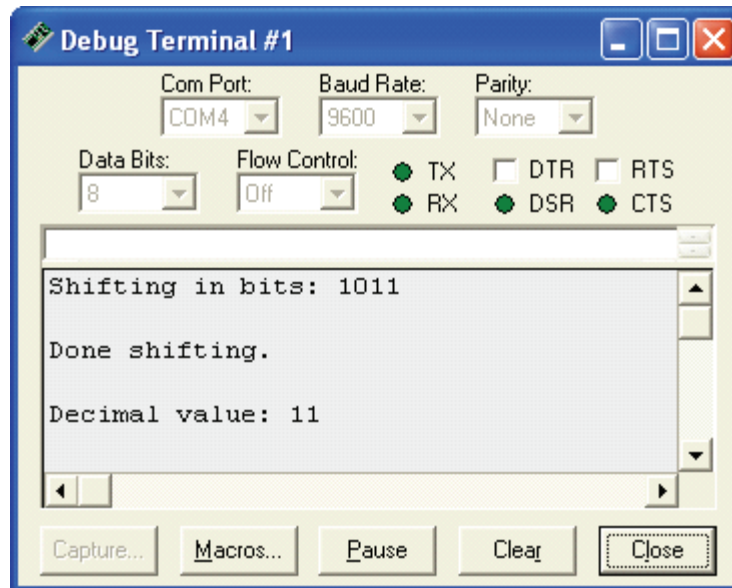
Figure 2-7
Debug Terminal
Output for Program
Listing 2.2.

If your program worked as shown, you just synchronously shifted 4 serial bits into the BASIC Stamp's RAM. In addition, it was verified that the decimal value of binary-1011 really is 11 in the decimal number system.

Figure 2-8 shows how these events occurred (from left to right) in a timing diagram. The left pushbutton was the clock signal. The clock signal consisted of a series of clock pulses. Each clock pulse was the press and release of the left pushbutton. This sent a low-high-low signal to P1. Each time the left pushbutton was released, the BASIC Stamp checked the state of the data line, which was the state of the right pushbutton (P0). The BASIC Stamp is programmed to read the input data after the clock signal's transition from high to low. This transition is referred to as the negative edge of the clock pulse.
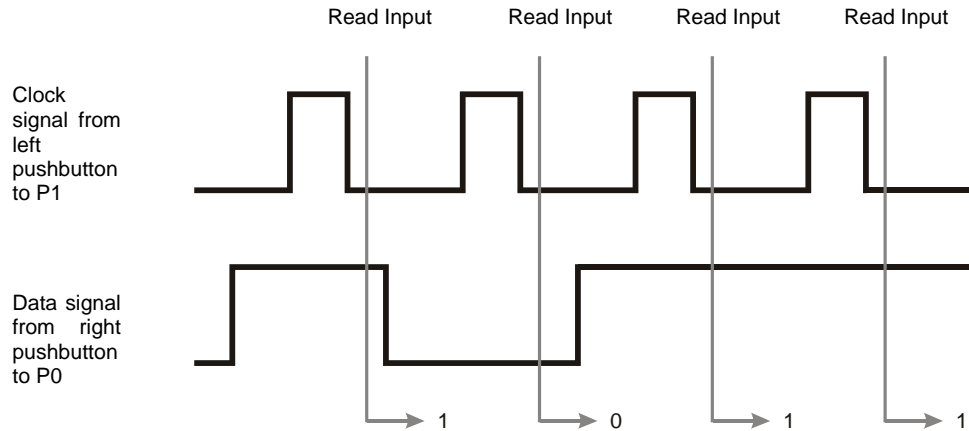
Figure 2-8: Timing Diagram.

About the Code

Like Program Listing 2.1, we use a comment to include some information about the program at the beginning. Remember, as soon as an apostrophe appears in a line of PBASIC code, everything to the right of the apostrophe is ignored by the BASIC Stamp. Next the two nibble variables are defined, and I/O pins P0 and P1 are set to function as inputs. One of those nibble variables, **n**, is used by the **FOR…NEXT** loop.

The main portion of the code is in a **FOR…NEXT** loop, which has the following syntax:

FOR *Counter* = *StartValue* **TO** *EndValue*

In the example code snippet below, **n** is the *Counter*, **1** is the *StartValue* argument and **4** is the *EndValue* argument of the **FOR…NEXT** command.

```
FOR n = 1 TO 4
    DEBUG DEC n, CR
NEXT

DEBUG "Done!"
```

This **FOR…NEXT** loop executes the PBASIC command between the **FOR** command and the **NEXT** command four times. The first time through the loop the value of **n** is 1, the second time through the value of **n** increments to 2, and so on until **n** gets to 4. After completing the 4th pass, the program skips out of the loop and continues on to the line of code after the **NEXT** command.

Program Listing 2.2 has a more complicated set of commands that gets repeated within its **FOR…NEXT** loop, and it is different from what we've seen before. It starts with a **DO…LOOP** conditional loop. This loop is used to check the value at pin P0. If the value of P0 is low, **LOOP UNTIL IN1=1** sends the program back to **DO**. If the value at pin P1 is high, then the program executes the next line, after the **LOOP** instruction.

```
FOR n = 1 TO 4
  DO
    'Wait for high
  LOOP UNTIL IN1=1
```

The same technique is applied with the next two lines of code (and one comment), which repeat themselves until a low signal is received.

```
  DO
    'Wait for low
  LOOP UNTIL IN1=0
```

The command **d=d<<1** shifts all the bits in **d** left by 1. When values are shifted, the vacant space in the rightmost bit (the LSB) is automatically filled with a zero. Then the bit value at pin P0 is loaded into the LSB position in the nibble. This is done using the **d=d+IN0** command, which adds the single bit measured at P0 to the value of the nibble variable d. The second time through the loop, the value that was placed in the LSB gets shifted left by one, so it ends up in bit-1. Meanwhile the next value sampled at P0 is placed in the LSB slot. The shift and add process is repeated four times as each bit is shifted into the byte.

```
  d = d << 1
  d = d + IN0
```

Each time a new bit is shifted in, we use the **DEBUG** command to display the new value. Each time the program gets to the **NEXT** command it returns to the **FOR n = 1 TO 4** command, and the value of **n** is incremented, until it gets to 4. Finally the program breaks out of the loop.

```
    DEBUG HOME, "Shifting in bits: ", BIN4 d
NEXT
```

Once the **FOR…NEXT** loop is finished and all the bits are shifted into the **d** variable, two messages are printed. The second of the two **DEBUG** commands has a new modifier, **DEC2**. This modifier is used to make the Debug Terminal display the value of **d** as two decimal digits.

```
DEBUG CR, CR, "Done shifting.", CR, CR
DEBUG "Decimal value: ", DEC2 d, CR, CR
```

While Chapter 1 introduced analog voltage, this chapter introduced the basics of sending and receiving binary numbers. In the next experiment, we'll combine these topics to build a digital DC voltmeter, which is a device that measures analog voltage and displays the measurement as a digital value.

What have I learned?

On the lines below, insert the appropriate words from the list on the left.

largest word decimal BASIC Stamp synchronously LSB PBASIC byte binary numbers	The _____ can be used to process information about analog measurements using _____ . Before working with analog to digital interfaces, it's important to understand how the BASIC Stamp sends, receives, and stores binary data. It's also important to be able to use _____ to program the BASIC Stamp to send, receive, and store binary data. When converting a binary number to its _____ equivalent, each bit should be multiplied by a particular power of two. The _____ is the rightmost bit in a binary number, and it makes the smallest contribution to the value of that number. It is also referenced as bit-0. The MSB is the leftmost bit. It makes the _____ contribution to the value of a binary number. The bit just to the left of the LSB is bit-1, the bit 2 bits to the left of LSB is bit-2 and so on. The BASIC Stamp can store a single bit, a nibble, which is 4 bits, a _____, which is 8 bits, or a _____, which is 16 bits. To save I/O pins, the BASIC Stamp can send and receive serial data as opposed to parallel data, which uses multiple data lines. Binary data can also be sent _____ or asynchronously.

Questions

1. Determine the decimal equivalents of these binary numbers: 1010, 1111, 0010, and 0100.

2. The command `d = d<<1` was used to shift the bits in the variable `d` to the left by 1. What command do you think would be used to shift the bits to the right? What command would you use to shift the bits by, say, 3 to the left?

3. Explain the difference between serial and parallel data.

4. Explain the difference between synchronous and asynchronous data transmission.

Challenge!

1. Program Listing 2.1 is used to count to three. Write a program that uses three parallel bits (use pin P2 for the third bit) and counts to 7. You can check your work by connecting the input pin P2 to Vdd for a high signal and to Vss for a low signal (using a 1 kΩ resistor).

2. Modify Program Listing 2.2 so that it displays the clock pulses you apply to the left pushbutton with the left LED.

3. Modify Program Listing 2.2 so that it shifts the bits back out, displaying them with the right LED.

2

Why did I learn it?

The goal in this series of experiments is to demonstrate how a device such as the BASIC Stamp, which processes binary data, can be used to interface with the analog world. Binary numbers and states form the foundation for how microcontrollers, microprocessors and an entire class of binary circuits process data.

Analog data can be effectively processed with binary numbers using the techniques in this experiment. Learning the basics of processing binary data will help make a myriad of electronic circuits easier to understand. This is also the foundation for how the home computer processes data. Understanding data at the binary level ("bit crunching") also makes various programming languages easier to understand.

How can I apply this?

In some of the upcoming experiments, we will process analog data using serial and parallel data. We will also use synchronous and asynchronous communication. The BASIC Stamp can be connected to other integrated circuits and exchange binary data. The BASIC Stamp can also be programmed to convert the binary data to a meaningful decimal form.

We'll use these techniques to measure voltage, sound, light, etc.

The BASIC Stamp has commands which automate the serial data transmit and receive processes in both asynchronous and synchronous modes. We will encounter the serial, synchronous method in Chapter #3. The BASIC Stamp also has features available to simplify the sending and receiving of parallel data, as we'll discover in Chapter #4 where asynchronous parallel data is transmitted.

Chapter #3: Basic Analog to Digital Conversion

BUILD YOUR OWN DIGITAL DC VOLTMETER

A digital DC voltmeter (DC DVM) is a handy tool for measuring voltage between two contact points. In this experiment, we will build a DVM for measuring DC voltage in the 0 to 5 volt range. A common use for a DC DVM is testing the voltage (potential) between the two terminals on a battery.

A digital voltmeter is so named because it displays its measurements with digits. The digits 0 through 9 and a decimal point are used to display the voltage measurements as decimal values. The digits 0 and 1 could be used. It would still be a "digital" voltmeter, but it would have binary display instead of a decimal display. Making sense out of each measurement would be time consuming. Since our DVM processes its measurements in binary, we'll start with a binary display and then modify it to a more conventional and easy to read decimal display.

In Chapter #1, we used an LED circuit to display changes in analog voltage level applied to a circuit. As a "continuously variable value", analog voltage varies within a continuous range. We'll use the potentiometer as we did in Chapter #1 to make a range of voltages that can vary continuously between 0 and 5 volts.

Although information about analog voltage can be processed efficiently with binary devices, the voltage has to be sampled and described using binary numbers first. The ADC0831 is a common integrated circuit that does this job. It describes the analog information with binary numbers for devices that process binary information, such as the BASIC Stamp.

In this experiment, we will make a DVM using the BASIC Stamp together with the ADC0831 integrated circuit. A pot will be wired to your Board of Education or HomeWork Board and adjusted to make analog output voltage. The DVM will then be used to measure samples from the pot's continuous range of voltage outputs.

> **Continuous range:** A minimum value, a maximum value, and everything in between. When a source of voltage varies over a continuous range, it is considered an analog voltage.
>
> We will use our DVM to sample voltages over a continuous range, from 0 to 5 volts. So, the voltage we measure might be 1.234 volts or 3.857564… volts, or 4.9999… volts, etc.

Parts Required

(1) ADC0831
(1) 10 kΩ potentiometer
(10) Jumper wires, give or take a few

The Potentiometer, a Source of Variable Voltage

There is a reason why the voltage at the wiper terminal of a pot changes when you turn the knob. The wiper terminal makes the single resistive element in the pot work like two resistors in series. Figure 3-1 shows two resistors in series. When input voltage is applied and output voltage is measured as shown in Figure 3-1, the circuit is referred to as a voltage divider. R_1 and R_2 are the resistances between the wiper and the other two terminals on the pot, and their values change as the pot is adjusted. Since the pot causes the R_1 and R_2 to vary, we can call our wiper terminal the output of a variable voltage divider.
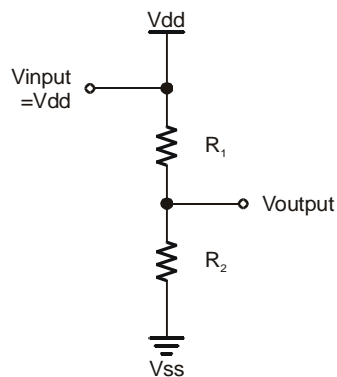
Figure 3-1
Voltage Divider Circuit

A Voltage Divider Circuit shows how the wiper in a potentiometer makes the single resistive element look like two resistors in series.

`Voutput` *is the voltage measured at the wiper terminal.*

Resistor Values: When you know the value of the two resistors in Figure 3-1, you can predict the output voltage using this equation.

$$V_{output} = V_{input} \times \frac{R_2}{R_1 + R_2}$$

Not surprisingly, it's called the voltage divider equation, and this technique for scaling down an input voltage is commonly referred to as using a voltage divider.

> **(?)** **Resistors in Series:** A chain of resistors connected end to end. Three resistors in series are shown below. The three resistors can be viewed as a single resistance whose value is:
>
> $$R_{series}=R_1+R_2+R_3$$
>
> ◦—$\bigwedge\bigwedge$—$\bigwedge\bigwedge$—$\bigwedge\bigwedge$—◦
>
> R1   R2   R3

The ADC0831 Integrated Circuit - An 8-bit Analog to Digital Converter

The ADC0831 is an integrated circuit referred to as an 8-bit analog to digital converter (A/D converter) with synchronous serial output. Let's look at what each of these terms mean:

- An integrated circuit (IC) is a circuit with microscopic components implanted on the surface of a silicon wafer. The Analog and Digital Parts Kit has three chips used in these experiments. Each chip is a black casing with eight pins. The black casing houses and protects an integrated circuit.

- An A/D converter measures an analog voltage sample and returns a binary number that describes the sample.

- 8-bit is the number of binary digits the ADC0831 uses to describe the analog voltage it samples. 8-bit is also the resolution of the A/D converter. You can count from 0 to 255 (decimal) using an 8-bit binary number. This means that the ADC0831 can approximate the voltage it measures as one of 256 levels. A higher resolution converter, such as 12-bit, would break the same voltage range into 4096 levels because you can count from 0 to 4095 with 12 binary bits.

- Synchronous and serial are terms we learned about in Chapter #2. We sent serial binary digits (bits) to the BASIC Stamp using one pushbutton and the bits were synchronized to a second pushbutton that was used to send a clock signal. The ADC0831 works in a similar way. The difference is that the ADC0831 depends on a clock signal sent by the BASIC Stamp to time the sending of each serial output bit.

The BASIC Stamp will be programmed to read and store the 8 serial bits transmitted by the ADC0831. We'll also program the BASIC Stamp to display the decimal equivalent of the binary output. Next, we'll use this decimal equivalent to calculate and displays the

measured voltage in decimal form (our DVM output). The BASIC Stamp must also be programmed to send binary control signals to make the ADC0831 do its job.

Figure 3-2 shows a pin map of the ADC0831. Each pin has a number and a label. The number is important for getting the wires connected to the right pins when constructing your circuit. The labels indicate the function of each pin.

> **?** **Binary control signal:** A voltage signal with two possible states, low or high, that is sent to tell a device how or when to do something. The ADC0831 requires control signals to activate it and a clock signal to synchronize the sending of each of its output bits.
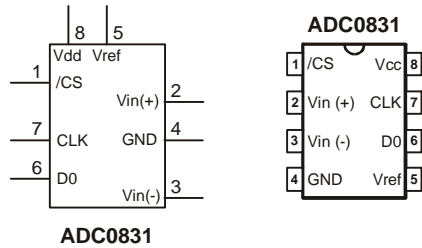
Figure 3-2
ADC0831 Circuit Symbol and Pin Map

The pin map on the right shows the pins and labels according to where they are on the chip. The circuit symbol on the left also shows the pins and their labels, but it's typically drawn in a way that most conveniently fits into the schematic.

The notation for the ADC0831's inputs and outputs works as follows: Vin(+) is the analog input, and D0 is the serial output. VREF and Vin(-) are used to bias the IC. Vcc and GND are used for supplying power to the IC. Vcc is essentially the same term as Vdd on the Board of Education or HomeWork Board, and GND corresponds to Vss. /CS stands for active low chip select, and CLK stands for clock. Both are inputs for binary control signals.

> **?** **Bias:** A method of applying specific voltage levels at certain places in a circuit to calibrate or tune it.

To prime the ADC0831 for taking a measurement, the /CS pin has to receive a signal from the BASIC Stamp that starts high, then goes low. This signal has to stay low for the duration of the conversion. Then the CLK input must receive a single clock pulse (a term introduced in Chapter #2, Figure 2-8) to signify that the conversion should start at the next clock pulse. For this IC, a clock pulse starts low, goes high, and then goes low again. It takes 8 more clock pulses to complete the conversion. Each time a clock pulse is received by the CLK input, another of the serial bits is sent by the D0 output.

Electronics designers use data sheets to find the kind of information just discussed. Each IC manufacturer publishes data sheets for the integrated circuits they make. The information just covered on the pin map and control signals was condensed from a data sheet published by National Semiconductor, the maker of the ADC0831. Of course all of the datasheets are available on the manufacturer's web sites.

Build It

Figure 3-3 shows the schematic for this experiment. This is a fairly simple circuit to build, so let's try it without the breadboard example. Hopefully you're getting the hang of the list of connections described by a schematic. Remember, when working with the connections to an IC, use the index mark on the chip along with the pin map to figure out the pin numbers!
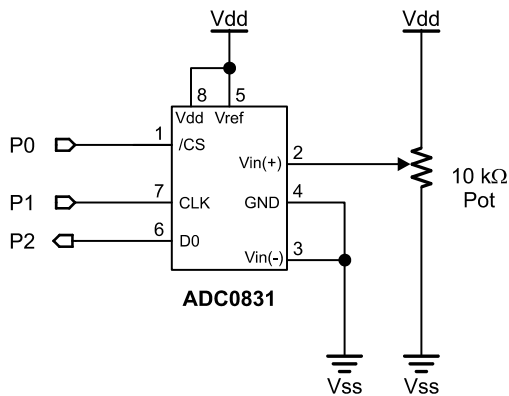
Figure 3-3
Schematic

List of connections made on this schematic:
- *Pin 1 on the ADC0831 is connected to I/O pin P0 on the BASIC Stamp.*
- *The wiper terminal of the pot is connected to pin 2 on the ADC0831.*
- *Of the two remaining terminals on the pot, one is connected to Vdd on your board, and the other is connected to Vss.*
- *Pins 3 and 4 on the ADC0831 are connected to Vss.*
- *Pins 5 and 8 on the ADC0831 are connected to Vdd.*
- *Pins 7 and 6 on the ADC0831 are connected BASIC Stamp I/O pins P1 and P2 respectively.*

Program It

Program Listing 3.1 is the first step to a functional DC voltmeter. This program displays the 8-bit serial output of the ADC0831. Enter the code and save it as P3_1R0.bs2. There will be three revisions of this program listing, so it will become important to keep track of your source code versions.

We'll modify the code so that it also displays the decimal conversion of the 8-bit binary number. Next we'll add some more code to adjust the number to a 5 volt scale. Make sure

your circuit is constructed correctly and your programming cable and power source are connected, then run the program.

```
' -----[ Title ]---------------------------------------------------------
' Basic Analog and Digital - PL3_1R0.bs2
' Program Listing 3.1 Revision 0.
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]--------------------------------------------------
adcBits          VAR     Byte
v                VAR     Byte
r                VAR     Byte
v2               VAR     Byte
v3               VAR     Byte

' -----[ Initialization ]------------------------------------------------
CS               PIN     0
CLK              PIN     1
DataOutput       PIN     2

DEBUG CLS                                      'Start display.

' -----[ Main Routine ]--------------------------------------------------
DO
  GOSUB ADC_Data
  GOSUB Calc_Volts
  GOSUB Display
LOOP

' -----[ Subroutines ]---------------------------------------------------
ADC_Data:
  LOW CLK
  LOW CS
  PULSOUT CLK, 210
  SHIFTIN DataOutput,CLK,MSBPOST,[adcBits\8]
  HIGH CS
RETURN

Calc_Volts:
RETURN

Display:
  DEBUG HOME
  DEBUG "8-bit binary value:  ", BIN8 adcBits
RETURN
```

The Output

If the pot is adjusted somewhere in the middle of its range, the output displayed in the Debug Terminal should look similar to Figure 3-4. As you adjust the pot, the zeros and ones should change rapidly. At one end of the pot's range of movement, you should get all 0's; at the other end you should get all 1's.
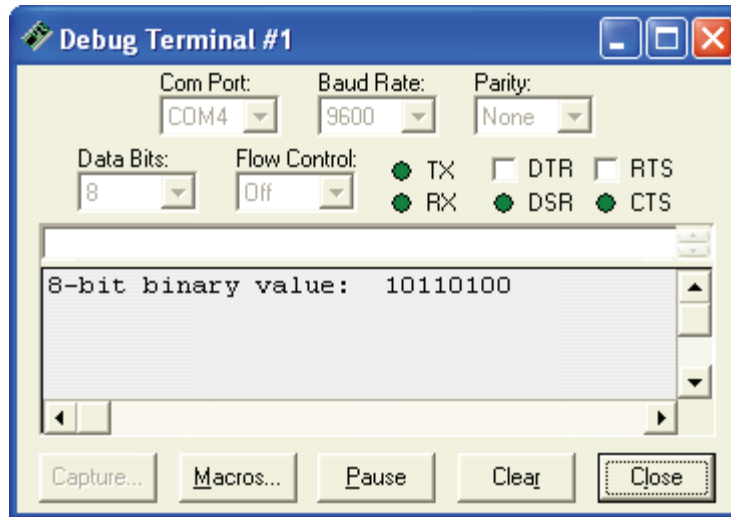
Figure 3-4
Debug Terminal Output for Program Listing 3.1.

If your Debug Terminal responds this way, it's likely your circuit and program are working right. If it doesn't do this, check the wiring on your circuit. Also make sure code is entered correctly. Sometimes just one wrong letter will cause the program not to work properly. The Debug Terminal could also be hidden from view. It can be accessed from the menus by selecting Run/Debug/New Terminal in the BASIC Stamp Editor.

About the Code

The first few lines of text in this program are comments that begin with apostrophes, and they don't have any function in the program aside from explaining it to someone reading the code, and identifying the BASIC Stamp model and PBASIC version.

```
' -----[ Title ]---------------------------------------------
' Basic Analog and Digital - PL3_1R0.bs2
' Program Listing 3.1 Revision 0.
' {$STAMP BS2}
' {$PBASIC 2.5}
```

The next section is called the variable declarations section, and it begins with a comment explaining that this is the declarations section. This program uses just the **adcBits** variable at present. We'll add code that will make use of the other four variables, **v**, **R**, **v2**, and **v3**.

```
' -----[ Declarations ]-------------------------------------
adcBits       VAR     Byte
v             VAR     Byte
r             VAR     Byte
v2            VAR     Byte
v3            VAR     Byte
```

Following is a new type of declaration we haven't used before. Three constants are defined using the **PIN** directive. After we define these constants, we can use **CS** in place of the number **0**, **CLK** in place of the number **1**, and **DataOutput** in place of the number **2**. The names for the pin identifications were chosen to correspond with the ADC0831's pin labels. The numbers were chosen based on BASIC Stamp I/O pin numbers.

```
' -----[ Initialization ]-----------------------------------
CS            PIN    0
CLK           PIN    1
DataOutput    PIN    2
```

Next there's the main routine section containing three **GOSUB** commands. The **DO…LOOP** loop runs 3 different subroutines over and over again. The subroutines are named **ADC_Data**, **Calc_Volts**, and **Display**.

> **?** **Subroutine** is a small program that does a specific task within a larger program.

```
' -----[ Main Routine ]-------------------------------------
DO
  GOSUB ADC_Data
  GOSUB Calc_Volts
  GOSUB Display
LOOP
```

So, how does a **GOSUB** command work? As shown in the flow diagram in Figure 3-5, **GOSUB ADC_Data** means go to the subroutine labeled **ADC_Data** and come back when finished. The program jumps to the **ADC_Data** label and starts executing commands. As soon as it gets to the **RETURN** command, the program jumps back to the command just after **GOSUB ADC_Data**. In this case, the next command is another **GOSUB** command, **GOSUB Calc_Volts**.

```
' -----[ Main Routine ]----------------------
DO
  GOSUB ADC_Data
  GOSUB Calc_Volts
  GOSUB Display
LOOP

' -----[ Subroutines ]----------------------
ADC_Data:
   LOW CLK
   LOW CS
   PULSOUT CLK, 210
   SHIFTIN DataOutput,CLK,MSBPOST,[adcBits\8]
   HIGH CS
RETURN
```
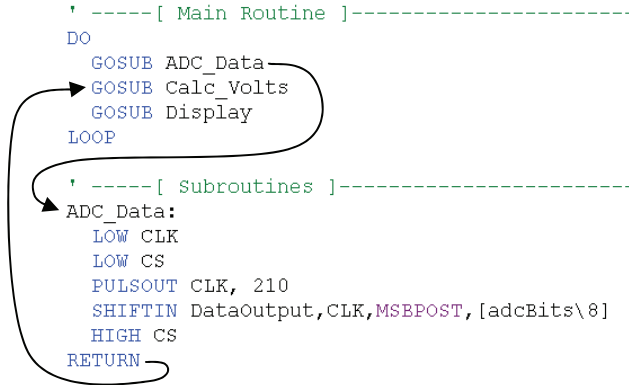
Figure 3-5
Flow Diagram

*A subroutine sends the program to the specified label. In this case the label is **ADC_DATA**. Then the program continues executing commands until it encounters the return command. The return command sends the program back to the command immediately after the **GOSUB** command. In this case it's another **GOSUB** command.*

The subroutine **ADC_Data** sends control signals to and collects output data from the ADC0831. This subroutine is where the usefulness of the **PIN** directive really shows. P0 on the BASIC Stamp is connected to the /CS pin on the ADC0831. Likewise, pins P1 and P2 are connected to CLK and D0. When sending signals to the /CS pin, we can enter a command like **HIGH CS** instead of **HIGH 0**. It makes more sense when writing the code and it makes deciphering the code easier too. It's also easier to change one definition in the top of the program should you decide to connect the ADC0831 to a different BASIC Stamp I/O pin

The **LOW CLK** command is necessary so that the clock pulses take the right form. Using this command guarantees a **PULSOUT** command a little later in the subroutine will send a clock pulse that has the right shape, low-high-low. The command **LOW CS** "enables" the ADC0831. You can have several ADC0831 chips connected to the same CLK (clock) and D0 (data) lines. The BASIC Stamp then sets only one ADC0831 chip's /CS low at a time to get its voltage measurement. The ADC0831 that's getting the low signal to its /CS pin is the one that will reply to pulses from the BASIC Stamp to its CLK pin by transmitting binary values that report the voltage measurement with its D0 pin. All the rest of the ADC0831's with high signals to their /CS pins would ignore the clock pulses.

```
ADC_Data:
   LOW CLK
   LOW CS
```

The **PULSOUT CLK, 210** command sends a clock pulse to the ADC0831's CLK input. This is the first clock pulse, and all it does is tell the ADC0831 to start converting on the next clock pulse. Because of this, we don't need to check for input from D0 after this first clock pulse.

```
PULSOUT CLK, 210
```

Since we set the clock low before this command, **PULSOUT** sends the desired low-high-low signal. The duration of the high segment is twice the number specified in the **PULSOUT** command, in two microsecond (µs) units. 1 µs = 1/1,000,000 of a second. Therefore the duration of this high signal is 2 µs × 210 = 420 µs.

The command **SHIFTIN D0,CLK,msbpost,[adcBits\8]** is a powerful instruction that takes care of all the synchronous serial communication so that we don't have to program it as we did in Chapter #2. In effect, this command sends clock pulses to the ADC0831's CLK input and reads output bits from ADC0831's D0 output. This command also loads each of the ADC 0831's output bits into the **adcBits** byte.

```
SHIFTIN DataOutput,CLK,MSBPOST,[adcBits\8]
```

The **SHIFTIN** command is discussed in more detail in the *BASIC Stamp Manual*, but the general format for the command is:

SHIFTIN *data_ pin, clock_pin, mode, [variable\bits]*

In our case, the data pin is **DataOutput**, a constant equal to the number 2. This constant is used to reference BASIC Stamp I/O pin P2 in this program. Likewise, the clock pin is **CLK**, which is a constant equal to the number 1, and it references BASIC Stamp I/O pin P1. The mode in this case is **MSBPOST**, and it's one of four transmission modes that can be used in this command. It indicates that the ADC0831's output bits are ready after the clock pulse's negative edge, the transition from high to low. It also indicates that the bits are transmitted in descending order, starting with the MSB. **[adcbits\8]** means the data is shifted into the **adcBits** variable, and 8 bits are expected.

A low signal had to be sent to the ADC0381's /CS pin near the beginning of the subroutine to enable it. Now that the subroutine is done getting the measurement, the ADC0831's /CS should be left high with **HIGH CS** to disable it. (The / to the left of CS indicates "active low", and CS is the abbreviation of Chip Select.)

```
HIGH CS
```

The **Calc_Volts** subroutine is empty right now, but we will develop the code for this subroutine shortly. The subroutine will calculate the measured voltage to the hundredth's decimal place.

```
Calc_Volts:
RETURN
```

At present, the **Display** subroutine just displays the binary output for each analog voltage sample taken by the ADC0831. It will be modified to display the decimal equivalent of the 8-bit binary value. It will also be modified to display the voltage measurement.

The following **DEBUG** commands send the cursor to the top-left "home" position in the Debug Terminal. Then it prints the message in quotes. The modifier **BIN8** makes it so the value of the **adcBits** variable is displayed as 8 binary digits.

```
DEBUG HOME
DEBUG "8-bit binary value:  ", BIN8 adcBits
```

If the number of digits displayed is likely to vary, when using the **DEBUG HOME** command, always specify how many digits the numeric outputs should have with modifiers like **BIN8**, **DEC3**, etc. When **DEBUG CLS** is used, it's OK to skip specifying the number of digits, so modifiers such as **BIN** and **DEC** can be used instead.

The **DEBUG HOME** command is better for programs that cycle through loops where the Debug Terminal display is updated frequently and rapidly. When **DEBUG CLS** is used under these circumstances, the repeated clearing the Debug Terminal causes a flicker that makes the display difficult to read.

The **RETURN** command sends the program back to the line immediately following the **GOSUB Display** command.

We will modify the **Display** subroutine to display the decimal equivalent of the binary contents of **adcBits** in the Debug Terminal. Code will also be added to make the Debug Terminal display our DVM reading.

Interpreting the Output

The ADC0831 measures an analog voltage at its input. Then it sends the BASIC Stamp a binary number describing the value it measured. For now, we'll focus on a voltage scale that starts with 0 volts and ends at 5 volts.

With an 8-bit binary number, you can start counting with 00000000 and count all the way up to 11111111. Translated to decimal numbers, it's the same as counting from 0 to 255. When applied to a 5 volt scale that starts at 0 volts, it's the same as counting from 0 to 5 volts using 255 voltage steps.

For the 5 volt scale, when the ADC0831 measures 0 volts, you get 00000000. When it measures 5 volts, the output is 11111111. It turns out that the Debug Terminal output 10110100 from Figure 3-4 is the same as the decimal number 180. Decimal-180 in turn corresponds to a measured voltage of 3.53 volts.

Binary to Decimal Conversion Revisited

So how do we know that 256 combinations can come from an 8-bit binary number? Remember, you can always tell how many numbers (combinations of 0s and 1s) can come from a given number of bits by using this formula from Chapter #2:

$$\text{combinations} = 2^{\text{bits}}$$

This means the number of combinations equals two raised to the power of the number of bits. For 8 bits, the number of combinations is $2^8 = 256$. For 12 bits, the number of combinations is $2^{12} = 4096$, and so on.

Let's use the two-step method from Chapter #2 to convert the 8-bit binary number 10100101 to its decimal equivalent. Here is a repeat of the bit multipliers table to work with:

Table 3-1: Bit Multipliers for an 8-bit Binary Number								
Bit	7	6	5	4	3	2	1	0
Multiplier	128	64	32	16	8	4	2	1

First, multiply each bit by its power of two from Table 3-1:

```
128 x 1 = 128
64 x 0 = 0
32 x 1 = 32
16 x 0 = 0
8 x 0 = 0
4 x 1 = 4
2 x 0 = 0
1 x 1 = 1
```

Second, add all 8 of the decimal values:

```
128 + 0 + 32 + 0 + 0 + 4 + 0 + 1 = 165
```

Now we know the binary number 10100101 is equal to the decimal number 165. To display this conversion, a single **DEBUG** command can be added to the **Display** subroutine. Added lines are shown with a "□".

```
Display:
  DEBUG HOME, "8 bit binary value:  ", BIN8 adcBits
  DEBUG CR, CR, "Decimal value:  ", DEC3 adcBits    '□ new line
RETURN
```

The command **DEBUG CR, CR, "Decimal value: ", DEC3 adcBits** tells the Debug Terminal to display two carriage returns followed by the message in quotes, followed again by the 3-digit decimal value of **adcBits**. If the actual number only has one or two digits, the Debug Terminal will automatically display leading zeros since **DEC3** was specified. For example, the number 7 will display as 007, and the number 85 as 085, etc. With some careful adjustment of the pot, we can check our work using output sample shown in Figure 3-6.
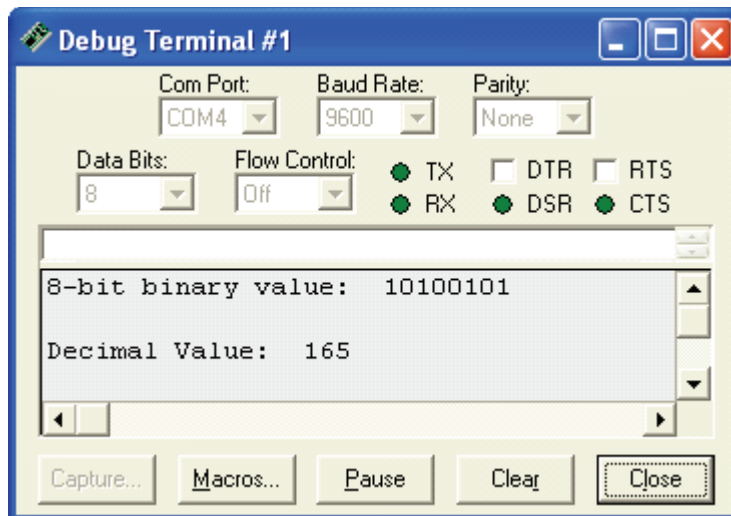
Figure 3-6
Debug Terminal
Output for Program
Listing 3.1,
Revision 1.

Calculate Voltage

Now that we know the decimal equivalent of the ADC0831's binary output, we can do a few calculations to get the measured voltage. To find out what voltage the decimal number corresponds to, we need to calculate where in the voltage range the number falls. Here is an effective way to think about the problem.

- We know that the voltage is on a 0 to 5 volt scale, and we know that the ADC0831's output is on a scale from 0 to 255.
- In other words, the measured voltage is to 5 as the A/D output is to 255.

This translates to fractions as:

$$\frac{\text{Voltage}}{5} = \frac{\text{Decimal A}/\text{D Output}}{255}$$

We can re-arrange this equality to calculate the voltage:

$$\text{Voltage} = \frac{5 \times (\text{Decimal A/D Output})}{255}$$

So, now we know to multiply by 5 and divide by 255 for a 5 volt scale with 256 levels. We can calculate the voltage from Figure 3-6 where the ADC0831's output is 10100101 = 165. The measured voltage is:

$$\text{Voltage} = \frac{5 \times 165}{255} = 3.24 \text{ Volts rounded to two decimal places.}$$

To calculate and display this voltage using the BASIC Stamp, we'll add some code to both the **Calc_Volts** and **Display** subroutines. First, the voltage equation needs to be expressed in PBASIC code. Here is an example of some code that could reasonably be expected to work.

```
v = 5 * adcBits / 255
```

This PBASIC calculation looks like it will give us the output we want, but it won't. It's instructive to try it this way and see what happens. Modify the **Calc_Volts** and **Display** subroutines in Program Listing 3.1 as follows:

```
Calc_Volts:
  v = 5 * adcBits / 255                           '□ new line
RETURN

Display:
  DEBUG HOME
  DEBUG "8-bit binary value:  ", BIN8 adcBits
  DEBUG CR, CR, "Decimal value:  ", DEC3 adcBits
  DEBUG CR, CR, "DVM Reading:  ", DEC3 v, " Volts"  '□ new line
RETURN
```

We calculated that 165 would lead to a measured voltage of 3.24 volts. The 003 volts shown in Figure 3-7 is only accurate to the nearest volt! What happened?
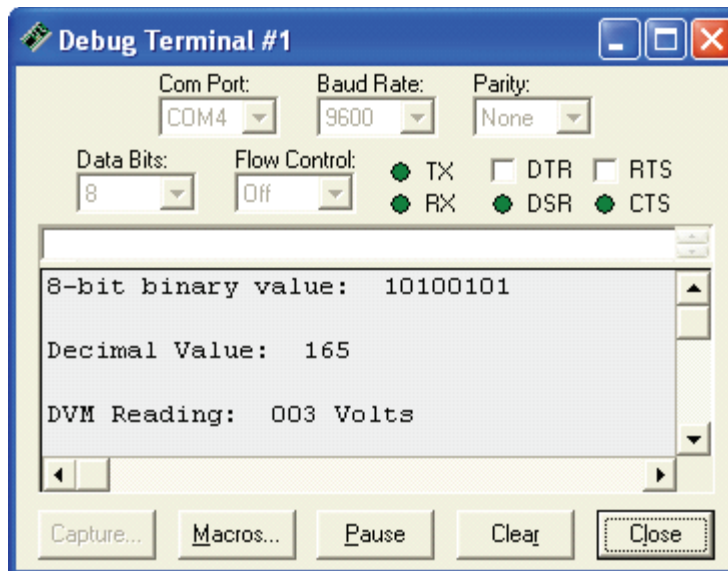
Figure 3-7
Debug Terminal
Output for Program
Listing 3.1, Revision
2.

The PBASIC command set for the BASIC Stamp does arithmetic using integer values. Integers are the counting numbers: …-2, -1, 0, 1, 2, 3, etc. The largest integer the BASIC Stamp can process is 65535. When using integer arithmetic, the fractional part of any answer is discarded. Fortunately, we can still use integer arithmetic to find the fractional values we are trying to display.

Before dividing, the A/D output is multiplied by 5. This doesn't cause any problems.

$5 \times$ (Decimal A/D Output) is essentially the same as $5 \times$ adcBits .

In our voltage calculation example, that's $5 \times 165 = 825$. Since 825 is an integer that is less than 65535, this part of the calculation goes without a hitch. The problem occurs when we try to divide 825 by 255. The answer has a fractional component that never gets calculated with integer math.

Doing "long division" with a pencil and a piece of paper takes several steps, and it relies solely on integer arithmetic. Let's look at how to calculate the answer for this division problem.

$$\text{Voltage} = \frac{5 \times (\text{Decimal A/D Output})}{255} = 3 + \text{a remainder of } 60$$

In long division, calculating the part of the answer to the right of the decimal point is repetitive. We multiply the remainder by 10, then divide by 255 again, then take another remainder, multiply it by 10, and divide by 255 again, etc. A shortcut to this procedure is to take the remainder, and multiply it by 100, then divide by 255. This gives us two decimal places. Let's try it.

$$(60 \times 100) \div 255 = 6000 \div 255 = 23.5924... \xrightarrow[\text{Integer Math}]{} 23$$

Remember: the BASIC Stamp cuts off everything to the right of the decimal point without rounding. This is called truncating. The result we get is 23. This result should have been rounded up to 24 because 23.5294 is more than half way to the next integer value, 24. For now, let's stick with 23 to the right of the decimal point.

Our answer using this algorithm is the integer 3 to the left of the decimal point, and the integer value 23 to the right of the decimal point. Since we used only integers in our arithmetic, it should work using PBASIC and the BASIC Stamp.

> **?** **What's an Algorithm?** An algorithm is a procedure for solving a problem. The procedure is broken down into repeatable steps.

Since the BASIC Stamp works with integers, it's not surprising that there is a PBASIC command to calculate the integer remainder of a division problem. The operator for division is / and the operator for getting the remainder is //. Let's try converting this algorithm into PBASIC code to do the work for us. The steps for long division below show the PBASIC commands corresponding to the steps in the algorithm.


```
                                    v+r  ◄──────────┐
v=5*adcBits/255      255)5×adcBits              R=(5*adcBits//255)

                                    v2
v2=(100*R)/255       255)100×r
```

This gives us our three PBASIC commands for calculating the values to the left and right of the decimal point. To reconstruct the fractional value on the display, we'll print a period "." in-between the two values.

The first of the three commands is already in our **Calc_Volts** subroutine. Just add the other two instructions to complete the algorithm.

```
Calc_Volts:
  v = 5 * adcBits / 255
  r = 5 * adcBits // 255               '□ new line
  v2 = 100 * R / 255                   '□ new line
RETURN
```

The **Display** subroutine also needs to be updated to print the two variable values with a period in between them. Make sure to update the line in the **Display** subroutine exactly as shown below.

```
Display:
  DEBUG HOME
  DEBUG "8-bit binary value:  ", BIN8 adcBits
  DEBUG CR, CR, "Decimal value:  ", DEC3 adcBits
  DEBUG CR, CR, "DVM Reading:  "             '□ new line
  DEBUG DEC1 v, ".", DEC2 v2, " Volts"       '□ new line
RETURN
```

Now run the program again, and see what happens. Figure 3-8 shows an output sample, which is now almost ready to display to the nearest hundredth of a volt. All that needs to be corrected is a rounding error in the hundredths decimal place.
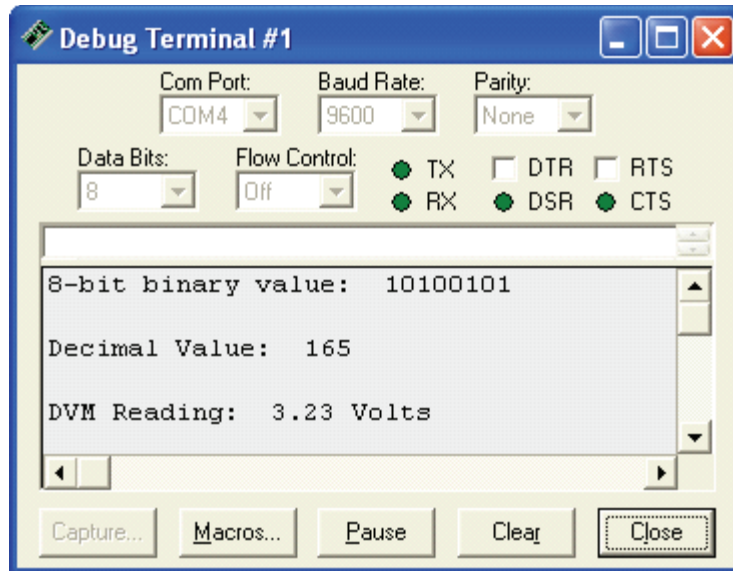
Figure 3-8
Debug
Terminal
Output for
Program
Listing 3.1,
Revision 3.

All that remains to be done is to correct the rounding error in the hundredth's place. This rounding problem can be corrected by adding the segment of code shown below to the `Calc_Volts` subroutine.

```
Calc_Volts:
  v = 5 * adcBits / 255
  r = 5 * adcBits // 255
  v2 = 100 * r / 255
  v3 = 100 * r // 255                  '□ new line
  v3 = 10 * v3 / 255                   '□ new line
  IF (v3 >= 5) THEN v2 = v2 + 1        '□ new line
  IF (v2 >= 100) THEN                  '□ new line
    v = v + 1                          '□ new line
    v2 = 0                             '□ new line
  ENDIF                                '□ new line
RETURN
```

The Output

The output sample from Figure 3-9 indicates that the DVM is now calculating to the hundredth's decimal place correctly.
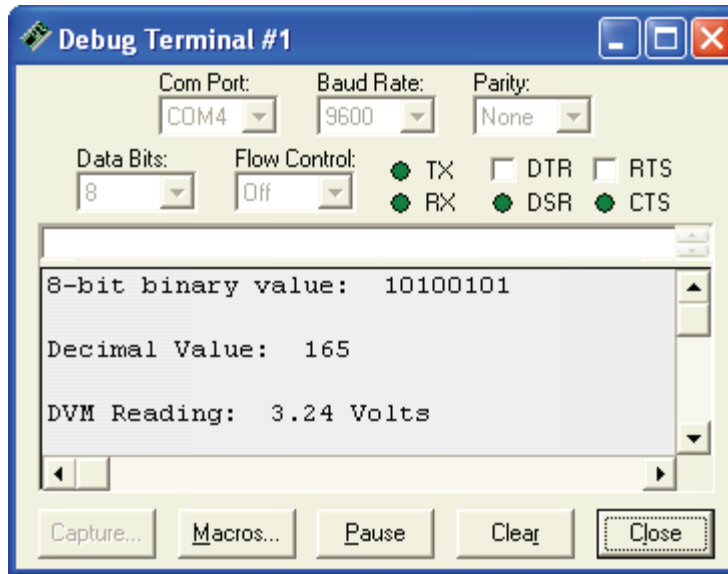
Figure 3-9
Debug Terminal Output for Program Listing 3.1, Revision 4.

> ! As soon as you're sure your program works right, save it as P3_1R4.bs2. We will add to both the code and circuit in the next experiment.

About the Code

To round off to the nearest hundredth, we need to know the digit in the thousandth's place. Using the rules of long division, we can simply set a new variable, v3 equal to the remainder from the calculation for **v2**, and divide by 255 again.

```
v3= 100 * R // 255
v3 = 10 * v3 / 255
```

Instead of using another variable, **v3** is simply redefined in the second line. The value of **v3** to the right of the equals sign is the one calculated in the first line. The value of **v3** to the left of the equals sign is the redefined value, which is ten times the old **v3**, divided by 255.

This process could be repeated over and over again to get the digit in the ten-thousandth's place, the hundred-thousandth's place, and so on.

Once the digit in the thousandth's place is known, the rules for rounding apply as follows:

- If the digit in the thousandth's decimal place is less than 5, don't add 1 to the hundredth's decimal place.
- If the digit in the thousandth's decimal place is equal to or more than 5, add 1 to the hundredth's decimal place.
- In either case, truncate everything after the hundredth's place.

Since the value **v2** is already truncated, we just need code for deciding whether or not to add 1 to the hundredth's place. It turns out to be a decision on whether or not to add 1 to **v2** as shown below.

```
IF (v3 >= 5) THEN v2 = v2 + 1
```

Since the value in the ones place is stored in a different value, we need to check and see if adding one to the hundredth's place increments that value. Without this code, 3.996 would round to 3.00 instead of 4.00.

```
IF (v2 >= 100) THEN
  v = v + 1
  v2 = 0
ENDIF
```

Save this code, and if possible, leave the circuit as it is because we can use this DVM to take measurements on the circuit we build in Chapter #4.

Resolution

The BASIC Stamp is now programmed to accurately calculate the voltage associated with the ADC0831's binary output, and the calculation is accurate to the hundredth's decimal place. Although sources of calculation error have been eliminated, there is another source of error that caused by the resolution limitation of the A/D converter.

The A/D converter chip we are using is capable of 256 binary values. This means that each measured voltage gets rounded to one of 256 discrete values. The step size is the

amount of the voltage range covered between each of these discrete values. Since the first value is zero, there are 255 voltage steps. The step size is given by:

$$\text{Step Size} = \frac{5\ \text{Volts}}{255\ \text{steps}} = 0.0196\ \text{Volts/step} \cong 0.02\ \text{Volts/step}$$

With this in mind, each time you adjust the pot, the converter comes close to approximating the analog value, but it's not exact because of the resolution constraints. So, there is still some uncertainty at the hundredth's decimal place. In some applications, the uncertainty is stated along with the measurement. Assuming the ADC0831 rounds at the half way point, we can use this convention to read the voltage from Figure 3-9 as "3.24 volts plus or minus 0.01 volts."

Higher resolution converters are available, such as 12 and 16 bit (and higher), but because of their higher resolutions, they come with a higher cost as well. The improvement in resolution is significant. As mentioned before, a 12-bit converter will give you a resolution of 4095 steps. This results in 5 volts / 4095 steps, or one step for every 0.0012 volts. A 12-bit converter typically costs more money than an 8-bit converter. There is also a cost in terms of the amount of memory the measurement takes, (12 as opposed to 8 bits) and the amount of processing it takes to get each measurement (13 clock pulses instead of 9).

Calibration

What if the power supply on the Board of Education or HomeWork Board only supplies 4.963 volts instead of 5.000 volts? The BASIC Stamp voltmeter can be calibrated using a second voltmeter known to be highly accurate. The difference between Vdd and Vss can be measured using the accurate voltmeter. Developing the code to correct for this error requires more representation of fractional values using integer math and is best left as a challenge problem.

Another item to consider if you're shooting for a high degree of precision is that different current draws on the power supply can also cause variations in power supply output voltage. This is an experiment unto itself that would also require additional equipment. As you probably guess, designing for a high degree of precision involves a number of design challenges. For the remaining experiments, the present degree of accuracy of our DVM is sufficient.

What have I learned?

On the lines below, insert the appropriate words from the list on the left.

binary resolution analog serial input 8-bit	Using the A/D converter makes it possible to process _____ information with the BASIC Stamp, a digital (_____) device. The converter used in this experiment is the ADC0831 integrated circuit, an 8-bit, _____ A/D converter. For a given analog _____ , the ADC0831 outputs an _____ binary number. The BASIC Stamp can be used to control and collect data from this A/D converter. Various programming techniques can be used to read, remember, and display this data. The digital representation of the converted analog signal is very good, but not perfect due to inherent _____ limitations of this A/D converter. A second source of error for the Stamp DC DVM can result from the fact that the power supply on the Board of Education or HomeWork Board does not necessarily supply exactly 5 volts.

Questions

1. In your own words, explain the function of an A/D converter.

2. What would be the resolution if you were to use a 16-bit A/D converter in this experiment?

3. How does the voltage divider equation relate to the wiper terminal of the pot? What would you expect the output to be if the resistors are equal? Can you prove this?

4. How do the measurements taken at the wiper terminal of the pot in this experiment differ from those taken in Chapter #1? What is gained by using the ADC0831 for measurements instead of just using a BASIC Stamp I/O pin to check the voltage?

5. Given the resolution of our 8 bit A/D converter, when the voltage on the pot is set to 3.6 volts, what decimal value will be displayed? What's the binary value?

Challenge!

1. Use another jumper wire to connect the wiper terminal of the pot to an unused BASIC Stamp I/O pin. Add a subroutine to the DVM program that monitors the state of the I/O pin set to input. Determine if the threshold voltage you were working with in Chapter #1 is indeed 1.4 volts.

2. Write a program that will monitor the analog value of the 10 kΩ potentiometer, and alert you to the fact that it has gone beyond a certain pre-set limit.

3. Write a program and build a circuit that creates a "safety zone" between 1.0 volts and 2.0 volts. If the analog voltage goes outside of these boundaries, an LED blinks.

4. Draw the complete schematic for Challenge #3, and modify the program so that the LED is only on when the voltage as set on the pot is exactly 2.0 volts.

5. Assume the power supply on your Board of Education or HomeWork Board supplies 4.960 volts. Develop a subroutine to adjust the voltage measurements to this scale.

Why did I learn it?

There is a wide variety of electronic applications where analog signals are measured and digital devices are used to process the analog signal data. In this experiment, we used a BASIC Stamp and A/D converter combination to construct a digital DC voltmeter. We'll be using the BASIC Stamp DC-DVM in several of the remaining experiments. There is a surprising variety of uses for such a device as you will discover with each new experiment.

How can I apply this?

In developing the digital voltmeter, the process of sampling voltage, converting it, and processing it in digital form was introduced. An example of another use for an A/D interface is the digital sampling of the analog signal from a microphone for digital recording purposes. Another example that could make use of the circuit we built is a door sensor. Our potentiometer could be attached to a door hinge, and the analog information could be used to monitor how far the door is open. This circuit could in turn be incorporated into a larger system that controls how far open the door swings.

The field of analog to digital conversion is an industry in itself. There are semi-conductor manufacturing companies that specialize solely in creating A/D conversion chips and systems. Whether you'd like to design at the component level, or at the integrated circuit level, there will always be a need for creative analog interfacing – simply because the world isn't black and white (binary), it's all the colors in-between as well (analog).

Chapter #4: Basic Digital to Analog Conversion

BUILD A RESISTIVE LADDER NETWORK

Digital to analog conversion (D/A conversion) is, for the most part, the reverse of A/D conversion. With A/D conversion, we started with a continuous voltage range at the converter's input. The A/D converter rounded to the nearest voltage step and sent a binary output indicating which step it measured.

D/A conversion starts with a binary number as the input, and the output is a voltage step. While the A/D process starts with an analog input and ends with a binary output, the D/A process starts with a binary input and ends with a voltage step for an output. It's not a true analog value that varies continuously; it's a discrete voltage that varies in steps.

The term resolution was introduced at the end of Chapter #3. Since a D/A converter's output always going gets rounded to a voltage step (a discrete voltage value) it's important to pick the right resolution for your D/A converter. Remember that with higher resolution comes higher precision, but it typically comes at the price of greater expense, more memory, and more processing steps.

The number of voltage levels a D/A converter can produce is given by how many counting numbers you can get from the number of binary bits in the resolution. We can use the combinations equation again to figure this out.

$$\text{combinations} = 2^{\text{bits}}$$

The D/A converter we will use in this experiment has 4-bit resolution, so the number of output voltage levels for the converter will be:

$$\text{combinations} = 2^{\text{bits}} = 2^4 = 16$$

In Chapter #3, we used an integrated circuit, which performed the A/D conversion. In this experiment, we'll build a D/A converter using resistors. It's called a resistive ladder network, and adding or removing resistors can be done to change the resolution of the converter. With a resistive ladder network, if you start with a 4-bit converter, and you want to increase the resolution by 1-bit, all it takes is two extra resistors added to the network.

In this chapter, we will build a resistive ladder network and program the BASIC Stamp to make the network do D/A conversion. PBASIC will be used to program the BASIC

Stamp to send the resistive ladder network sets of binary voltage levels. These sets of binary voltages are converted by the resistive ladder network to discrete output voltages.

The DVM from Chapter #3 will be used to measure the converter's output voltages. Measuring all the D/A converter's output levels is called a voltage sweep. We will use PBASIC to automate our DVM to perform the entire voltage sweep. That way, the D/A converter's output voltages can be measured without manually repeating each measurement.

Parts Required

Gather these parts from your parts kit and let's get started:

(6) 2 kΩ resistors
(3) 1 kΩ resistors
(1) ADC0831 A/D Converter
(1) Red LED
(1) 270 Ω resistor
(1) LM 358 op-amp

The resistive ladder network for this experiment is shown in shown in Figure 4-1. The name comes from the fact that the resistor network in the schematic resembles a ladder. It's definitely an inexpensive alternative compared to an integrated circuit digital to analog converter (D/A converter or DAC). A few resistors are a fraction of the cost of an integrated circuit.
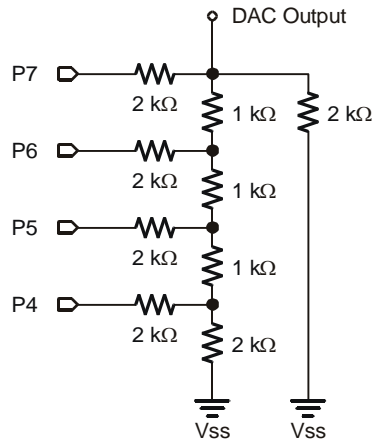
Figure 4-1
Resistive Ladder D/A Converter.

This resistive ladder network can be used as a D/A converter. The binary number input is sent in parallel as 4 bits across 4 separate data lines, P4 through P7. As long as the value of all four bits are present at the same time, the output of the D/A converter output will be the intended discrete voltage value.

This being the case, why doesn't everybody use resistive ladder networks for A/D and D/A conversion? The resistive ladder network is also used in many A/D and D/A integrated circuits, such as the ADC0831. The resistors used in integrated circuits are microscopic implants on the surface of a silicon wafer. One advantage of IC converters is that they have a high degree of accuracy. Another advantage an IC has is extra built-in circuitry similar to the voltage follower we built in Chapter #1

Build It

Build the circuit as shown in Figure 4-2. Pay careful attention to the values of the resistors as well as what each resistor connects to. If your circuit from Chapter #3 is still intact, just remove the potentiometer and build the resistive ladder network near the sockets for pins P4 through P7. The input lead to the DVM, which was connected to the wiper terminal on the potentiometer, should now be connected to the output of the D/A converter. Be sure to pay close attention so that the resistors don't touch each other except where they're supposed to on the breadboard nodes. (Note that we are setting aside the LM 358 op-amp and red LED for the time being.)
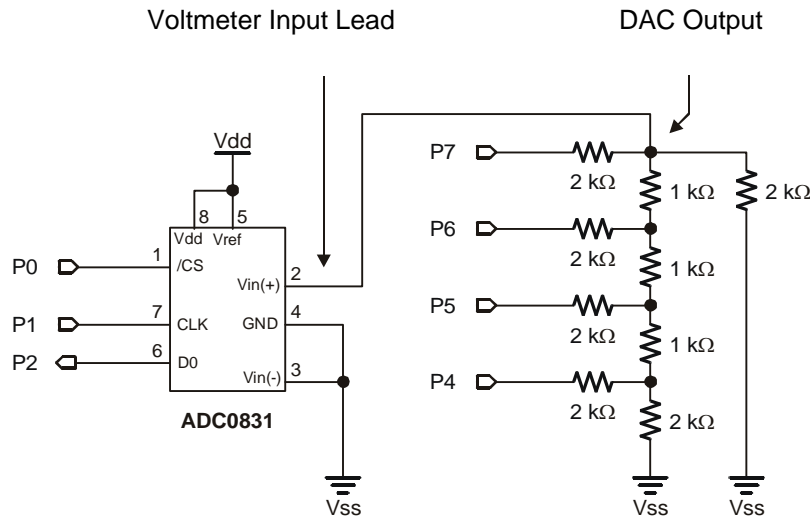
Voltmeter Input Lead   DAC Output

Figure 4-2
Schematic.

The DC voltmeter from Chapter #3 is connected to the output of the resistive ladder network D/A converter.

Program It

Not only can we use the DVM to measure the output voltage from the D/A converter, we can automate the testing process to measure all 16 the D/A converter's output voltage levels. This might not be a big deal for just 16 measurements, but just imagine trying to test all 4096 voltage steps on a 12-bit converter!

With some relatively simple additions to the code from Chapter #3, which was saved as file P3_1R3.bs2, we can control both devices. PBASIC can be used to instruct the BASIC Stamp to send an output signal to the D/A converter. The code for this will be added to the final version of Program Listing 3.1. This way we can use our DVM from this experiment to measure the D/A converter's output

Program Listing 4.1 is shown below. It's the final revision of Program Listing 3.1 with a subroutine labeled **DAC** added to send binary voltages to the D/A converter. There are also a few additional changes that are pointed out using comments such as '□ which means add this line and 'Δ which shows the lines that have been changed.

If you saved the program listing from Chapter #3, add and modify the code for this experiment, and save the program under the name P4_1R0.bs2. If you do not have the code from Chapter #3, enter the entire program listing below using the BASIC Stamp

Editor and make sure to save it for future use. When the circuit is built and the code is entered and saved, run Program Listing 4.1, and let's see how it works.

4

```
' -----[ Title ]-------------------------------------------------------
' Basic Analog and Digital - PL4_1R0.bs2
' Digital Voltmeter (DVM). D/A Converter Added
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]------------------------------------------------
adcBits          VAR     Byte
v                VAR     Byte
r                VAR     Byte
v2               VAR     Byte
v3               VAR     Byte
n                VAR     Nib

' -----[ Initialization ]----------------------------------------------
CS               PIN     0
CLK              PIN     1
DataOutput       PIN     2

DEBUG CLS                                          'Start display.

' -----[ Main Routine ]------------------------------------------------
DO
  GOSUB DAC
  GOSUB ADC_Data
  GOSUB Calc_Volts
  GOSUB Display
LOOP

' -----[ Subroutines ]-------------------------------------------------
DAC:
  n = 11

  OUTPUT 7
  OUTPUT 6
  OUTPUT 5
  OUTPUT 4

  OUT7 = n.BIT3
  OUT6 = n.BIT2
  OUT5 = n.BIT1
  OUT4 = n.BIT0
RETURN

ADC_Data:
  LOW CLK
  LOW CS
```

```
  PULSOUT CLK, 210
  SHIFTIN DataOutput,CLK,MSBPOST,[adcBits\8]
  HIGH CS
RETURN

Calc_Volts:
  v = 5 * adcBits / 255
  r = 5 * adcBits // 255
  v2 = 100 * R / 255
  v3 = 100 * R // 255
  v3 = 10 * v3 / 255
  IF (v3 >= 5) THEN v2 = v2 + 1
  IF (v2 >= 100) THEN
    v = v + 1
    v2 = 0
  ENDIF
RETURN

Display:
  DEBUG HOME, CR, CR, "Decimal value to DAC:  ", DEC2 n
  DEBUG CR, CR, "Binary value to DAC:  ", BIN4 n
  DEBUG CR, CR, "DVM Reading:  ", DEC1 v, ".", DEC2 v2, " Volts"
RETURN
```

The Output

Given perfect resistor values, the measurement would be 2.20 volts. The resistors used in this sample have a 10% tolerance. This means that the measured resistance for each resistor should have a value within ±10% of what it's supposed to be. Because of this, we can expect the output to be slightly different than what's expected, such as the measured value in Figure 4-3.

4
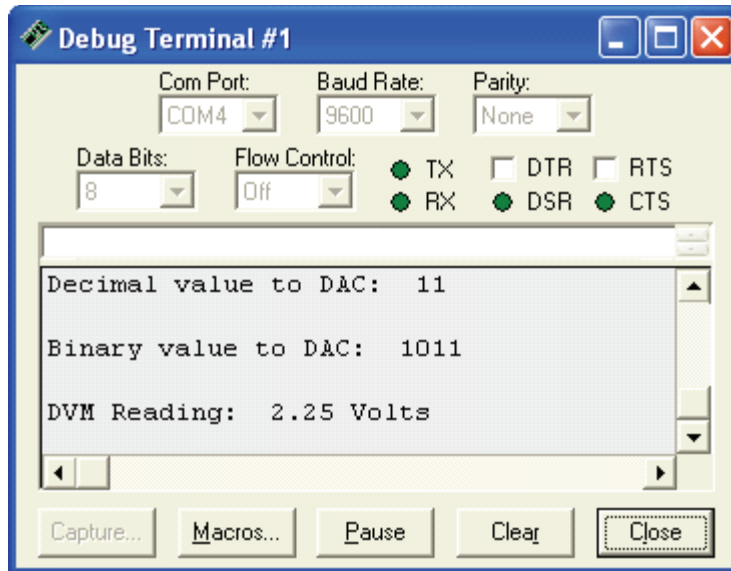
Figure 4-3
Debug Terminal
Output for Program
Listing 4.1.

About the Code

As mentioned, this code started with the final revision of Program Listing 3.1. The comments were updated to indicate that this is now Program Listing 4.1. In the third comment is a reference to indicate that a function was added to the DVM that does D/A conversion.

```
' -----[ Title ]---------------------------------------------
' Basic Analog and Digital - PL4_1R0.bs2              'Δ
' Digital Voltmeter (DVM). D/A Converter Added        'Δ
' {$STAMP BS2}
' {$PBASIC 2.5}
```

A nibble size variable **n** was added to the declarations section, and it will be used to store the binary value for the D/A converter.

```
n      VAR    Nib            '☐
```

A **GOSUB** command was added to the main routine that sends the program to the **DAC** subroutine.

```
GOSUB DAC                    '☐
```

This is the start of the digital to analog conversion (DAC) subroutine, so it's descriptively labeled **DAC**. The value of **n** is set to 11. This means that the output should be **n** steps above 0 on an output scale of 0 to 16. The value of **n** can be changed to specify voltage.

```
DAC:                            '□
  n = 11                        '□
```

The BASIC Stamp I/O pins connected to the D/A converter are set to output. These commands are normally found in the Declarations section. If they were placed in the declarations section, the program would run faster because these commands would only be done once at the beginning of the program. Instead, they are executed each time the subroutine is run. The reason they were placed in the subroutine is to make it easier to present some new PBASIC techniques.

```
    OUTPUT 7                    '□
    OUTPUT 6                    '□
    OUTPUT 5                    '□
    OUTPUT 4                    '□
```

Next, the BASIC Stamp's parallel binary output is sent to the D/A converter. We are using the same command for sending outputs that was used in Chapters #1 and #2, but there is a new feature added. The variable **n** has an extension to indicate which bit in the nibble value is being used. For example, the command **OUT7=n.BIT3** sets the output value of pin P7 equal to the value of bit 3 in the nibble variable **n**. Since we set the value of **n** to 11, the binary value of **n** is 1011. Bit 3 is the leftmost bit of the binary number, so it's 1, which means the output value of P7 is set high. Also, when **n = 11**, P6 is set low, P5 is set high, and P4 is set high.

```
    OUT7 = n.BIT3               '□
    OUT6 = n.BIT2               '□
    OUT5 = n.BIT1               '□
    OUT4 = n.BIT0               '□
```

That's all it takes for programming digital to analog conversion using a resistive ladder network. The RETURN command sends the program back to the command immediately following the GOSUB DAC command in the main: routine.

```
    RETURN                      '□
```

The first two lines in the **Display** subroutine were modified to show the decimal and binary values of **n**, the 4-bit binary value used D/A output value.

```
    DEBUG HOME, CR, CR, "Decimal value to DAC:  ", DEC2 n     '△
    DEBUG CR, CR, "Binary value to DAC:  ", BIN4 n            '△
```

Modify the Code

If the D/A converter works as we expect, each time the value of n is incremented by 1, the D/A output should increment by 0.2 volts. Try starting with n=0 by modifying the value of **n** in the **DAC** subroutine:

```
n = 0                                              '△
```

Then run the program again with n=1:

```
n = 1                                              '△
```

Then change n to 2 and run the program a third time:

```
n = 2                                              '△
```

Continue for each value up to **n=15.**

Figure 4-4 shows a sample measurement of a resistive ladder D/A conversion with the value of n set to 15. Remember that the 10% resistors lead to some error in the output. In Chapter #3, we were interested in programming our DVM for accurate calculations to the nearest hundredth of a volt. In this experiment, anything within 10% of the expected value is fine. So your end output when **n=15** might be as low as 2.7 volts or as high as 3.3 volts. If the errors are larger than that, double check to make sure that a 1 kΩ resistor didn't get swapped with a 2 kΩ somewhere in the resistive ladder.
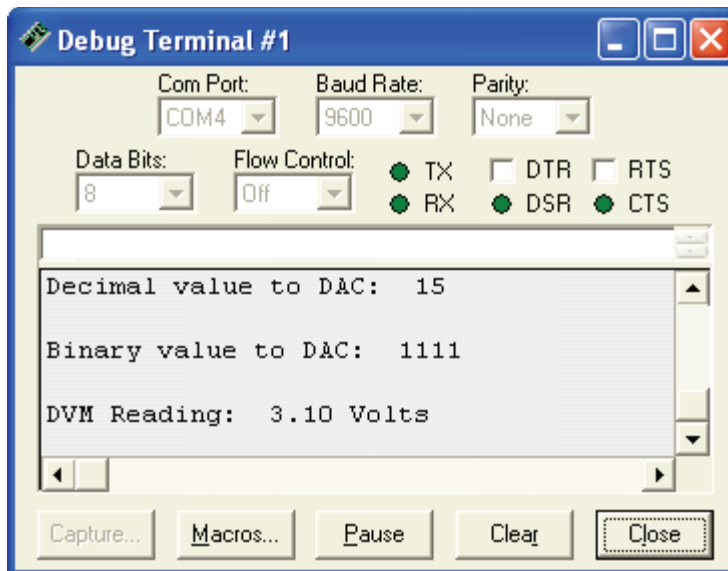
Figure 4-4
Debug Terminal
Output for Program
Listing 4.1, Revision
1.

Addressing

Until now, we've been addressing each of the I/O lines one at a time. This works well whenever you need to have control over the status of a particular control line. For example, a single LED is easily addressed by the individual I/O pin to which it is connected using OUTp=value, where p is a pin number between 0 and 15, and value is either 0 or 1.

> **Addressed:** When an I/O pin is addressed, it means that a value has been written into a particular place in the BASIC Stamp module's RAM. For example, a particular memory location can be set high to set a given I/O pin to be an output. Another address might be used to set the pin high or low.
>
> Performing these operations one bit at a time is not always efficient. The memory locations are adjacent to each other so that the operations can be performed 1 nibble (4 bits) at a time or one byte (8 bits) at a time, or even a word (16 bits) at a time.

Since the I/O pins (P4 through P7) are used as outputs in this experiment, it would be easier and more efficient to have a method of addressing this group of bits. Notice that it takes four lines of code just to set up the bits as outputs, and then it takes 4 additional lines to individually set each bit. It's not a big deal now, but as time goes on and you

begin to create more complex programs, you may find yourself looking for ways to get the most out of the least code.

There are two registers that we need to set to control output on a specific group of I/O lines. The first register is called "direction". The command "output" sets the direction as an "output". Conversely, "input" sets the I/O line up as input. The second is the "data" register. If the I/O pin has been set up as an output, then this register's "data" can be set to either 0 or 1. This in turn sends either a low or high signal to that pin, and the measured output will either be 0 or 5 volts.

Certain commands in PBASIC allow us to directly address its I/O lines as a word (16 individual bits), two bytes (two sets of 8 individual bits) or as 4 nibbles (four sets of four individual bits). To modify our code, we want to address a nibble at a time, and the four bits that we're using are P4-P7. According to the BASIC Stamp Manual (which you should have a copy of by now - it's free download at www.parallax.com and printed copies are not expensive) the group "P4-P7" is called nibble "b". The next set of four (P8-P11) is nibble "c", and so on.

Let's try using this in a program and figure out how it works. Rewrite the DAC: subroutine in Program Listing 4.1 as follows:

```
DAC:
  n = 11

  DIRB = 15
  OUTB = n
RETURN
```

Figure 4-5 shows the output is identical to the previous version of the **DAC** subroutine. We did the same job with two lines of code instead of eight.
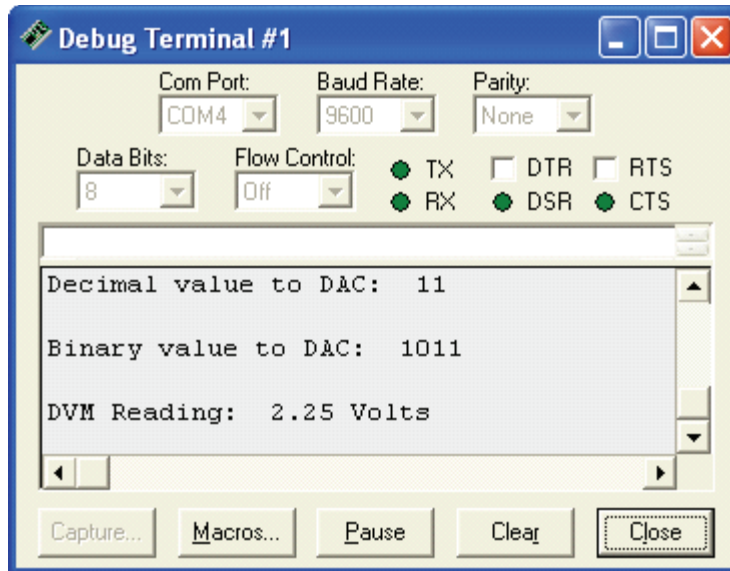
Figure 4-5
Debug Terminal
Output for Program
Listing 4.1, Revision
2.

Here is how to count from 0 to 15 using a nibble:

0 = 0000	4 = 0100	8 = 1000	12 = 1100
1 = 0001	5 = 0101	9 = 1001	13 = 1101
2 = 0010	6 = 0110	10 = 1010	14 = 1110
3 = 0011	7 = 0111	11 = 1011	15 = 1111

When nibble b is selected using **DIRB**, each bit in the number **DIRB** is set equal to a value corresponding to a data direction as shown below:

Bit in nibble B	3	2	1	0
I/O pin	P7	P6	P5	P4

If we used the command "**DIRB = 4**" the following direction register bits would be set like this:

Bit value	0	1	0	0
I/O pin	P7	P6	P5	P4

This would result in I/O pin P6 being set as an output, and all the other pins (P0, P1, P3) set as inputs. Hence the command **DIRB=15** (by virtue of the fact that all four bits are a "1") sets up each of the I/O lines as outputs.

Sweeping the value of **n** from 0 to 15 should result in the same values as before.

A truly powerful aspect of using this method of addressing is that we can use PBASIC to count up and down or access values from a lookup table to automatically address the I/O pins. The result is that we can program the BASIC Stamp to more effectively control the D/A converter output.

Modify the code labeled **'Start Display** in Program Listing 4.1. First, modify the **DEBUG CLS** command, and then add a second line as shown.

```
'Start display                                          '□
DEBUG CLS, "DAC Nibble Values", CR                      '△
DEBUG "Decimal      Binary DVM", CR                      '□
```

Modify the main subroutine as shown:

```
' -----[ Main Routine ]-------------------------------------
FOR n = 0 TO 15
  GOSUB DAC                        '□
  GOSUB ADC_Data
  GOSUB Calc_Volts
  GOSUB Display
NEXT
STOP
```

Delete the line that sets the value of **n** in the **DAC** subroutine. Once that's done it should look like this:

```
DAC:
  DIRB = 15
  OUTB = n
RETURN
```

Also modify the **Display** subroutine as shown.

```
Display:
  DEBUG DEC2 n, "    ", BIN4 n, "  "
  DEBUG DEC1 v, ".", DEC2 v2, " Volts", CR
RETURN
```

Figure 4-6 shows the output. Try that with a hand held voltmeter and you'll begin to see the usefulness of combining the BASIC Stamp with analog interfaces. Imagine trying to test all 4096 levels of a 12-bit DAC one at a time!
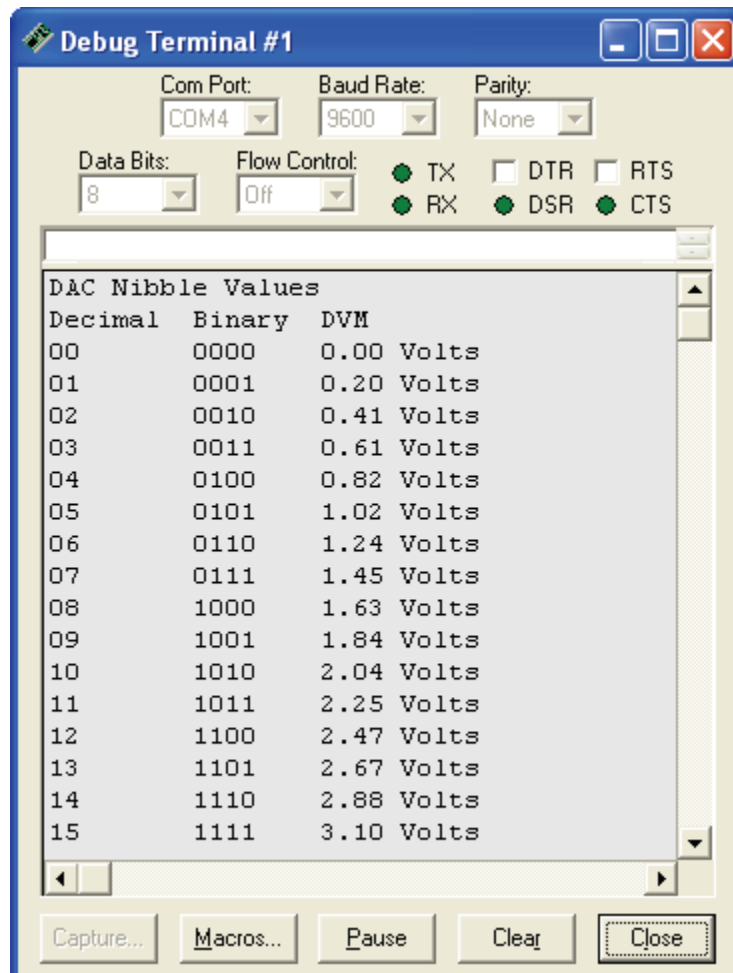
```
Debug Terminal #1                          [_][□][X]

        Com Port:    Baud Rate:    Parity:
        COM4  ▼      9600   ▼      None  ▼

   Data Bits:   Flow Control:   ● TX   □ DTR  □ RTS
    8      ▼     Off      ▼     ● RX   ● DSR  ● CTS

  ┌──────────────────────────────────────────┐
  │                                           │
  ├──────────────────────────────────────────┤
  │ DAC Nibble Values                      ▲  │
  │ Decimal   Binary   DVM                 █  │
  │ 00        0000     0.00 Volts          █  │
  │ 01        0001     0.20 Volts             │
  │ 02        0010     0.41 Volts             │
  │ 03        0011     0.61 Volts             │
  │ 04        0100     0.82 Volts             │
  │ 05        0101     1.02 Volts             │
  │ 06        0110     1.24 Volts             │
  │ 07        0111     1.45 Volts             │
  │ 08        1000     1.63 Volts             │
  │ 09        1001     1.84 Volts             │
  │ 10        1010     2.04 Volts             │
  │ 11        1011     2.25 Volts             │
  │ 12        1100     2.47 Volts             │
  │ 13        1101     2.67 Volts             │
  │ 14        1110     2.88 Volts             │
  │ 15        1111     3.10 Volts           ▼ │
  │ ◄                                      ►  │
  └──────────────────────────────────────────┘

  Capture...   Macros...   Pause   Clear   Close
```

Figure 4-6
Debug Terminal Sample Output for Program Listing 4.1, Revision 3.

This is a very efficient way to collect your voltage sweep data. Looking at the data for the voltage sweep brings several things to light. First, the voltage output for the D/A converter is always a little high. Second, the error increases as the output voltage

increases. Third, the largest error is 0.1 volts. This kind of data can be exceedingly useful in electronics design, and the automated testing process is a huge time saver.

The Voltage Follower with the Op Amp

Let's use the voltage sweep to analyze what happens when the output of the D/A converter is connected to another circuit. We'll use the output of the D/A converter to drive an LED circuit. First we'll connect the D/A converter output directly to the input of the LED circuit. Then we'll use the voltage follower as an intermediate step between the D/A converter output and the LED circuit input.

4
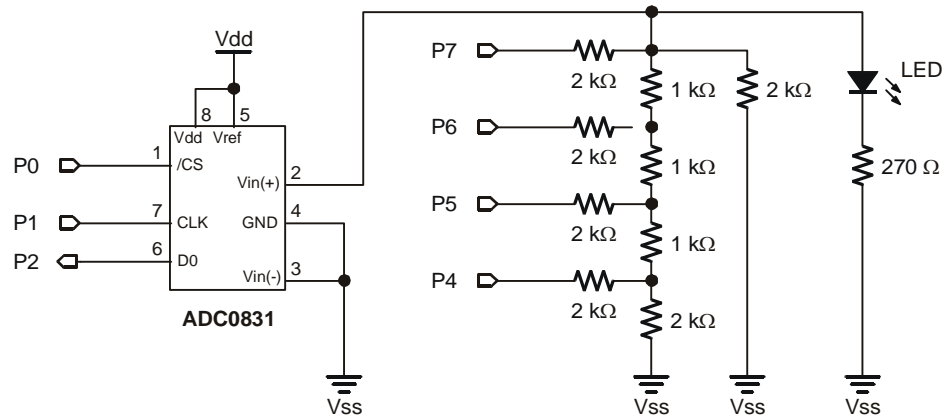
Figure 4-7 D/A Circuit without a Buffer.

Table 4-1:D/A Converter Output Without Buffer		
Decimal	**Binary**	**DVM (volts)**
00	0000	
01	0001	
02	0010	
03	0011	
04	0100	
05	0101	
06	0110	
07	0111	
08	1000	
09	1001	
10	1010	
11	1011	
12	1100	
13	1101	
14	1110	
15	1111	

Figure 4-7 shows the D/A converter with an LED circuit added. The LED circuit is the "load" that the D/A converter must "drive". Run a voltage sweep on this circuit, and fill in the table below.

Then try the voltage sweep using the output of the voltage follower in as shown in Figure 4-8. Fill out the same table of voltage sweep information for this second circuit, and compare the two. The voltage follower in Figure 4-7 is referred to as a buffer. The LED circuit has no voltage follower to separate it from the resistive ladder network. The LED is an "unbuffered" load for the D/A converter.
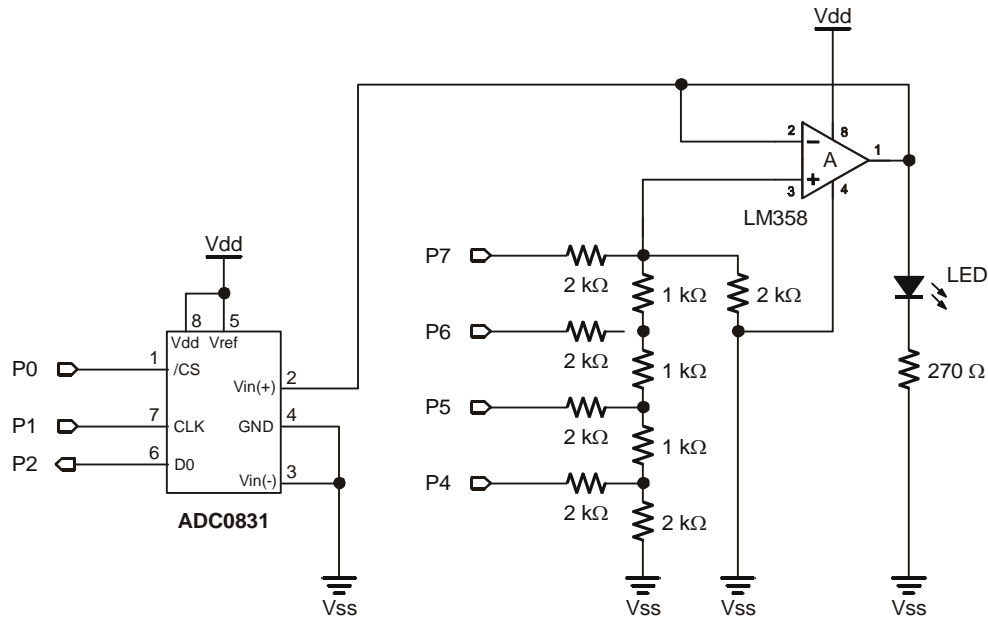
Figure 4-8 D/A Circuit With a Buffer

Comparing the two tables, it should be fairly clear that the buffer (voltage follower) eliminated a problem caused by connecting the LED circuit directly to the D/A converter's output. The voltage of the D/A output without a buffer reached a maximum well below the D/A converter's 3 volt maximum. On the other hand, the buffered output went all the way to 3 volts with no problem.

Table 4-2:D/A Converter Output with Buffer		
Decimal	**Binary**	**DVM (Volts)**
00	0000	
01	0001	
02	0010	
03	0011	
04	0100	
05	0101	
06	0110	
07	0111	
08	1000	
09	1001	
10	1010	
11	1011	
12	1100	
13	1101	
14	1110	
15	1111	

The reason for this goes back to Ohm's Law: $V = I \times R$ (voltage equals current multiplied by resistance). Each BASIC Stamp I/O pin can supply up to 20 mA of current. In the case of the resistive ladder network without the buffer, the BASIC Stamp I/O pins reached their maximum current output. Meanwhile, the resistance seen by the I/O pins stayed about the same. In other words, I x R reached a limit because I (the current) could no longer increase, and R is a fixed value. So the output voltage is equal to a current value that can't get any higher multiplied by a fixed resistance. This is why the voltage stops increasing.

Aside from isolating two circuits from each other, an op-amp in voltage follower configuration can typically supply more current than the circuit connected to its input. The name "buffer" is commonly used when a voltage follower is used to supply extra current.

The circuit from Figure 4-8 will also be used at the beginning of Chapter #5, so do not disassemble the circuit after completing Chapter #4.

What have I learned?

On the lines below, insert the appropriate words from the list on the left.

4

binary discrete bit output bit current addressing direction sweep automate	A set of _____ voltage values sent to the inputs of a resistive ladder network results in a _____ voltage level at the output. This method is an inexpensive alternative to integrated circuits. Its advantages are flexibility in resolution and low cost. Its main disadvantage is accuracy. The **.BITx** extension can be attached to a variable to select a particular _____ within a nibble, byte or word. This can be used to select bits from a binary number in memory and set _____ values equal to those bits. PBASIC can be used to address BASIC Stamp I/O pins a _____ at a time, or alternatives exist for _____ nibble, byte or word size groups of I/O pins. This allows the _____ and output values of the I/O pins to be addressed as groups of bits instead of as a single bit. A voltage _____ can be run on a D/A converter. This allows one to view the D/A converter's outputs for all possible binary inputs. The BASIC Stamp can be programmed to _____ this process and display the data in a table format. The voltage follower can be used as a buffer, which can supply extra _____ to the input of a circuit connected to the D/A converter's output.

Questions

1. "1101" is what value in the decimal number system? What voltage would you expect from the D/A converter output if you sent it this binary number?

2. What function is provided by a D/A converter?

3. What are some of the advantages and disadvantages of the resistive ladder network?

4. Why does the D/A voltage "jump" from one value to another, unlike the voltage available on a pot?

5. How does the voltage follower solve the output voltage range problem when the D/A converter's output is connected to the LED circuit's input?

Challenge!

1. Create an 8-bit "resistive ladder" D/A converter. Draw the complete schematic.

2. Write a program that will step through 256 different analog voltages. Each voltage should be present for 100 milliseconds on the voltmeter.

Why did I learn it?

There are many different "real world" circuits that require some sort of analog voltage. For example, when you listen to a compact disc, you're listening to sounds that started as analog signals from a microphone. Then the signal underwent the A/D process when it was digitized. The CD player then performs the D/A conversion using the digital information on the CD. The analog signal is amplified and played on a speaker.

When designing commercial products, it is your responsibility to determine the most appropriate (and cost effective) method to accomplish a given task. A resistive D/A converter is a very economical method to get an analog voltage from a digital device.

How can I apply this?

You've just built a variable voltage source and a digital voltmeter on the same breadboard. You also used some PBASIC programming techniques to obtain information about the converter using a voltage sweep. One application of this can be testing circuits, but there are many others. In Chapter #5, we'll see how PBASIC can be used to control the volume of tones emitted by a speaker. In Chapter #7, we'll use D/A conversion to control the brightness of an LED transmitting a signal to a photoresistor.

The applications for D/A and A/D interfaces combined with a microcontroller are only limited by one's imagination. These techniques can be applied to automated houses, irrigation systems, rocket guidance systems, and robotics to name a few. Control systems engineering is a field of study within electrical engineering that you might look into if you find designing such systems interesting.

Chapter #5: Time Varying Signals

In this chapter we'll view signals with the Stamp-O-Scope. To do this we'll use the BASIC Stamp along with the A/D and D/A circuits built in Chapters #3 and #4 to emulate the function of an oscilloscope.

The voltage output of the resistive ladder network will be varied with time, and the changes will be tracked using our oscilloscope emulator, the Stamp-O-Scope. The Stamp-O-Scope will gather its input voltage data from the ADC0831 and display it graphically in the Debug Terminal.

5

We'll view two different signals using the Stamp-O-Scope, and we'll use the D/A converter to adjust the attributes of both signals. The first signal we'll work with is called a triangle wave, and the second is called a pulse train. The LED will be used to monitor the activity of both signals.

Unlike a normal oscilloscope, the Stamp-O-Scope cannot monitor voltage signals at frequencies in the audible sound range. However, the BASIC Stamp can easily send a voltage signal in the audible sound range via the D/A converter. This signal can be used to make a speaker emit tones of adjustable pitch and volume.

The pitch and volume will be adjusted by changing two of the pulse train's properties. But first, let's get started and view these properties with the Stamp-O-Scope.

Also in this experiment, the BASIC Stamp is used to generate sine waves (see Figure 5-1) of adjustable frequency. These sine waves are used to make musical notes.

Oscilloscope: A device that measures and displays time-varying voltage signals. The oscilloscope is a common tool used by many electronics technicians and engineers to display these signals. The signals of interest often repeat themselves many times each second. The oscilloscope can be used to view the general shape of the signal, how fast it repeats itself, and the maximum and minimum voltage values of the signal as it repeats itself.
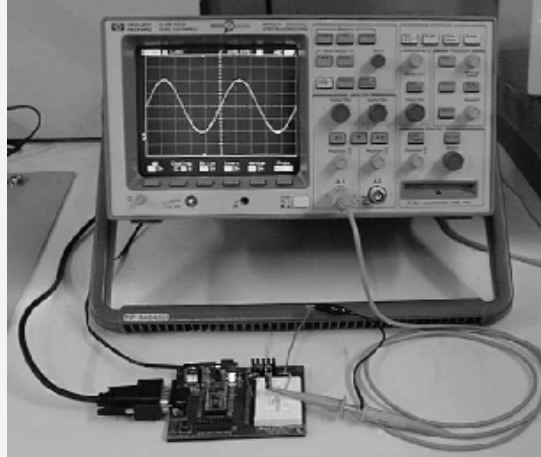

Figure 5-1
Oscilloscope displaying a sine wave output from Program Listing 5.3.

Frequency: The rate at which a periodic time varying signal repeats itself. Frequency is measured in repetitions per second. The name **hertz** (Hz) is used when referring to frequency. One hertz is one repetition per second; that is:

1 Hz = 1 repetition/second = 1/s

Parts Required

The same circuit used in Chapter #4, Figure 4-8 is used to begin this experiment. Later, the LED circuit is replaced with a piezoelectric speaker circuit. For this experiment you'll need the following parts:

(6) 2 kΩ resistors
(3) 1 kΩ resistors
(1) ADC0831 A/D Converter
(10) Jumper wires
(1) Piezoelectric speaker

The piezoelectric speaker shown in Figure 5-2 has positive and negative terminals shown on both the circuit symbol and the component. The speaker from the Analog and Digital Parts Kit has a plus (+) on its topside above the appropriate pin.
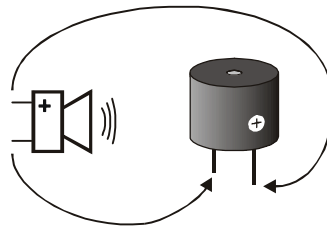

Figure 5-2
Piezo Speaker

Circuit symbol and part.

If you saved the circuit that was developed in Chapters #3 and #4, you are ready for the next step. Otherwise, rebuild the circuit shown in Chapter #4, Figure 4-8.

Program It

Start with an unmodified version of Program Listing 4.1, and make the following changes. Changed lines are shown with Δ and added lines are shown with a □.

Modify the first line as shown:

```
' Basic Analog and Digital – PL5_1R0.bs2            'Δ
```

After the **DEBUG CLS** command, insert a code block nested in a **DO…LOOP** as shown below:

```
'Start display
DEBUG CLS

DO                                      '□
  FOR n = 7 TO 15                       '□
    GOSUB Main                          '□
  NEXT                                  '□
  FOR n = 14 TO 8                       '□
    GOSUB Main                          '□
  NEXT                                  '□
LOOP                                    '□
```

Modify the Main Routine as shown. There are two changes to make. First, the **GOSUB Calc_Volts** command should be commented with an apostrophe, and second, the **DO…LOOP** command should be changed to a **MAIN: label** and a **RETURN** command.

```
' -----[ Main Routine ]-------
MAIN:                                  '∆
  GOSUB DAC
  GOSUB ADC_Data
  'GOSUB Calc_Volts                    '∆ (Comment this line!)
  GOSUB Display
RETURN                                 '∆
```

Replace the commands in the **DAC** subroutine so it looks like this:

```
DAC:
  DIRB = 15
  OUTB = n
  PAUSE 50
RETURN
```

Also replace the commands in the **Display** subroutine from Program Listing 4.1 so it looks like this:

```
Display:
  DEBUG REP " "\adcBits/4,"*", CR
RETURN
```

The Output - A Triangle Wave

Save the program as P5_1R0.bs2, and download it to your BASIC Stamp. The Stamp-O-Scope output should look like Figure 5-3. The red LED on the breadboard should be gradually getting bright and then dim again at the same pace as the triangle shaped wave varies in the Debug Terminal. The asterisk in the bottom line of the Debug Terminal indicates the current voltage measurement. The further right the asterisk appears in the Debug Terminal, the higher the measured voltage.
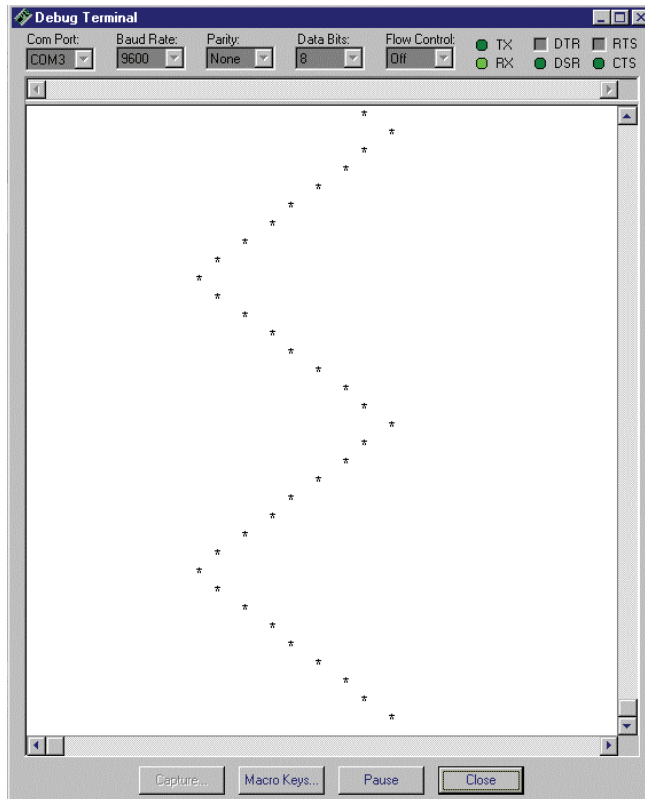
Figure 5-3
Output Sample of Time-Varying Waveform

For Program Listing 5.1 is the Stamp-O-Scope display. The waveform displayed is commonly referred to as a triangle wave.

If you were to rotate the Stamp-O-Scope display a quarter turn counter-clockwise, it would resemble the display of a normal oscilloscope. Figure 5-4 shows the Stamp-O-Scope rotated this way. It also shows three of the most basic measurements taken on an oscilloscope: amplitude, frequency, and DC offset. An oscilloscope typically displays time on the horizontal scale and voltage on the vertical scale. Time passes across an oscilloscope screen from left to right, and as the wave gets higher in the screen, the voltage is higher. Most graphs of time varying waveforms are also presented in this format.
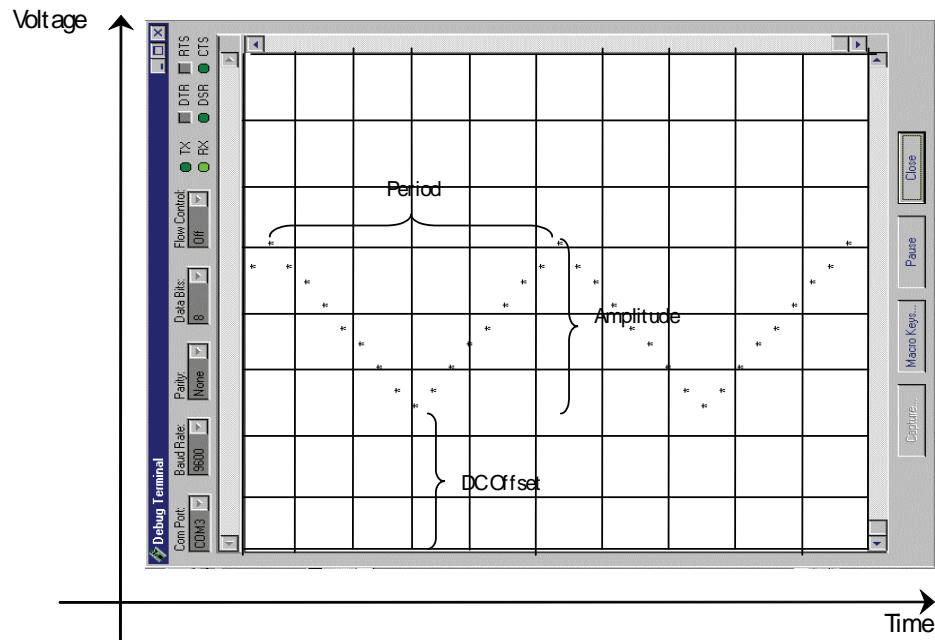
Figure 5-4: Horizontal Output Sample of Time-Varying Waveform

A more conventional way of viewing time-varying waveforms is shown in Figure 5-4. This is the way they are typically displayed using an oscilloscope. The amplitude, period and DC offset are three common measurements taken using an oscilloscope. Note how the time axis is the horizontal axis in this picture, and the voltage axis is the vertical axis

The period of a waveform is the amount of time it takes to repeat itself. In this case, that amount of time is about 1 second. The frequency is the inverse of the period, so we can use a simple equation to determine the frequency when we know the period:

$$f = 1/T$$

The term f is the frequency, measured in hertz (Hz), and T is the period, measured in seconds (s). Since the period is one second, and 1 ¸ 1 = 1, the frequency is also 1 Hz. As another example, if the period is 1/100 of a second, the frequency is 1, 1/100 = 100 Hz.

Amplitude and DC offset are two quantities related to voltage that are measured differently for different signals. The amplitude shown in Figure 5-4 is called the peak-to-

peak amplitude. The D/A converter's output is programmed for a maximum of 3 volts and a minimum of 1.6 volts. Therefore the peak-to-peak amplitude is 3 – 1.6 volts = 1.4 volts.

The DC offset is the difference between 0 volts and the waveform's minimum value. We already know the lowest point of the sample triangle wave is 1.6 volts, and that's the DC offset.

An alternative reference point for both Amplitude and DC offset is half way between the upper and lower peak. The half way point would be $(3 + 1.6)$, 2 volts = 2.3 volts. The amplitude would be 3 – 2.3 volts = 0.7 volts. The DC offset would be 1.6 + 0.7 volts = 2.4 volts.

About the Code

The **Main** routine is has been relegated to a subroutine in favor of the **DO…LOOP** code block. This code block uses a **FOR…NEXT** loop to increment and decrement up and down counts for **n**, the value sent to the D/A converter. This is why the measured output of the D/A converter increases and then decreases in the Stamp-O-Scope display.

Each time a **FOR…NEXT** loop in the **DO…LOOP** routine increments the value of **n**, it also sends the program to the **Main** subroutine.

```
DO
  FOR n = 7 TO 15
    GOSUB Main
  NEXT
  FOR n = 14 TO 8
    GOSUB Main
  NEXT
LOOP
```

The **Main** subroutine still does the same job it did in the previous experiment. The only difference is that now it's a subroutine. The **Calc_Volts** subroutine is not necessary for our Stamp-O-Scope display, so the **GOSUB Calc_Volts** command is commented. To save EEPROM memory, you could also delete the entire subroutine. Keep that in mind if you use this in a larger application and start to run out of EEPROM memory to store your program.

```
Main:
  GOSUB DAC
  GOSUB ADC_Data
  'GOSUB Calc_Volts
```

```
      GOSUB Display
    RETURN
```

This is the simplest form of the **DAC** subroutine covered in Chapter #4.

```
DAC:
  DIRB = 15
  OUTB = n
  PAUSE 50
RETURN
```

The **DEBUG** command in the **Display** subroutine is what places the asterisks in the location in the Debug Terminal corresponding to the magnitude of the measured voltage at that instant. The higher the voltage measurement, the further to the right the cursor is printed in the Debug Terminal.

The **DEBUG** modifier **REP** is what causes the " " (single space) character to be printed over and over again. In this case it's repeated **adcBits/4** (the value of **adcBits** divided by four) times. After all those spaces, an asterisk and carriage return are printed so that the next asterisk for the next voltage measurement will appear on the next line.

```
Display:
  DEBUG REP " "\adcBits/4,"*", CR
RETURN
```

How many spaces before the asterisk is printed? Let's say we are measuring 3 volts, the highest voltage level the DAC is supposed to output. Remember that **adcBits** is the number sent by the ADC0831, and it's between 0 and 255 on a 5-volt scale. Therefore when 3-volts is measured:

$$adcBits = (3/5) \times 255 = 153$$

The number of spaces is:

adcBits/4, which is $153/4 = 38.25 = 38$ rounded to zero decimal places.

The maximum voltage will place an asterisk 39 spaces across the Debug Terminal.

Try changing the value of **n** in the **FOR…NEXT** loop in the code block. In this case, it will change both the amplitude and the period of the triangle wave. This is an example where the limits of the **FOR…NEXT** loop are set to the voltage output limits of the resistive ladder network D/A converter.

```
DO
  FOR n = 1 TO 15                              '△
    GOSUB Main
  NEXT
  FOR n = 14 TO 0                              '△
    GOSUB Main
  NEXT
LOOP
```

Note that both the amplitude and period of the waveform increased. This is because the BASIC Stamp is programmed to pause for a fixed amount of time at each output voltage level. The pause statement is part of the **DAC** routine.

The Pulse Train

The next waveform we'll examine is called the pulse train, though a better name for it might be a rectangle wave. For a pulse train, the amplitude is the voltage between the high and low voltage levels, and the DC offset is the voltage between 0 volts and the low part of the signal. The period and frequency are the same for a pulse train as they are for a triangle wave. So the period is still the amount of time the waveform takes to repeat itself, and the frequency is still the number of times the waveform repeats itself in a second. The frequency is also still the inverse of the period.

The **DO…LOOP** code block can be modified to make a pulse train that has the same frequency of the original triangle wave. The variable v3 is not used right now because we are skipping the **Calc_Volts** subroutine. It can be used to for counting up and down. The value of **n**, which controls the D/A converter output, can be set to a high value and a low value. Let's start with a high value of 3 volts and a low value of 0 volts. Modify the **DO…LOOP** code block as shown below, and save the modified program as P5_1R1.bs2

```
DO
  FOR v2 = 0 TO 15                    '△
    n=15                              '□
    GOSUB Main
  NEXT
  FOR v2 = 15 TO 0                    '△
    n=0                              '□
    GOSUB Main
  NEXT
LOOP
```

Run the program and view the pulse train. Then try to change the various adjustments to the code and view what happens to the pulse train. Is it really square? You can adjust the

value of **n** in the first **FOR…NEXT** loop to change the amplitude. You can change the values of **n** in the second **FOR…NEXT** loop to get various DC offsets. Just remember that **n** has to be between 0 and 15.

Changing the limits of the first **FOR…NEXT** loop changes the amount of time the signal stays high. In other words, you can change the pulse width buy adjusting the *StartValue* and *EndValue* arguments of the **FOR…NEXT** loop. Changing these values in the second **FOR…NEXT** loop adjusts the duration of the low portion of the signal.

By changing both the duration of the low and high portion of the signal proportionately, you can change the frequency without affecting the duty cycle. By changing only one of the two sets of limits, you change the duty cycle as well as the frequency of the signal.

> **Pulse width:** A pulse train goes high-low-high-low-high-low... the pulse width is the amount of time that a pulse train stays high in between the lows
>
> **Duty cycle:** The duty cycle is the ratio of time the signal stays high to the amount of time it takes the signal to repeat itself. In other words it's the pulse width divided by the period of the signal.

Now let's run the frequency of this pulse train up into the audible sound frequency range and listen to the result of adjusting these signal characteristics. Remove the ADC0831 from the circuit. Also, replace the LED circuit with the piezoelectric speaker circuit shown in Figure 5-5.
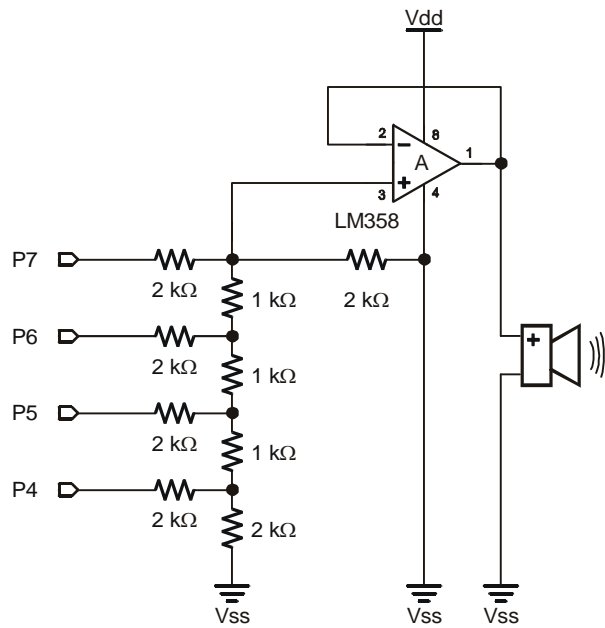
Figure 5-5
Piezoelectric Speaker Circuit

Connected to the buffered output of the D/A converter.

Enter this program into the BASIC Stamp Editor. This is not a modification to the previous programs. Also, the output is the piezoelectric speaker and there is no Debug Terminal output. Save this program as P5_2RO.bs2.

```
' Basic Analog and Digital - PL5_2R0.bs2
' D/A Converter Making Sound, n changes volume, m changes frequency
' {$STAMP BS2}
' {$PBASIC 2.5}

n        VAR     Nib
m        VAR     Word

DIRB=15

n = 15
m = 500

DAC:
  OUTB = n
  PAUSE m
  OUTB = 0
  PAUSE m
GOTO DAC
```

First, run the program as shown. The piezoelectric speaker should tick about every half-second indicating the transitions between the high and low signal.

Next, change the value of `m` from 500 to 100, and note that the ticking is much quicker. Also try the values of 50, 20, 10, 5, and 1. Experiment a bit.

The value of `m = 1` should cause the piezoelectric speaker to emit a fairly clear tone.

Next, let's change the value of the amplitude by varying the value of `n.` First, try the value of `n = 1`. The tone should be the same, just much quieter. Next try the value of `n = 5`. The tone is still the same pitch, just louder. For values of `n = 10` and `n = 15`, the volume should increase two more notches.

In this application, we caused the piezoelectric speaker to make sound by applying pulses using the BASIC Stamp. We steadily increased the frequency by decreasing the amount of time the signal paused in its high and low state, effectively decreasing the signal's period. Since frequency is the inverse of period, decreasing the period increased the frequency. When the frequency was increased the pitch of the tone also increased. Next we changed the amplitude of the pulse train by changing the output value for the high signal. This changed the volume of the sound made by the piezoelectric speaker.

Causing variations in air pressure creates sound. Any sound can be represented by some sort of time-varying waveform. To cause the piezoelectric speaker to make sound, we sent a time-varying periodic voltage signal to its input. The speaker converted the signal into motion of a membrane inside the plastic case. As the membrane vibrated, it caused air pressure variations. Our eardrum (another membrane) senses these pressure variations, and we, in turn, heard the tones.

The pulse train we sent to the piezoelectric speaker represented variations in air pressure. We varied the amplitude of the signal to increase the amplitude of the air pressure variations, which in turn increased the volume of the tone. The frequency of the pulse train was varied to change the frequency of the pressure variations. This in turn changed the pitch of the tone.

The Sine Wave and Pulse Width Modulation (PWM)

The BASIC Stamp has a built in function to generate tones similar to those you hear when you strike a tuning fork. Disconnect the terminal of the 100 Ω resistor connected to the buffer output (LM358 pin 1), and connect it to P10 on your board as shown in Figure

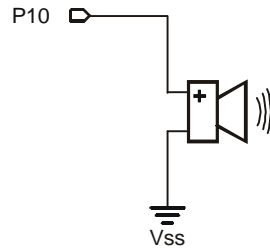5-6. These three circuit elements are all that's necessary to connect to the BASIC Stamp to play tunes and tones.

P10

Figure 5-6
BASIC Stamp Music
Circuit

Vss

5

The piezoelectric speaker shown in Figure 5-6 works like a rechargeable battery that charges and discharges over short periods of time. By applying lots of pulses to the circuit, the voltage output can actually be shaped into a signal.

These pulses can be varied in width and applied more or less frequently. Depending on the pulse widths and frequencies, the capacitance of the piezospeakers gains or loses voltage at varying rates. That's the gist of how pulse width modulation (PWM) works, and it can be used to shape voltage signals. Figure 5-7 shows a sample of a PWM signal applied to the input of an RC circuit as well as a sine wave output.
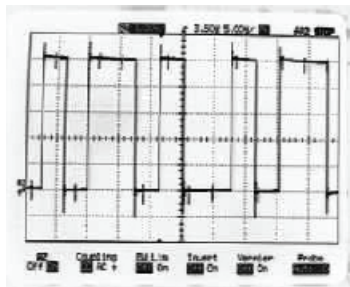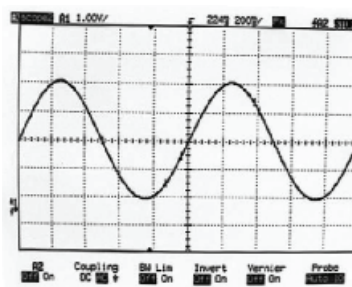
Figure 5-7
Oscilloscope
Samples

*Of a PWM
signal and a
sine wave.*

PWM Signal   Sine Wave

Sine waves can be used to represent musical notes. When a sine wave is played on the piezoelectric speaker, the sound has a significantly improved quality. The variations in air pressure caused by a tuning fork are represented by a sine wave. Program Listing 5.3

defines an octave of musical notes and their corresponding frequencies. A list of these musical notes is then played using **FREQOUT** commands. Let's try it.

Reprogram the Circuit for Musical Notes

```
' Basic Analog and Digital - PL5_3R0.bs2
' Charge!!!
' {$STAMP BS2}
' {$PBASIC 2.5}

C               CON     2093
CSharp          CON     2218
D               CON     2349
DSharp          CON     2489
E               CON     2637
F               CON     2794
FSharp          CON     2960
G               CON     3136
GSharp          CON     3322
A               CON     3520
ASharp          CON     3729
B               CON     3951


DEBUG CLS
FREQOUT 10, 100, C
DEBUG HOME, "Frequency of PWM sine wave output is ", DEC4 C, " Hz.", CR
FREQOUT 10, 100, D
DEBUG "Frequency of PWM sine wave output is ", DEC4 D, " Hz.", CR
FREQOUT 10, 100, DSharp
DEBUG "Frequency of PWM sine wave output is ", DEC4 DSharp, " Hz.", CR
FREQOUT 10, 200, G
DEBUG "Frequency of PWM sine wave output is ", DEC4 G, " Hz.", CR
FREQOUT 10, 100, DSharp
DEBUG "Frequency of PWM sine wave output is ", DEC4 DSharp, " Hz.", CR
FREQOUT 10, 200, G
DEBUG "Frequency of PWM sine wave output is ", DEC4 G, " Hz.", CR
```

The Output

The output sample shown in Figure 5-8 is the last of the eight that are played. The tones sound significantly better than the equivalent pulse trains, which in general have more of a kazoo sound to them.
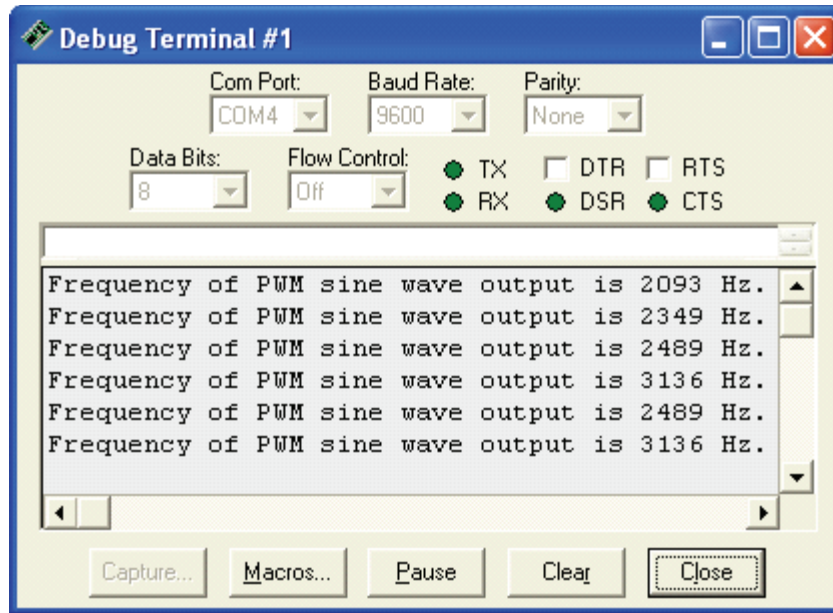
Figure 5-8
Debug Terminal output displaying the frequencies played by Program Listing 5.3.

About the Code

The only new command introduced is `FREQOUT`. The `FREQOUT` command is used as follows:

FREQOUT *Pin, Duration, Freq1, Freq2*

We already know that *Pin* is a number between 1 and 15 that refers to your choice of BASIC Stamp I/O pin, P1 through P15. The *Duration* is a number between 1 and 65535 that specifies how long the tone is played in milliseconds (ms). The term *Freq1* is used to specify the frequency of the tone that gets played. A second frequency *(Freq1)* can be played simultaneously for some interesting effects. For example, the tones you hear when you dial the telephone are actually two tones played simultaneously. The BASIC Stamp also has a command for telephone tones called DTMF, which uses the same principle.

What have I learned?

On the lines below, insert the appropriate words from the list on the left.

voltage	An oscilloscope is a tool used by many electronics technicians and engineers, and it measures and displays _____ signals. Amplitude, DC offset, phase shift, period, and _____ are useful signal characteristics that can be measured with an oscilloscope.
period	
amplitude	
time	An oscilloscope typically displays _____ on the horizontal scale and _____ on the vertical scale. _____ and DC offset are two quantities related to voltage that are measured differently for different signals.
inverse	
amplitude	For a pulse train, the amplitude is the voltage between the high and low signal. For a _____ triangle wave, as with the sine wave, the amplitude is the maximum deviation from the center line of the signal. For the sine wave and triangle wave, the _____ is the voltage between the center line and 0-volts.
RC circuit	
frequency	
symmetric	The frequency is the number of times a waveform repeats itself and the _____ is the amount of time the waveform takes to repeat itself. The frequency is the _____ of the period.
sine	
time-varying	Time varying voltage signals sent to a speaker can result in audible tones. Adjusting the _____ of a voltage signal adjusts the volume of the tone. Adjusting the frequency of the signal adjusts the _____ .
pitch	
DC offset	Musical notes are represented by _____ waves oscillating at certain frequencies. The BASIC Stamp uses pulse width modulation (PWM) sent to the input of an _____ to generate a sine wave at the output.

Questions

1. What's the main difference between the Stamp-O-Scope display and a regular oscilloscope display?

2. If the frequency of a signal is 1000 Hz, what's the period?

3. How can you adjust the scale to display finer variations in voltage using the Stamp-O-Scope? Hint: The answer involves an adjustment to the PBASIC code for Program Listing 5.1.

4. How can you increase the refresh rate of the Stamp-O-Scope? Hint: examine the code again; what factor limits the speed at which the signal is measured?

Challenge!

1. Modify Program Listing 5.3 so that it plays two notes simultaneously. For a hint, look up the `FREQOUT` command in the *BASIC Stamp Manual* or the BASIC Stamp Editor Help menu.

2. Modify Program Listing 5.1 and add a second lead that measures only 0 and 1 logic levels. Modify the code to display voltage activity of this second lead and the Stamp-O-Scope input simultaneously on the same display. Good luck!

3. If you succeeded with Challenge #1, add a potentiometer to your board and wire it as a voltage divider. Hook the new, second input lead to the wiper terminal of the pot. Vary the pot above and below threshold voltage and see if you can match the frequency of the triangle wave.

4. A really cool thing to do would be to discover a phenomenon called phase shift. Connect the input lead of the second channel you developed in Challenge 2 to the D/A converter output. Also, connect the channel 1 lead (ADC0831 Vin(+)) to the D/A converter output. Adjust the D/A converter to up count and down count between 4 and 15. Note that there's a delay between the mid point of the triangle wave and the transition of the pulse train. Adjusting the lower limits of the counting variables and notice that the delay shifts. This delay is called phase shift.

Why did I learn it?

The oscilloscope is an essential tool in the electronics industry, and many hobbyists use them as well. This is a good first exposure to using the oscilloscope and to viewing time-varying waveforms. If you find yourself learning how to use an oscilloscope, some of the concepts introduced here will make some of the functions of the buttons and dials easier to learn.

How can I apply this?

Familiarity with the oscilloscope and with time-varying waveforms will make it easier to understand explanations of various phenomena that vary with time. They appear in many chemistry, physics and electronics texts.

Chapter #6: Recording Frequency Data

In the previous chapter we used the BASIC Stamp to control a D/A converter to produce audible tones. In this chapter, we'll use a 555 timer to generate frequencies in the audible sound range, then we'll use the BASIC Stamp to sample and process the frequency data.

We'll start by viewing a pulse train generated by a 555 timer on a simplified version of the Stamp-O-Scope. Two potentiometers will be used to control the frequency and duty cycle of the pulse train. After viewing the pulse train's signal characteristics, the 555 timer's output will be directed to a piezoelectric speaker. The 555 timer's output will also be connected to a BASIC Stamp I/O pin set to input. The BASIC Stamp will be programmed to monitor and record the frequencies generated by the 555 timer.

6

Parts Required

This experiment requires the following parts:

(2) 10 kΩ potentiometers
(1) Piezoelectric speaker
(1) 555 timer
(1) 100 μF electrolytic capacitor
(1) 1 μF electrolytic capacitor
(2) 220 Ω resistor

Not all the parts will get used at once on the same circuit. Certain capacitors and resistors are substituted in the second circuit to increase the 555 timer's output frequency.

The electrolytic capacitor shown in Figure 6-1 has a positive and negative terminal. The wire that comes out of the metal canister closest to the black stripe is the negative terminal. The arrows (>>) on the black stripe point to the negative terminal. The canister also has the value printed on it. 1μF indicates 1 microfarad while 10 μF indicates a 10 microfarad capacitance value. Likewise, 100 μF indicates 100 microfarads, and so on.

> **The electrolytic capacitor has a positive (+) and a negative (-) terminal.** The negative terminal is the lead that comes out of the metal canister closest to the stripe with a negative (–) sign. Always make sure to connect these terminals as shown in the circuit diagrams. Connecting one of these capacitors incorrectly can damage it. In some circuits, connecting this type of capacitor incorrectly and then connecting power can cause it to rupture or even explode.
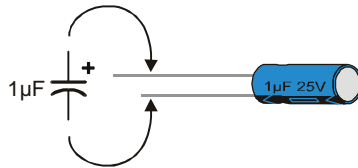

Figure 6-1
Electrolytic Capacitor
Circuit Symbol and
Component

Figure 6-2 shows the symbol and pin map for the 555 timer. As with the ADC0831, the circuit symbol for the 555 timer is drawn for convenience in the circuit diagram. This means that the locations of the pins on the symbol and its size may change from one diagram to the next. The pin map, on the other hand, does not change. Make sure to locate the index mark, and use the pin map in conjunction with the schematic while wiring your circuits.
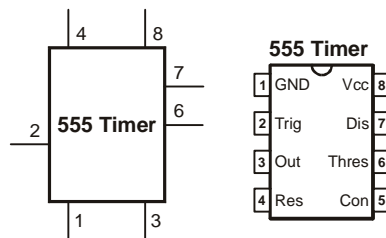
Figure 6-2
Symbol and pin map
for 555 Timer

As always, make sure to locate the index mark when you place the part on the breadboard.

Build It

The circuit in Figure 6-3 is called an "astable multivibrator" and it's used to generate a steady stream of pulses. The timing of these pulses is determined by the capacitor and the resistances of the two potentiometers. You will be able to change the potentiometer resistances by twisting their knobs/dials, which will in turn change the pulse duration and frequency the circuit sends to the speaker. The speaker makes the signal audible, and since the circuit's output is also connect to P0, you will be able to program the BASIC Stamp to measure certain characteristics of the signal as well.

Build the circuit shown in Figure 6-3. Note that there is a lower case "nc" below two potentiometer terminals in the figure. This stands for "not connected". Since the potentiometer is used as a single variable resistor instead of a voltage divider, the third terminal doesn't get connected to anything. It's OK to plug the nc terminal into a row on the breadboard; just don't plug anything else into the same row.
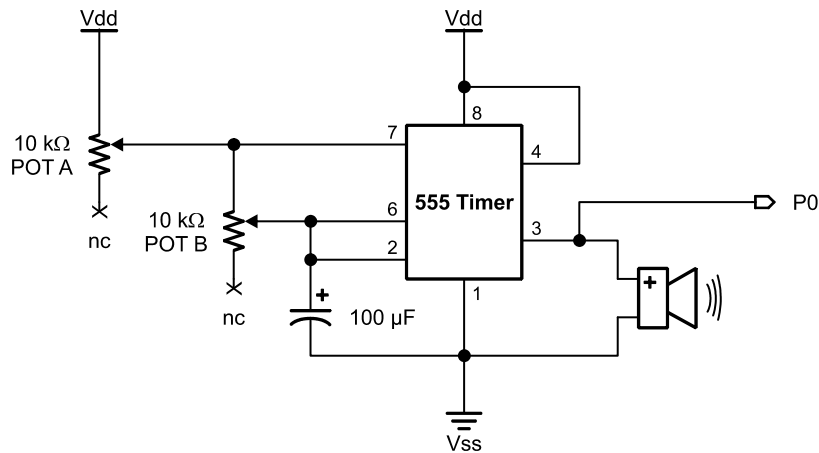

Figure 6-3
Astable
Multivibrator
circuit

With adjustable high and low times.

No program is needed yet at this point. After you've built the circuit, adjust the potentiometers until the piezoelectric speaker makes a steady, slow tic-toc sound. A frequency of 1 Hz is desirable. You can make the tic-toc stop any time you want by connecting pin-5 of the 555 Timer to ground (Vss).

How to predict the 555 Timer's frequency?

$f = 1.45/[C*(R_A + 2R_B)$

More facts acquired from a data sheet, this one on the 555 family of timers. T_H, the pulse width, is given by:

$0.69 \times C \times (R_A + R_B)$,

…where C, R_A and R_B are the values of the capacitor and two resistance values of pot A and pot B as shown in Figure 6.3. T_L, the duration of the low signal (the time between pulses) is given by:

$0.69 \times C \times R_B$

In the previous experiment, the Stamp-O-Scope digitally sampled and displayed signal data. Program Listing 6.1 is a greatly simplified version of the Stamp-O-Scope (the Stamp-O-Scope 2), and it only displays variation in logic level (0 or 1). Since the 555 timer's output is a pulse train that varies between 0 and 5 volts, A/D conversion is not required. The Stamp-O-Scope 2 is all that's needed.

```
' Basic Analog and Digital - PL6_1R0.bs2
' Stamp-O-Scope 2
' {$STAMP BS2}
' {$PBASIC 2.5}

DEBUG CLS                   'Start display

DO
  DEBUG REP " "\IN0*20, "*", CR
  PAUSE 100
LOOP
```

About the Code

The command `DEBUG REP " "\IN0*20, "*", CR` uses the input value measured at pin P0 to dictate how many spaces are printed. No variables are used. When the input is zero, the asterisk gets printed without any spaces before it, indicating a 0 at the input. When the value at P0 is 1, twenty spaces are printed before the asterisk, indicating a measured input value of 1.

The Output

Figure 6-4 shows the Stamp-O-Scope 2 display. You can adjust the potentiometers and change the frequency characteristics of the waveform. For example, pot B directly affects the amount of time the signal is low. It also indirectly affects the amount of time the signal is high. Try adjusting pot B until the low signal is 4 asterisks wide. Then, you can adjust pot A so that the pulse-width varies. Try adjusting A so that the pulse-width is also 5-asterisks wide. Toggle the Pause/Resume button at the bottom of the Debug Terminal by clicking on it with your mouse to freeze and unfreeze the display.
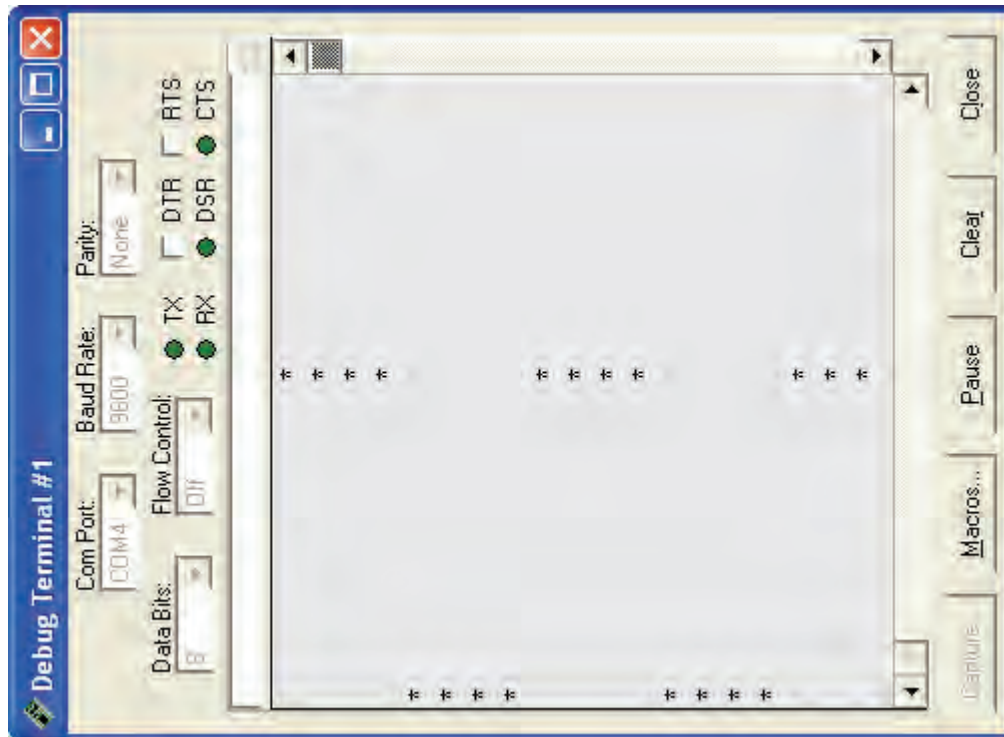
Figure 6-4: Output Sample of 555 Timer Pulse Train

This is a time-varying display of the pulse train generated by the 555 timer as measured by pin P0. Again, the display has been turned sideways to match conventional displays and graphs.

Now, try adjusting pot B and make the low signal two asterisks wide. What happened to the pulse width? It should also have gotten narrower too.

Next, try adjusting pot B as far as it will go to increase the frequency (making the pulse widths narrower). The piezoelectric speaker should be clicking pretty fast now. Carefully adjust pot A and you should be able to hear an audible tone out of the piezo speaker. It's not a pretty tone; it sounds about like a kazoo.

The Stamp-O-Scope 2 is probably not working very well any more, as shown in Figure 6-5. We'll replace it with another program that lends itself to measuring audible sound.
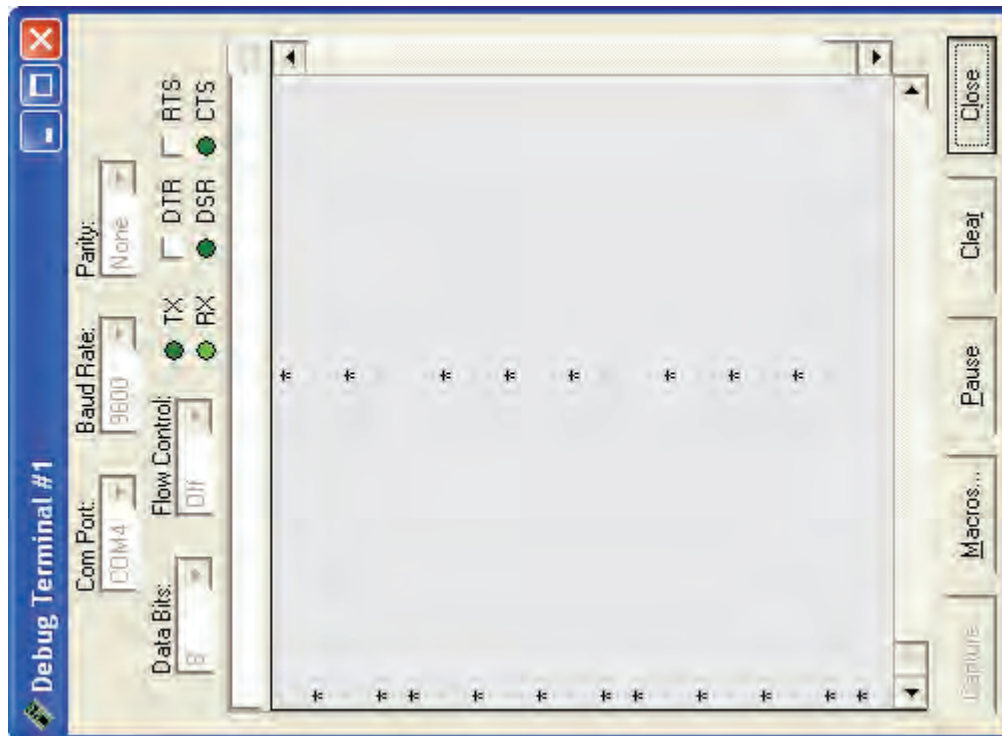
Figure 6-5: Output Sample of Faster Pulse Train

The pulse train is now repeating itself several times as fast as the sampling rate. So the Stamp-O-Scope 2 display is no longer valid.

Now back way off on pot B so that all you get is a periodic clicking out of the piezoelectric speaker. We'll want to crank pot B back up in a little while to get that tone again, but first, let's consider what happened to the Stamp-O-Scope 2 display as the frequency of the 555 Timer was increased.

The display shown in Figure 6-5 still showed signal activity, even though it was erroneous. This is called aliasing. Aliasing happens when you don't sample something fast enough to get a true representation of what's going on. Aliasing can cause problems because sometimes the signal that gets displayed actually looks valid.

For the most part, the Debug Terminal is the limiting factor for sampling rate. So let's make a program that allows the BASIC Stamp take turns between working at full speed and sending messages to the Debug Terminal.

> **When is aliasing a concern?** A recommended sampling rate depends on the characteristics of the signal being measured. Some signals are sampled just a few times per cycle while others are sampled thousands of times per cycle.
>
> The absolute minimum sampling rate, in theory, that can be used to gather valid signal data is twice the frequency being sampled. When the sampling rate is less than twice the frequency of the signal being sampled, aliasing is guaranteed. This minimum frequency is called the Nyquist rate.

A BASIC Stamp I/O pin can be monitored to check the number of threshold voltage crossings per time interval. The BASIC Stamp 2 can be programmed to track these changes once every two microseconds, which makes the period between samples 2 microseconds. From Chapter #5, we know that frequency is the inverse of the period:

```
f = 1 / T
```

We can use this equation with a period of T = 2 μs to calculate the sampling frequency, which is also commonly referred to as the "sampling rate":

```
1 ÷ 2 x 10⁻⁶ seconds = 250 x 10³ = 500 kHz
```

Since the sampling rate is 500 kHz, the theoretical maximum frequency we could sample is 250 kHz. The audible sound range is between 20 Hz and 20 kHz, and the sounds we will work with in this experiment will vary between 50 Hz and 3.5 kHz. So the BASIC Stamp will instead be "oversampling" the signal, and aliasing will not be a problem.

Program It

Let's make a program that counts how many times the pulse train repeats itself. In effect, we will make a program that determines the frequency of the pulse train. The BASIC Stamp has a built in feature for counting frequency, and the PBASIC command is the COUNT command.

With the COUNT command, the BASIC Stamp increments a value when the input voltage passes the 1.4 volt I/O pin threshold twice. This makes programming and interpreting the frequency data easy for periodic signals such as the pulse train, triangle wave and sine wave. Figure 6-6 shows why there are two threshold crossings per repetition of the waveform. When a waveform repeats itself, it's called a cycle. Notice how the waveform crosses the threshold voltage twice per cycle.
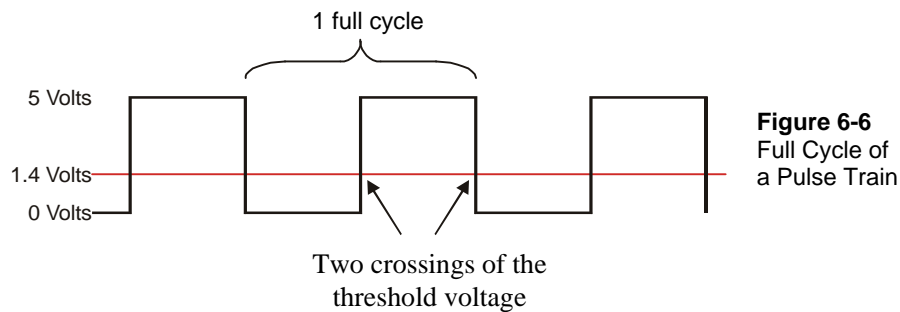
Figure 6-6
Full Cycle of
a Pulse Train

Enter Program Listing 6.2 into the BASIC Stamp Editor and save it as P6_2R0.bs2.

```
' Basic Analog and Digital - PL6_2R0.bs2
' Frequency Meter
' {$STAMP BS2}
' {$PBASIC 2.5}

f       VAR     Word

DO
  COUNT 0, 1000, f
  DEBUG HOME, "Frequency:  ", DEC4 f, " Hz.", CR, CR
LOOP
```

When you run the program, and adjust pot B to its maximum, the output should be somewhere in the neighborhood of the frequency shown in Figure 6-7.
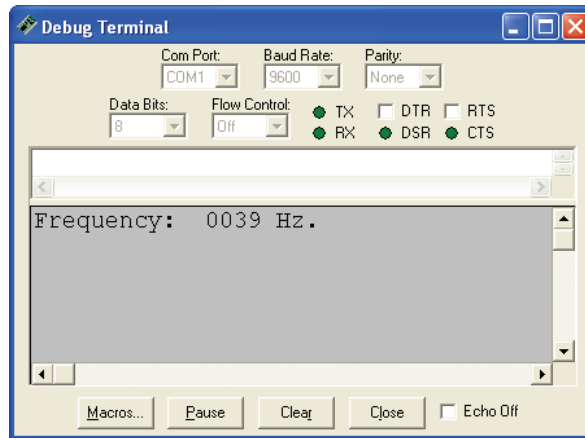
Figure 6-7
Debug Output for
Program Listing 6.2

Try replacing the 100 µF capacitor with a 10 µF capacitor and 1 µF, then adjust the pots to note the frequency ranges and sounds that are possible. Remember that you can turn the sound (noise, racket, whatever you want to call it by now) off by shorting pin 5 on the 555 timer to Vss. When you want the sound back on, just disconnect the wire from the Vss terminal again.

Modify the test circuit by replacing pot A with two 220 Ω resistors in series, and using the 1 µF capacitor. Figure 6-8 shows how the circuit should look when the changes to the circuit have been made.
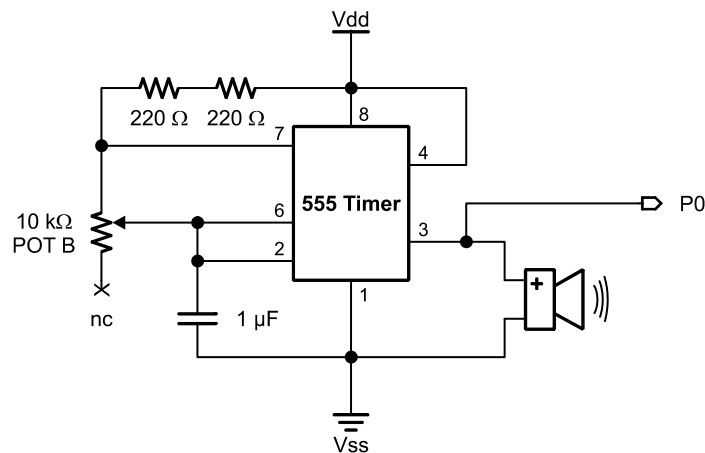
Figure 6-8
Astable Multivibrator
Circuit

*With R_A fixed at 440 Ω.
R_B is can still be
adjusted with pot B.*

Adjusting pot B, you should now be able to measure sound on a frequency range from about 60 Hz to about 3.5 kHz. You can log frequency data as you adjust the pot with the revised version of Program Listing 6.2 below.

```
' Basic Analog and Digital - PL6_2R1.bs2
' Frequency Meter
' {$STAMP BS2}
' {$PBASIC 2.5}

f        VAR     Word(10)
n        VAR     Nib

DO
  DEBUG CLS

  FOR n = 0 TO 9
    COUNT 0, 1000, f(n)
    DEBUG HOME, "Frequency:  ", DEC4 f, " Hz.", CR, CR
  NEXT

  PAUSE 1000

  FOR n = 0 TO 9
    DEBUG "Frequency at  ", DEC1 n
    DEBUG "  seconds was ", DEC4 f(n), " Hz.", CR
  NEXT

  PAUSE 5000
LOOP
```

During the time this program records the frequency, the Debug Terminal display is similar to the previous revision of the program. When the program is done recording, it displays ten frequency samples. The Debug Terminal data shown in Figure 6-9 was generated by adjusting the pot back and forth to some random frequencies (tones).
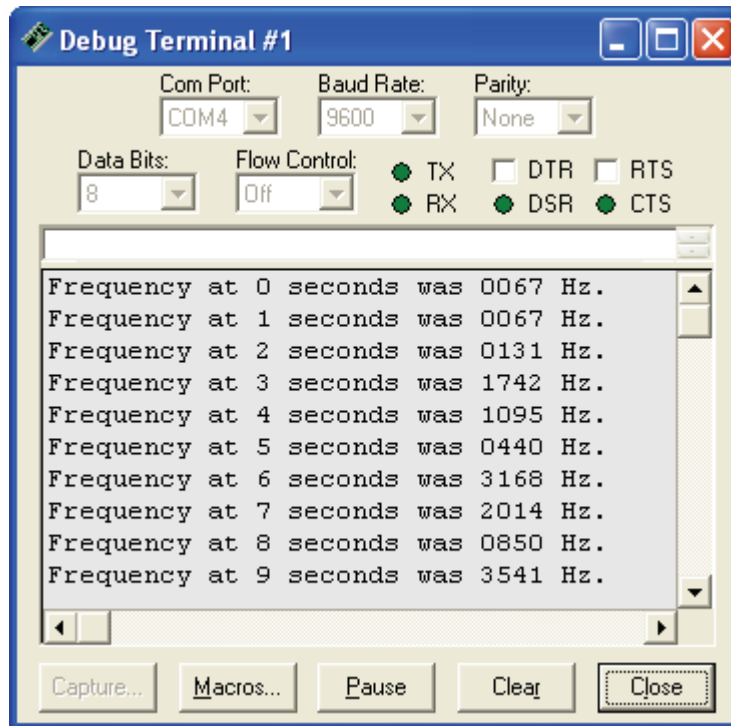
Figure 6-9
Debug Output for
Program Listing 6.2,
Revision 1.

About the Code

Program Listing 6.2 Revision 2 introduces the concept of data storage using an array.
We used the command:

```
f       VAR     Word(10)
```

to reserve 10 words of space in the BASIC Stamp module's RAM. There's a word labeled
f(0), another word labeled f(1), and so on through f(9). Each of these words is stored
adjacent to each other in RAM.

As the value of **n** is incremented in the **FOR…NEXT** loop, so is the index of the array. The
first time through the array, a value is loaded into word f(0), the second time through the
array, a value is loaded into word f(1), and so on through word f(9).

```
FOR n = 0 TO 9
  COUNT 0, 1000, f(n)
```

```
      DEBUG HOME, "Frequency:  ", DEC4 f, " Hz.", CR, CR
NEXT
```

The same concept worked for printing the values back to the Debug Terminal.

```
FOR n = 0 TO 9
  DEBUG "Frequency at  ", DEC1 n
  DEBUG "  seconds was ", DEC4 f(n), " Hz.", CR
NEXT
```

The time-varying waveforms we've looked at in this chapter and in Chapter #5 have been periodic. That is, they've repeated themselves. In the next two experiments, we'll take a look at waveforms that vary with time, but they aren't necessarily periodic.

What have I learned?

On the lines below, insert the appropriate words from the list on the left.

sampling rate
pulse train
time-varying
amount
array variable
incremented
period
duty cycle

Sounds can be characterized by _____ waveforms. The frequency range for audible sound is between 20 and 20,000 Hz. When a time-varying waveform such as a pulse train of sufficient frequency and amplitude is sent to a speaker, the speaker emits audible sound.

The 555 timer in an astable multivibrator circuit can be adjusted to generate a _____. The frequency, pulse width, and _____ can be adjusted by changing the values of the passive elements in the circuit, i.e. the capacitor, and the two resistors.

The pulse width is the _____ of time a signal remains high, and the duty cycle is the ratio of the pulse width to the _____ of the waveform.

Aliasing is a phenomenon that occurs when the _____ is less than twice the frequency of the signal being sampled. This can cause significant misinterpretations of the signal data.

Data can be conveniently stored and accessed in RAM using an _____. The index of an array variable can be _____ using a **FOR…NEXT** loop. When you specify an array of 10 bytes, they are indexed as bytes 0 through 9.

6

Questions

1. Given four different frequencies of sound, 3.5 Hz, 350 Hz, 3,500 Hz, 35,000 Hz, which ones can you hear and which ones can't you hear? Explain your answers. Also, compare the pitch of the sounds you can hear.

2. Explain aliasing. If a signal is sampled at 1 kHz, how low can the signal's frequency get before aliasing is guaranteed?

3. For the circuit in Figure 6-7, what resistance value does the pot have to be to make the astable multivibrator circuit generate a 2 kHz signal? This requires some algebra.

4. Construct a PBASIC command to declare an array of 5 nibbles. Make a command that sets the fist nibble in the array equal to the fifth nibble in the array.

Challenge!

1. In Program Listing 6.2 Revision 2, the frequency samples we took were once every 1 second because the count command counted for a second. Revise the program listing to count the frequency samples for ½ of a second. Make sure your output data is correct by comparing output of the revised program to output of the original for a known frequency. Fix any bugs

2. Design a program to record and play back tones. Make it so that the recorded tone is played on one speaker, and the playback tone is sent to a second speaker. Add a function to the program that shuts the 555 timer off by sending a low signal to pin-5 on the timer when you want to hear the playback signal. Hint: You can combine program listings from Chapter #5 with a program listing from this experiment, and you'll be almost done. All you'll have to do is program the control signal that disables the 555 timer.

3. Design and build the circuit to implement the design from Challenge 2. You'll need to be creative about finding enough space on the breadboard.

4. Add two pushbuttons to the system developed in Challenge 2 and 3 so that you can control when to record and play back single sound samples. Now you'll need to be really creative about finding enough space on the breadboard. Use 10 kΩ

resistors with the pushbuttons, and design them to send either high or low signals. Hint: Review Chapter #1 for circuit and program techniques for implementing the pushbuttons.

Why did I learn it?

Digital processing of audio signals is still expanding in the audio industry. New, more efficient ways of storing and transmitting sound are being developed so that you can transmit music more quickly over the internet and so that your voice sounds more like your voice to the person on the other end of the phone line.

Digital signal processing is also the crux of the telecommunications industry with cellular phones and normal telephones. When you talk on the phone, your voice signal is converted to 0s and 1s and sent via the telephone network in binary format. Your telephone still sends and receives analog voice signals, but your voice information is A/D and D/A converted at various points in the telephone network.

How can I apply this?

Frequency is an effective way to think about sound, but there is a myriad of other uses for frequency counting. For example, engine speed and car speed are determined by how fast certain moving parts on the car turn. When something rotates, each time it turns full circle it has repeated itself. If a shaft is rotating at 100 turns per second, its frequency of rotation is 100 Hz. Machine vibration can also be measured in terms of frequency, and so can many metabolic signals such as breathing, heart rate, and so on.

Chapter #7: Digital to Analog the Easy Way using PWM

A resistor, capacitor, BASIC Stamp, and a single line of PBASIC code. That's all it takes to build a D/A converter with 8-bit resolution. Sound too good to be true? In a sense it is, because a buffer is necessary for this circuit to maintain a given voltage. Even so, it's easy to build and offers a higher degree of accuracy than its resistive ladder counterpart. Figure 7-1 shows the circuit that's used - it's a simple RC circuit. The input receives a pulse width modulated (PWM) signal, and the output rises or falls to the desired voltage level.
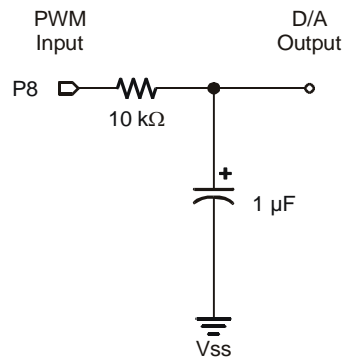
Figure 7-1
RC Circuit

Which can be connected to the BASIC Stamp for D/A conversion.

How can sending pulses of varied width and frequency control or set a voltage level at the D/A output? The answer is the RC circuit shown in Figure 7-1. This circuit behaves like a rechargeable battery. Figure 7-2 shows an example of the simplest kind of capacitor, the parallel plate capacitor. The charges migrate from the battery terminals and accumulate on the two metal plates until the voltage across the capacitor is essentially the same as the battery voltage.
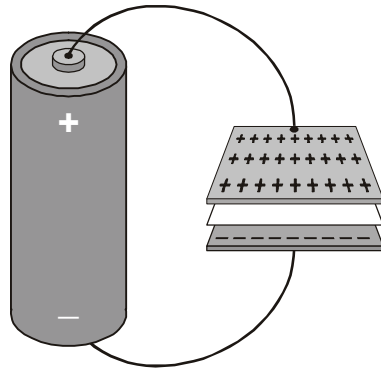
Figure 7-2
Parallel Plate Capacitor Charged by a Battery

The opposed charges congregate at their respective plates. They want to get across the gap between the two metal plates, but there is an insulating material called a dielectric in between the plates, so they can't get across.

You could also disconnect the battery from the capacitor, and it would maintain its voltage. Typically, there is a small amount of current that does make it through the dielectric material that separates the plates. It's called leakage current. Leakage current on a charged capacitor will cause the voltage to slowly dissipate.

The RC circuit shown can charge to within 1% of 5 volts very quickly. Assume that a single pulse is applied to the circuit in Figure 7-1 instead of the many pulses sent via a PWM signal. If the values of the components are exact, it would take about 6.93 milliseconds for the capacitor to charge up to 2.5 volts, and would take 46.1 milliseconds for the output to get all the way to 4.95 volts, which is 99% of the full 5 volts.

For advanced math enthusiasts, let's take a look at the proof. The modeling equation for the RC circuit from Figure 7-1 responding to a pulse that goes from 0 to 5 volts is:

$$V_{Output} = V_{Input} \times \left(1 - e^{\frac{-t}{R \times C}} \right)$$

The modeling equation is rearranged to isolate the exponential term:

$$1 - \frac{V_{Output}}{V_{Input}} = e^{\frac{-t}{R \times C}}$$

The natural log of both sides is taken to remove the exponential term, and the terms are rearranged:

$$\ln\left(1-\frac{V_{Output}}{V_{Input}}\right)=\ln\left(e^{\frac{-t}{R\times C}}\right) \quad\Rightarrow\quad \frac{-t}{R\times C}=\ln\left(1-\frac{V_{Output}}{V_{Input}}\right) \quad\Rightarrow\quad t=-R\times C\times\ln\left(1-\frac{V_{Output}}{V_{Input}}\right)$$

All that's left is to plug in the numbers to solve for $t_{2.5\ volts}$ and $t_{4.95\ volts}$. R and C are the resistance and capacitance values shown in Figure 7-1, and V_{Input} = 5 volts. The first V_{Output} is 2.5 volts.

$$t_{2.5\ Volts}=-R\times C\times\ln\left(1-\frac{V_{Output}}{V_{Input}}\right)=-(10\times10^{3})\times(1\times10^{-6})\times\ln\left(1-\frac{2.5}{5.0}\right)=6.93\times10^{-3}\ \text{seconds}$$

Then, the same calculation can be done for $t_{4.95\ Volts}$.

$$t_{4.95\ Volts}=-R\times C\times\ln\left(1-\frac{V_{Output}}{V_{Input}}\right)=-(10\times10^{3})\times(1\times10^{-6})\times\ln\left(1-\frac{4.95}{5.0}\right)=46.1\times10^{-3}\ \text{seconds}$$

An oscilloscope could be used to monitor the output voltage of the RC circuit from Figure 7-1. The response when a single, very wide pulse (a voltage step) is applied would resemble that shown in Figure 7-3. As you can see, it takes just a fraction of a second for the capacitor to charge up to 5 volts. Nonetheless, it is a gradual change when compared to pulses sent by the BASIC Stamp, which can be as narrow as 2 microseconds
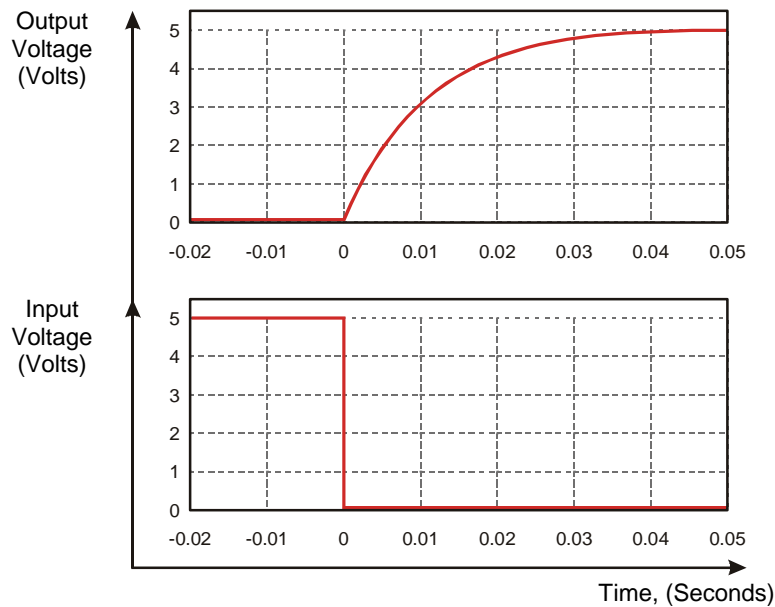
Figure 7-3
RC circuit Step Response

Shows the voltage response to a single voltage step applied at the input.

Applying many narrow pulses as opposed to the single very wide pulse applied in Figure 7-3 has advantages in terms of accuracy. As you will soon find out, this D/A conversion technique is surprisingly accurate - especially compared to the accuracy of the parts used to make the conversion. The resistor has a 10% tolerance and the capacitor is only accurate to 20%!

Parts Required

This experiment requires the following parts:

(1) ADC0831 A/D converter
(1) LM358 op-amp
(2) 10 kΩ resistor
(1) 1.0 μF electrolytic capacitor
(1) Photoresistor
(1) 470 Ω resistor
(1) Yellow LED

The photoresistor senses light level. Under a bright light, the resistance can drop to as low as a few ohms. In darkness, the resistance can climb as high as 50 kΩs. Figure 7-4 shows the circuit symbol for a photoresistor and a drawing of the component
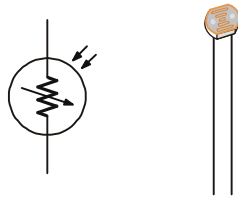

Figure 7-4
Photoresistor Circuit
Symbol and Component

Build It

Build the circuit shown in Figure 7-5. Recall from Chapter #4 that the inputs of the ADC0831 have very high resistance, and they resemble an open circuit. Because of this, it's okay to connect the output of the D/A converter to the ADC0831's Vin(+) input without the buffer. We will use the remaining components later.
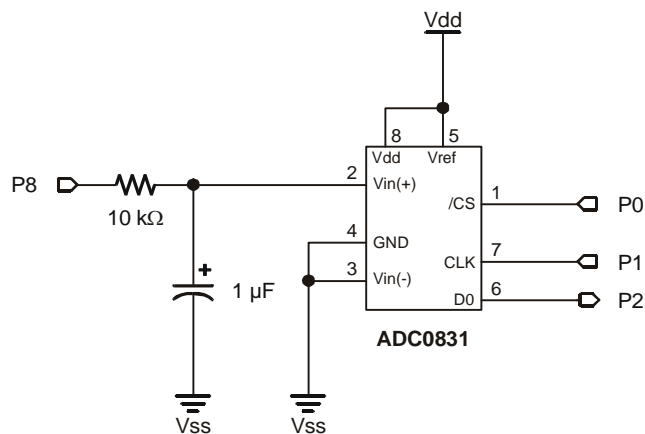

Figure 7-5
RC Circuit for PWM D/A
Conversion

Connected to the Vin(+) terminal of the ADC0831. The ADC0831 is connected to the output of the D/A converter to measure the output voltages.

We are going to use the **PWM** command to send pulses to the RC circuit for D/A conversion. Then we'll measure the voltage using the ADC0831 as a DVM.

The format of the **PWM** command is:

PWM *Pin, Duty, Cycles*

The term *Pin* refers to a BASIC Stamp I/O pin, and you can specify a pin number between 0 and 15 to select a pin. Since the converter input is connected to BASIC Stamp pin P8 in the schematic, we need to set the pin value to 8.

Now, what about *Duty* ? Remember from the Chapter #5 that duty cycle is the ratio of the pulse width to signal period. The term duty is different in the sense that it isn't the duty of a cycle. It's the duty of all the pulses over a certain period of time. In other words, the term *Duty* specifies the ratio of high signal to the time duration of all the pulses in a specific interval of time.

When *Duty* is 255, that means that all the pulses are high, and there is no low signal between them. When all the pulses are high, it means that the capacitor will charge all the way to 5 volts, the high value of the pulses. In theory, the capacitor can never actually charge to 5 volts, but in practice, a time of 5×R×C brings the voltage to within 99.3% of 5 volts.

Therefore, *Duty* is what controls the voltage level, but how do we figure out what we want the duty to be? It's actually pretty simple because it works on a scale from 0 to 255, where 0 is 0 volts and 255 is 5 volts. This looks like a method we already covered in Chapter #3.

Assume we want 3.25 volts. In Chapter #3, we learned how to normalize a scale of 0 to 5 volts to a scale of 0 to 255. Remember that the measured voltage was to 5 as the A/D output was to 255. This translated to fractions as:

$$\frac{\text{Voltage}}{5} = \frac{\text{Decimal A / D input}}{255}$$

In this case, the D/A output voltage is to 5 as the duty is to 255, which translates to fractions as:

$$\frac{\text{D/A Output}}{5} = \frac{\text{duty}}{255}$$

Usually, we know what voltage we want, so we can rearrange the equation to tell us what duty we need:

```
duty = 255 × (D/A Output ÷ 5)
```

Since we want our D/A output to be 3.25 volts:

```
duty = 255 × (3.25 ÷ 5) = 165.75
```

The number 165.75 should be rounded to 166 so that we can set the duty to an integer value.

The last quantity we need to determine for the **PWM** command is *Cycles*, which specifies how many milliseconds the PWM output lasts, and it can be a number up to 65,535. The typical method for choosing how many cycles to charge is:

```
cycles = 4 × R × C, in milliseconds
```

Since we used a 1.0 μF capacitor and a 10 kΩ resistor,

```
cycles = 4 × 0.000001 × 10,000 = 0.04 = 40 × 10-3 = 40 ms
```

For the components we are using, we need to set the *Cycles* value in the **PWM** command to 40.

Now we know that to generate a 3.25 volt output, our **PWM** command needs to be:

```
PWM 8, 166, 40
```

Let's try it.

Program It

Let's start with Program 4.1 from page 71. Modify the declarations section so that the variable n is defined as a word instead of a nibble:

```
' -----[ Declarations ]------------------------------------
adcBits         VAR    Byte
v               VAR    Byte
r               VAR    Byte
v2              VAR    Byte
v3              VAR    Byte
n               VAR    Word           'Δ changed line
```

Replace the **DAC** subroutine as follows:

```
DAC:
  n = 166
  PWM 8, n, 40
RETURN
```

Also replace the **Display** subroutine so that it only displays the voltage, as shown below.

```
Display:
  DEBUG HOME, CR, "DVM Reading:   ", DEC1 v
  DEBUG ".", DEC2 v2, " Volts"
RETURN
```

Also modify the first line of the program for future reference, and save the modified program as P7_1R0.bs2. Then, run the program and check the output.

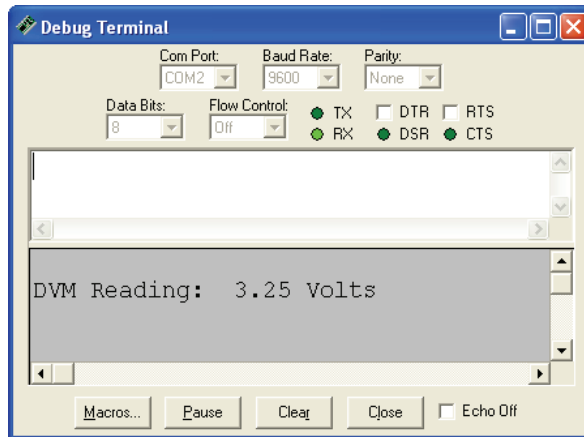### The Output

Take a look at Figure 7-6; not bad!

Figure 7-6
Debug Output for
Program Listing 7.1

Try setting **n** to a few different values in the **DAC** subroutine.

> ⓘ **High input impedance:** BASIC Stamp I/O pins set to input mode, op-amp input terminals, and the ADC0831's Vin terminals all have an important aspect in common that makes them invisible to other circuits. It's high input impedance.
>
> The term impedance includes both resistance and capacitance. High input impedance means that the resistance is huge, and there is a very small capacitance as well.
>
> The impedance at the input terminals of these devices is so high that it fools the most circuits into behaving as if the inputs to these devices are open circuits. So, for the most part, you can connect the output of just about any circuit to these inputs.
>
> On the other hand, we've seen how comparatively low resistance circuits such as a 10kΩ resistor or an LED circuit can drastically change the output of D/A circuits.
>
> One advantage of many D/A converter integrated circuits is that they provide reliable, buffered output

7

Important: This circuit must be isolated from other circuits. The input impedance of the V(+) pin on the ADC0831 is very high, high enough to make it invisible to the RC circuit. But if you tried to use this circuit to drive an LED-resistor circuit without the buffer, all the charge that built up in the capacitor would exit through the LED and resistor.

Try connecting a 10k Ω resistor from the D/A converter output to Vss. Figure 7-7 shows what happens when you try to send 3.25 volts with a load on the D/A converter.

Since the D/A **PWM** repeats itself without a pause in the program, the capacitor does not have enough time to discharge all the way to 0 volts.

If the program were set to carry out the **PWM** command only once, the capacitor would discharge very quickly
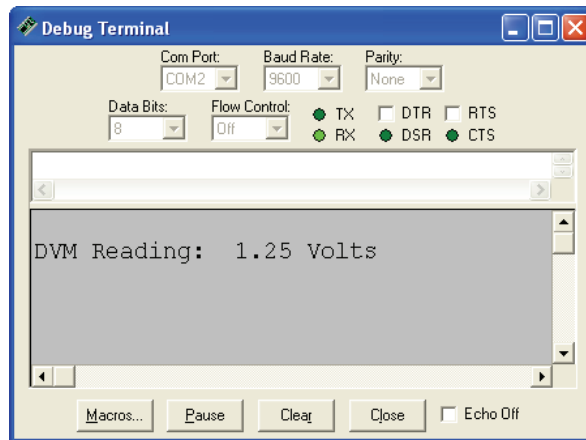
Figure 7-7
Debug Terminal
Output for Program
Listing 7.1 with an
Unbuffered 10 kΩ
load.

As you can see, a buffer is essential to keep the voltage steady. You can use the same voltage follower that we used in Chapters #2, #4 and #5 to buffer the output.

Place a voltage follower in between the D/A output and an LED circuit input as shown in Figure 7-8, then run Program Listing 7.1 again.
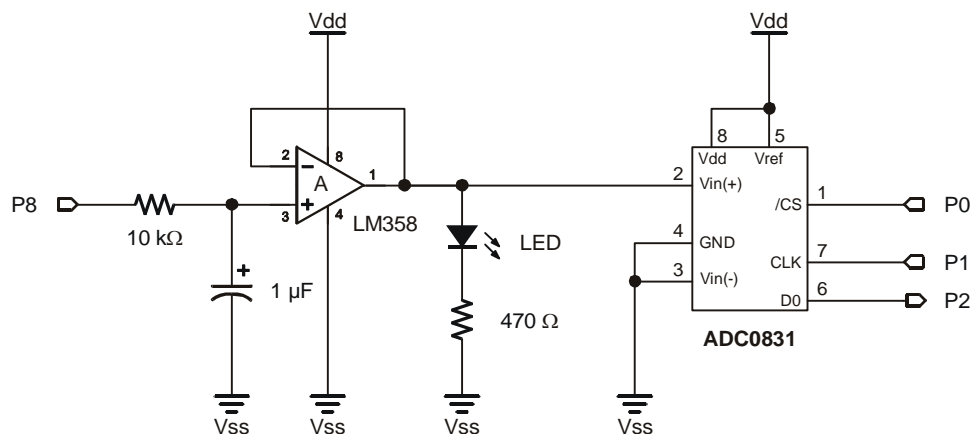

Figure 7-8: PWM DAC with Op-amp Buffer and a Load

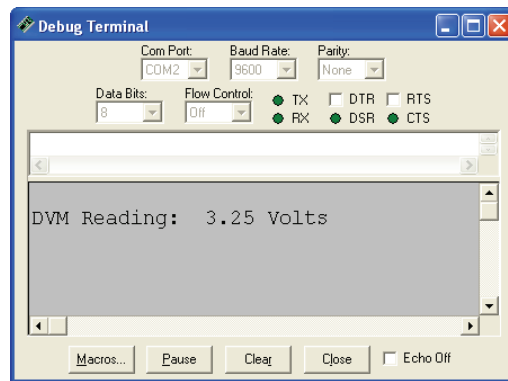Figure 7-9 indicates that the buffer is doing its job.

Figure 7-9
Debug Terminal
Output for Program
Listing 7.1 With a
Buffered LED
Circuit for a Load.

Next, add a photoresistor in series with a 10 kΩ resistor as shown in Figure 7-10. The positive lead of your BASIC Stamp DVM should be connected to the node where the photoresistor connects to the 10 kΩ resistor. This is a voltage divider output, and as the resistance of the photoresistor varies, so will the voltage seen at the output of the voltage divider. Also, point the top of the yellow LED directly at the face of the photoresistor as shown.
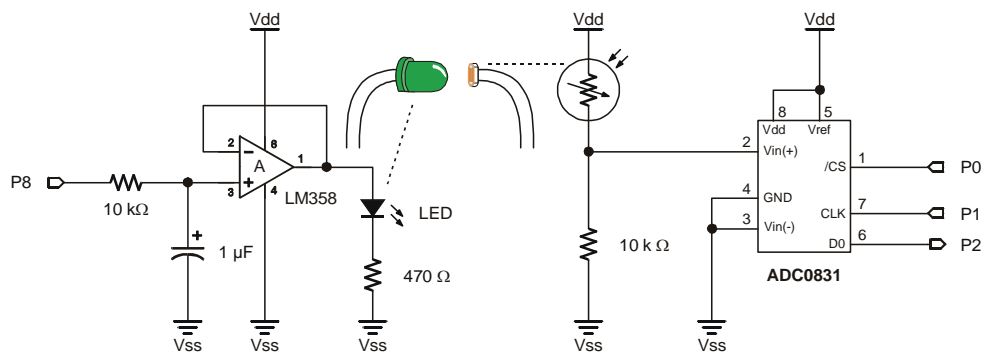

Figure 7-10: Transmitting an Optical Signal.

PWM sets the RC circuit's output voltage, which in turn drives an LED circuit via a voltage follower.

The light from the LED is transmitted to the photoresistor. Face the two devices at each other as shown. The voltmeter from Chapter #3 senses the changes in voltage divider output.

Let's try adjusting the brightness of the LED and see what happens to output at the voltage divider. Make the changes to the Declarations and Main Routine sections, and to the **Display** and **DAC** subroutines shown below. The **n** variable is defined as a word. Then save the program as P7_1R1.bs2, and run it.

```
' -----[ Title ]------------------------------------------------------
' Basic Analog and Digital - PL7_1R1.bs2
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-----------------------------------------------
adcBits         VAR     Byte
v               VAR     Byte
r               VAR     Byte
v2              VAR     Byte
v3              VAR     Byte
n               VAR     Word

' -----[ Initialization ]---------------------------------------------
CS              PIN     0
CLK             PIN     1
DataOutput      PIN     2

DEBUG CLS                                       'Start display.

' -----[ Main Routine ]-----------------------------------------------
DO
  FOR n = 82 TO 199 STEP 9
    GOSUB DAC

    GOSUB ADC_Data
    GOSUB Calc_Volts
    GOSUB Display
    PAUSE 4000
  NEXT
LOOP

' -----[ Subroutines ]------------------------------------------------
DAC:
  PWM 8, n, 40
RETURN

ADC_Data:
  LOW CLK
```

```
  LOW CS
  PULSOUT CLK, 210
  SHIFTIN DataOutput,CLK,MSBPOST,[adcBits\8]
  HIGH CS
RETURN

Calc_Volts:
  v = 5 * adcBits / 255
  r = 5 * adcBits // 255
  v2 = 100 * R / 255
  v3 = 100 * R // 255
  v3 = 10 * v3 / 255
  IF (v3 >= 5) THEN v2 = v2 + 1
  IF (v2 >= 100) THEN
    v = v + 1
    v2 = 0
  ENDIF
RETURN


Display:
  DEBUG HOME, "Decimal value to DAC:  ", DEC3 n
  DEBUG CR, CR, "DVM Reading:  ", DEC1 v
  DEBUG ".", DEC2 v2, "-Volts", CR, CR
RETURN
```

7

The Debug Terminal output should indicate that the output of the voltage divider varies as the LED gets brighter and dimmer. To get the highest swings in measured voltage, turn the lights off in the room. The circuit should work fine under normal lighting conditions as long as no direct light from windows or overhead lamps is hitting the front of the photoresistor. Use a book or a clipboard between these light sources and the photoresistor to cast a shadow. If your circuit is in direct sunlight, the differences in output voltage may be too small to measure.

The voltage divider equation can be used to determine the resistance of the photoresistor. From Chapter #3, the voltage divider equation is:

$$V_{Output} = V_{Input} \times \frac{R_2}{R_1 + R_2}$$

R_2 is a known resistor value, and R1 is the unknown resistance value of the photoresistor. V_{input} is 5 volts, and V_{output} should be noted from the Debug Terminal. It takes some algebra to rearrange the terms; the final result is:

$$R_1 = \left(R_2 \times \frac{V_{Input}}{V_{Output}} \right) - R_2$$

This is one method of using a variable resistor as a sensor. Clearly it can detect different levels of light, and the ADC0831 handles converting the analog information into digital information. In the next chapter, we'll introduce another method for measuring variable resistance that uses significantly less hardware and code.

What have I learned?

On the lines below, insert the appropriate words from the list on the left.

entire analog RC buffer voltage divider high capacitor	The BASIC Stamp has a built in feature for D/A conversion that uses pulse width modulation. The technique uses a series of pulses to charge the capacitor in an _____ circuit to the desired voltage. PWM relies on the duty, which is different from duty cycle, to control the voltage the capacitor is charged to. Duty refers to a series of pulses instead of a single cycle. It's the total amount of time the signal is _____ divided by the total amount of time for the _____ pulse train. A _____ is essential for this circuit when driving a load. Without it, the _____ will quickly loose its charge. A capacitor also looses its charge due to leakage current, so the PWM should be repeated periodically to refresh the voltage. A photoresistor is a device whose resistance value changes with the brightness of light that shines on a collecting surface. The photoresistor or any resistor whose value changes with some analog value can be used in a _____ and the result is a variable voltage output. This information can be interpreted using the ADC0831 and the BASIC Stamp to measure the _____ voltage output.

7

Questions

1. Can changing the capacitor or resistor values improve the resolution of the PWM D/A converter? Explain your answer.

2. Given the command: `PWM 14,51,50`, approximately how many milliseconds does the command take to execute? How many milliseconds is the pulse train high?

3. Why does the capacitor loose its charge if you attach a resistive element to the output of the `PWM` D/A converter?

4. If you want a 2.5 volt output, what do you have to set the duty to? Assume you are using the `PWM` command in Program Listing 7.1. Explain your reasoning.

5. If you want to do DAC using a 10 µF capacitor and a 47 k-Ω resistor, what should your `cycles` be?

6. If the measured voltage in a voltage divider with a photoresistor as shown in Figure 7-10 is 2.3 volts, what is the value of the photoresistor?

Challenge!

1. Write a program that controls the voltage on 2 separate D/A converters that maintain different voltage values.

2. Design and build the circuit for the program in Challenge 1. You will need to make use of the second op-amp in the LM358. Refer to Figure 1-5 for the schematic.

3. Program your dual output DAC to make the LED load on channel 1 get gradually brighter as the LED on channel 2 gets gradually dimmer.

4. Write a program that tells you the measured resistance value of the photoresistor from Figure 7-8. Hint: Apply the voltage divider equation and use the existing DVM program. You will need to modify the `Calc_Volts` and `Display` subroutines.

Why did I learn it?

Using a simple RC circuit with a buffer is an inexpensive and easy way to get 8-bit DAC. It's also really easy to program, and the accuracy is much closer to that of a D/A converter integrated circuit than it is to the accuracy of the resistive ladder network with 10% resistors.

There are many instances in which analog quantities will change the resistance of a device. The voltage divider can be an effective way to track these changes in resistance. The measured voltage at the output of a voltage divider can give us information, not only about the resistance value, but about the analog quantity that causes the resistance value to change.

How can I apply this?

The optics industry has many applications where the levels of light are controlled and sensed. In the communications industry, fiber optic cables are used to send your phone conversation across long distances. In the robotics industry, sensing light is an important aspect of controlling the movement of the mechanical parts of the robot.

The application you're probably most familiar with is the remote control on your television set. When you point the remote control at your television and press a button to change the channel, that information is sent using light, which is received by a sensor on the front of your TV.

Many industrial processes need to use feedback to sense the level of light and adjust it to keep it in a particular range. In photography, many cameras depend on analog measurements taken by a light meter. These measurements can in turn be used to control the aperture of the shutter when taking a photograph. The aperture is how far the shutter opens to let light into the camera. In the next chapter, we'll build a light meter.

Chapter #8: Light Meter Gizmo with R/C Time Constants

In this chapter, we'll be using concepts we learned in previous activities to develop a light meter gizmo. Here is the list of gizmo specifications:

- The gizmo should beep every half-second.
- When the light level in the room gets brighter, the gizmo should beep at a higher pitch.
- When the light level in the room gets dimmer, the gizmo should beep at a lower pitch.

This experiment introduces a way of using a BASIC Stamp I/O pin to measure variable resistance or capacitance. A number of sensors aside from the photoresistor vary in either resistance or capacitance. A simple and straightforward way of measuring these quantities can make interfacing with these sensors immensely easier.

In this experiment, we'll begin by building our gizmo, next we'll look more closely at the measurement technique to see how it can be used to monitor variable resistance or capacitance. Figure 8-1 shows a generic RC circuit that can be used to take RC time constant measurements.
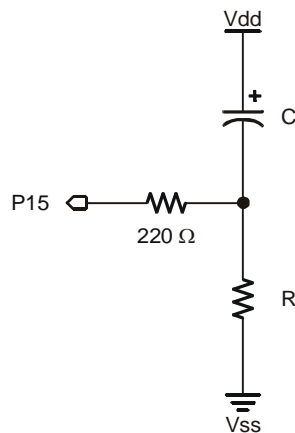

Figure 8-1
RC Circuit

For measuring values of R or C

In Chapter #7, the amount of time it takes a capacitor to charge in an RC circuit with a step input was introduced. The same principle applies here. A step will be applied to the circuit, and the amount of time it takes the capacitor to charge will be measured. Here's how it works:

The BASIC Stamp must be programmed to set P15 (the I/O pin we'll be using) high for a few milliseconds to let the voltage across the capacitor approach 0 volts. The reason the voltage across the capacitor approaches 0 volts is because Vdd is 5 volts at the positive terminal of the capacitor, and P15 is set to 5 volts at negative terminal. So, the voltage across the capacitor is 5-5 volts = 0 volts.

As soon as the voltage across the capacitor is close enough to 0 volts, P15 is set to function as an input. Immediately, the BASIC Stamp starts counting in 2 microsecond increments. As soon as P15 becomes an input, it doesn't have an output voltage of 5 volts any more. P15 behaves like an open circuit as far as the RC circuit is concerned.

All of the sudden, the circuit is different. It's a resistor and a capacitor in series. The capacitor starts with no charge, but the circuit has 5 volts across it (Vdd – Vss), so the capacitor starts to charge up. As the capacitor starts charging, the voltage at P15 starts falling. The voltage at P15 will fall until it crosses the BASIC Stamp I/O pin threshold voltage. It takes a certain amount of time for the voltage at P15 to decay from 5 volts, to the 1.4 volt threshold voltage of P15.

The BASIC Stamp can be used to measure the amount of time it takes the output to drop from 5 to 1.4 volts, and this measurement can be used to determine an unknown capacitance or resistance.

The equation for the voltage at P15 is given by:

$$V_{P15} = V_{dd} \times e^{\frac{-t}{R \times C}}$$

This equation is widely used in many fields, and it's called the exponential decay equation. In our case, it's an equation of exponentially decaying voltage. Figure 8-2 shows the decay curves of three RC circuits that differ only in resistance. Each of the voltage outputs takes a different amount of time to decay from 5 volts to 1.4 volts.
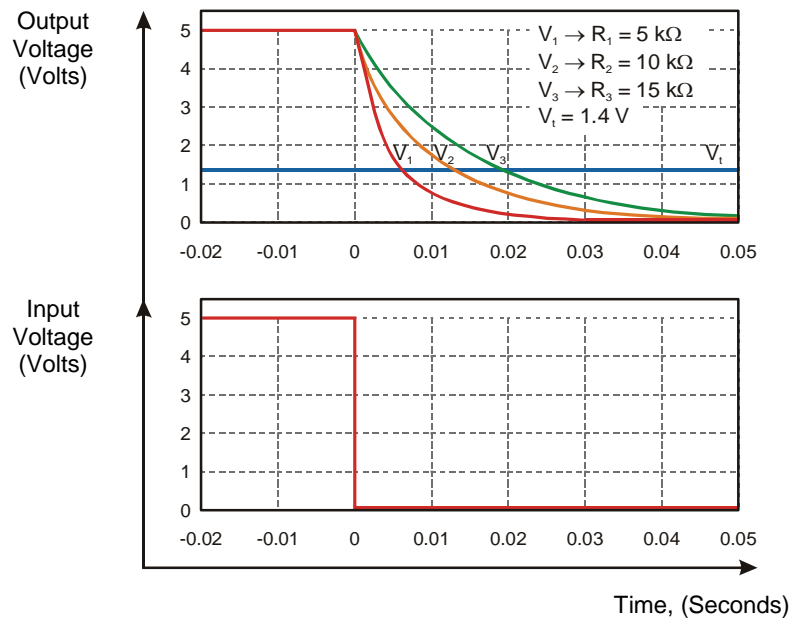
Figure 8-2
RC circuit Responds…

…to a negative step input for three different resistance values. Note how each output takes a different amount of time to decay from 5 volts to 1.4 volts.

The difference in time it takes the three different circuit outputs to decay can be used to determine a resistance value if the capacitance value stays the same. Likewise, if the resistance is fixed but the capacitance varies, the capacitance can be determined by measuring the time between the step input and the threshold voltage.

A PBASIC command called **RCTIME** can be used to make the BASIC Stamp determine the amount of time it took the RC output to decay from 5 volts to 1.4 volts, which is the value of t in the exponential decay equation. The value of t that gets measured is $t_{1.4\ Volts}$. The value of $t_{1.4\ Volts}$ is the amount of time it takes for the voltage at I/O pin P15 to fall from 5 volts to 1.4 volts.

It turns out that the amount of time it takes for the voltage to decay is directly proportional to R multiplied by C. R×C is called the RC time constant, and it is often represented by the Greek letter tau "τ."

At the instant the voltage at P15 reaches 1.4 volts, the input value at pin P15 goes from 1 to 0, and the BASIC Stamp stops counting 2 μs intervals.

Parts Required

This experiment requires the following parts:

(1) 0.1 µF poly capacitor
(1) 1 µF electrolytic capacitors
(1) 10 µF electrolytic capacitor
(1) Piezoelectric speaker
(1) 220 Ω resistor
(1) Photoresistor

Build It

Build the circuit shown in Figure 8-3 Note that the resistor in the circuit on the left uses the variable photoresistor. The circuit on the right is the circuit we used to play music in Chapter #5.
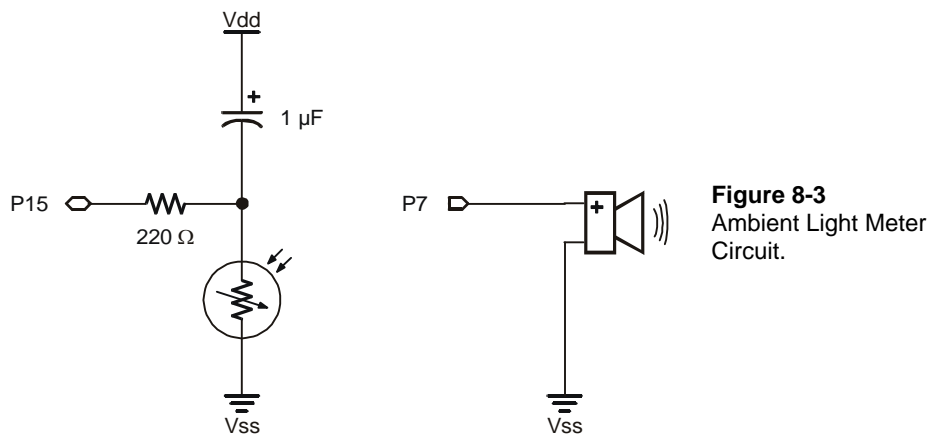

Figure 8-3
Ambient Light Meter Circuit.

You will have to calibrate both your circuit and program due to varying light conditions. Do not point the photoresistor directly at the sun. We'll use the slot in the center of the breadboard as our light collector. Point the photoresistor at the slot as shown in Figure 8-4.
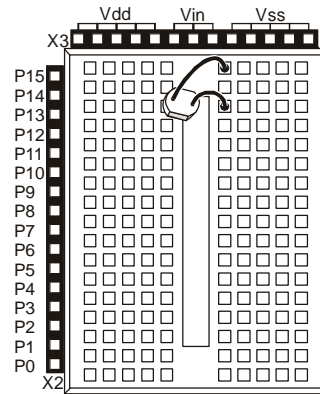
Figure 8-4
Placement of the Photoresistor

Face the photoresistor into the slot in the middle of the breadboard. This way the photoresistor will detect the light reflected off the white breadboard. This will make the photoresistor less subject to wide resistance swings that can happen when the photoresistor is faced directly at bright light sources.

The PBASIC instruction set has a command called **RCTIME**, which measures the amount of time it takes a capacitor to discharge under the conditions just described. The format of the **RCTIME** command is:

RCTIME *Pin, State, Result*

As usual, *Pin* refers to a BASIC Stamp I/O pin. Referring to Figure 8-1, we are using pin P15. Now, what about *State*? We are measuring RC decay time, waiting for the logic to transition from 1 to 0, so we will set it to 0. If we were measuring RC growth instead, the *State* would be set to 0. The *Result* is a number of 2 microsecond time increments that the BASIC Stamp counted during discharge. How does this work?

First I/O pin 15 must be set high, to charge the capacitor to 5 volts. When the BASIC Stamp gets to the **RCTIME** command, it switches I/O pin 15 to input. The voltage starts to drop as shown in Figure 8-2. The BASIC Stamp counts the number of 2 microsecond increments between switching to input and the time the voltage at P15 crosses the 1.4 volt logic threshold. The number of 2 microsecond increments then gets saved as the result variable.

We'll use this segment of code in Program Listing 8.1 to measure the RC time. We will first wait 100 ms with pin P15 high. This gives the capacitor enough time to charge up.

```
HIGH 15
PAUSE 100
```

Then **RCTIME** takes care of the time measurement and data storage.

```
RCTIME 15, 1, light
```

Program It

Enter Program 8.1 in your BASIC Stamp Editor, and download to the BASIC Stamp.

```
' Basic Analog and Digital - PL8_1R0.bs2
' Program Listing 8.1 Revision 0.
' {$STAMP BS2}
' {$PBASIC 2.5}

light           VAR     Word
dischargeTime   VAR     Word
sound           VAR     Word

a               CON     40

DO
  HIGH 15
  PAUSE 100
  RCTIME 15, 1, light

  dischargeTime = light * 2
  sound = (65535 - light) / a

  FREQOUT 7, 500, sound

  DEBUG HOME
  DEBUG "Discharge time = ", DEC5 dischargeTime, " u-seconds."
  DEBUG CR, CR, "Sound Frequency = ", DEC5 sound, " Hz."
  PAUSE 500
LOOP
```

The Output

The piezoelectric speaker should beep every half second or so. If you put your hand over the circuit to shade it, the tone of the beep should get lower. If you face your breadboard directly at a light in the room, the pitch should get higher. Figure 8-5 is a sample output for a well-lit room.
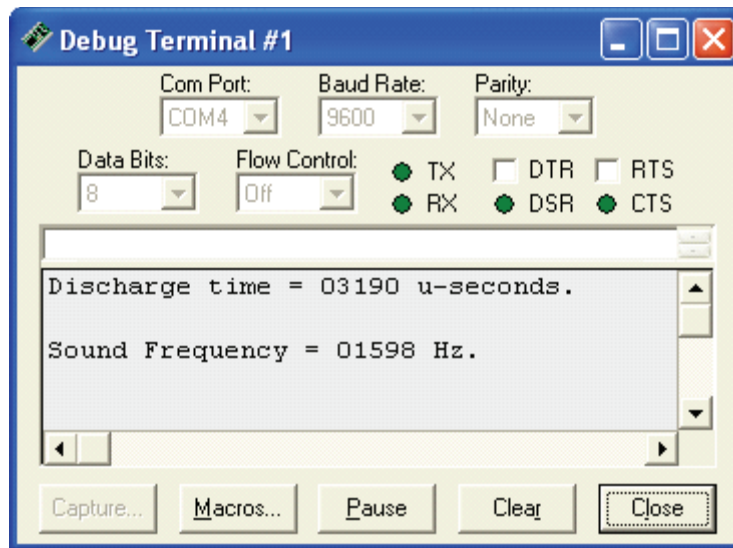
Figure 8-5
Sample Debug
Terminal Output in
a Well-lit Area with
no Direct Sun

8

Figure 8-6 shows a sample output with a book casting a shadow over the light meter. The tone emitted by the speaker is audibly lower
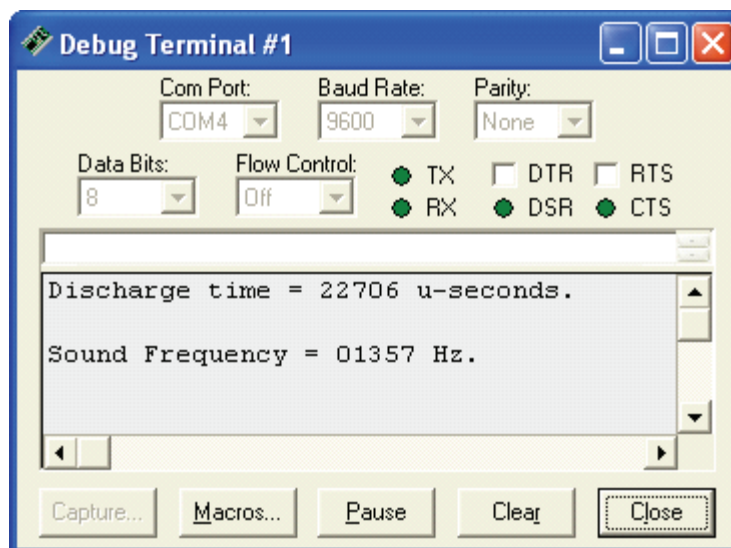

Figure 8-6
Sample Debug
Terminal Output in
Shade

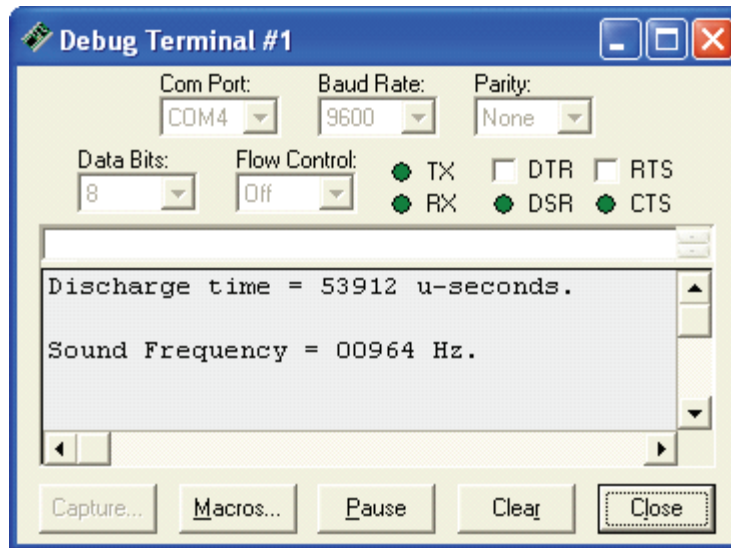Figure 8-7 shows the output under low light conditions.

Figure 8-7
Sample Debug
Terminal
Output for Low
Light

Figure 8-8 shows a sample of the output with the lights out. Overflow occurred. The discharge time was more than 65,536 microseconds. As a result, the variable reset to zero because the light variable also overflowed. Since both `dischargeTime` and `light` reset to zero, the variable `sound` reached its maximum. This is because sound was programmed as: `sound = (65535 – light)/40`. Light was reset to zero due to overflow, so sound became (65535 -–0)/40 = 1638.
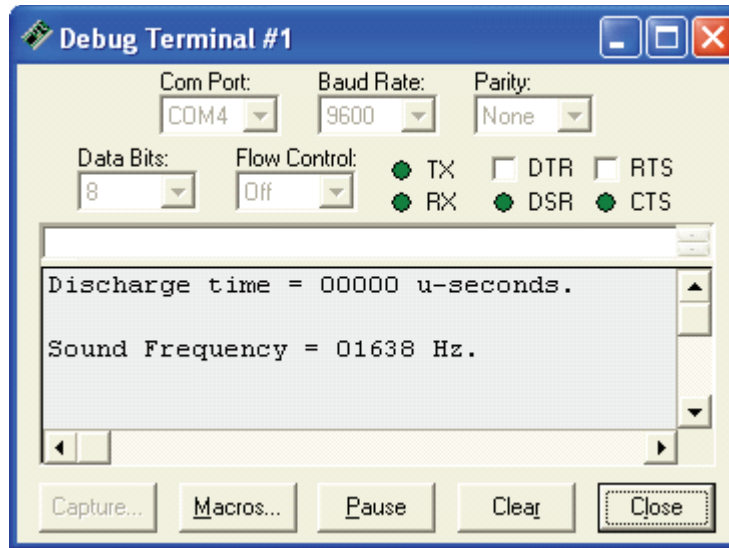
Figure 8-8
Sample Debug
Terminal Output
in After
Somebody
Turned Out the
Lights

Under brighter light conditions, the resolution is poor. You can replace the 1 µF capacitor with a 10 µF capacitor to increase the resolution at higher light levels. For full sun, some kind of shade for the photoresistor would help. If you are dealing with low light conditions, use a 0.1 µF capacitor instead.

The reason for using different capacitors is because the range of resistances for the photoresistor varies for different ranges of light levels. Under lower lighting, the photoresistor has a much higher range of resistance values than it does in a well-lit or full sun area.

As discussed earlier, the discharge time depends on the RC time constant, R x C. If R tends toward a relatively high range of resistances, using a smaller value of C will bring the value of R multiplied by C back down. Likewise, if R is low, using a larger value of C helps bring the value of RC back up.

Do the Math

The RC time measurements along with the exponential decay equation can be a powerful tool for determining an unknown resistance or capacitance.

$$V_{P15} = V_{dd} \times e^{\frac{-t}{R \times C}}$$

Since we'll use PBASIC to make the BASIC Stamp tell us the measured amount of time, t, we have the following information shown in Table 8-1

Table 8-1: Known Values at the Instant VP15 Crosses the Threshold Voltage	
Values	**Comments**
Vdd = 5.0 volts	The initial condition of the voltage at P15.
VP15 = 1.4 volts	The final condition of the voltage at P15.
e ≈ 2.718	The value of e is a constant found in many algebra books, physics texts, etc.
C = Can solve for	If the value of resistance (R) is known, then the capacitance (C) can be determined.
t = Known	The BASIC Stamp counts the time for us in 2 microsecond increments.
R = Can solve for	If the value of capacitance (C) is known, then resistance (R) can be determined.

Algebra can be applied to the exponential decay equation to solve for R or C. First, take the natural log of both sides of the equation. It eliminates the e term.

$$\ln(V_{P15}) = \ln\left(V_{dd} \times e^{\frac{-t}{R \times C}}\right) \quad \Rightarrow \quad \ln(V_{P15}) = \ln(V_{dd}) - \frac{t}{R \times C}$$

Rearranging terms yields:

$$\frac{t}{R \times C} = \ln(V_{dd}) - \ln(V_{P15})$$

Using properties of logarithms, this simplifies to:

$$\frac{t}{R \times C} = \ln\left(\frac{V_{dd}}{V_{P15}}\right)$$

Rearranging again gives us an equality in terms of R and C and t:

$$R \times C = \frac{t}{\ln\left(\dfrac{V_{dd}}{V_{P15}}\right)}$$

If R is the unknown value, divide both sides by C. If C is the unknown value, divide both sides by R. The resulting two equations are useful for determining an unknown R if C is known, or visa versa.

$$R = \frac{t}{C \times \ln\left(\dfrac{V_{dd}}{V_{P15}}\right)} \qquad \text{or} \qquad C = \frac{t}{R \times \ln\left(\dfrac{V_{dd}}{V_{P15}}\right)}$$

Try a few different resistor and capacitor combinations from the Analog and Digital Parts Kit. Remember that capacitor values can vary as much as 20% from their nominal (named) value. This will in turn cause error in your measured values. However, if a resistor with a very low tolerance, such as 1% is available, the measurements of capacitance can be pretty accurate. Once the capacitor has been calibrated, then it's possible to take accurate resistance measurements with the RC circuit as well.

What have I learned?

On the lines below, insert the appropriate words from the list on the left.

threshold capacitance capacitor RC exponential discharge	Analog sensors that vary in resistance or capacitance are used to measure a variety of physical quantities. An _____ circuit can be used in conjunction with a BASIC Stamp binary input pint to measure resistance or capacitance. The _____ is charged to 5-volts, then it is allowed to discharge, and the time it takes to get from 5-volts to the _____ voltage of a BASIC Stamp input pin is measured. The equation for _____ decay can then be used to determine a value of resistance or capacitance if the other quantity is known. An RC circuit where the resistance varies can be calibrated to a certain range of _____ times by changing the value of the capacitor. This method can be used because the resistance multiplied by the _____ is logarithmically related to the rate of decay.

Questions

1. Given the circuit in Figure 8.3, how long will it take to discharge if the value of the photoresistor is 47 kΩ?

2. For the discharge time in Question 1, would you expect any of the variables in Program Listing 8.1 to overflow? If so, which ones? Explain your answer.

3. If the light conditions are so low that none of the capacitors in the kit work any more, a store-bought capacitor might be the answer. What values of capacitor might you try to find? Explain your answer.

4. For better resolution in the higher end of the light level scale, you can also reduce the value of `a` in Program 8.1. How would that affect your Debug Terminal readout and the sound of the piezoelectric speaker?

Challenge!

1. Design a circuit which has two photoresistors that are monitored. Assume that you will be displaying the light intensity information on the Debug Terminal, and that you won't be using a piezoelectric speaker.

2. Program the BASIC Stamp to monitor the light level at both of the photoresistors and display the information using the Debug Terminal.

3. Try holding different colored papers in front of the photoresistors; black and white will work best. Write a program to distinguish which photoresistor senses white and which sees black.

4. Make sure your program from Challenge #3 is not affected by the ambient light level in the room. Hint: You will need to compare the RELATIVE intensities of the light measured by both photoresistors instead of comparing them to some fixed number that was experimentally determined using a fixed value of ambient light.

8

Why did I learn it?

Pretty neat, huh? We used light for our input and sound was our output. We measured light intensity and generated a particular tone based on that measured intensity.

This chapter illustrates how the concepts introduced in this series of experiments can be mixed and matched to actually design an electronic device. It also demonstrates that capacitors are not the exact value they are stated to be due to tolerances. You now know how to calculate the capacitor value.

How can I apply this?

While learning all this, we've covered a number of circuit designs and a wide variety of programming techniques. We've also gained some first hand experience with analog phenomenon. We've also covered how to electronically interface, process, and in general, do things with a digital device, the BASIC Stamp microcontroller. Below is a list of key concepts that we've worked with in these experiments.

Chapter #1: Serial and parallel data, and synchronous and asynchronous communication.
Chapter #2: The comparator, the buffer, and threshold voltage.
Chapter #3: Analog to digital conversion, voltage dividers, and normalized voltage measurements.
Chapter #4: Digital to analog conversion, the resistive ladder network, and sound frequency and volume.
Chapter #5: The oscilloscope and time varying signals.
Chapter #6: The 555 timer, frequency and measuring frequency signals.
Chapter #7: The optical couple and pulse width management for D/A conversion.
Chapter #8: Building an electronic gizmo - a light meter, and A/D conversion using RC time constants.

These eight experiments provide a first exposure to analyzing a variety of analog phenomenon, the basics of electric circuits, and the essentials of digitally processing information about analog measurements using a digital electronic device.

If you are an electronics hobbyist, hopefully there were a few items to add to your bag of electronic design tricks. If you are a high school or college student continuing on to an electrical technical or engineering field, you will undoubtedly revisit many of the concepts covered here in much grater detail.

Many of the parts used in this series of experiment are very basic and inexpensive. Applications kits and a myriad of components are available that can be used to add a new level of functionality to the BASIC Stamp. Some examples of these are digital keypads, liquid crystal displays and servo motors.

If you have completed this series of experiments as well as the *What's a Microcontroller?* series, connecting the parts from each will be a relatively simple matter. In addition, imagine what you can design and build using a digital keypad, a liquid crystal display, and the analog interface concepts learned. A few examples: build an interface for a factory automation process, a temperature and altitude sensor for your rocket system, or a weather monitoring station. Good luck!

8

Appendix A: Parts Listing and Sources

To use this kit you need:

- A PC running Windows 2K/XP/Vista with the BASIC Stamp Editor v2.4 or higher installed.
- A BASIC Stamp 2 module on a prototyping platform with a programming cable to connect it to your PC. Popular and convenient kit options include:
  - o The Board of Education Full Kit (#28103 serial, or #28803 USB)
  - o BASIC Stamp HomeWork Board (available in the BASIC Stamp Activity Kit (#90005)
- The Basic Analog and Digital Parts & Text Kit or Parts Kit.

Board of Education Full Kit – Serial (#28103)		
Part #	**Description**	**Quantity**
28150	Board of Education - Serial	1
800-00016	Jumper wires	10
BS2-IC	BASIC Stamp 2 module	1
800-00003	Serial programming cable	1

Board of Education Full Kit – USB (#28803)		
Parallax Part #	**Description**	**Quantity**
28850	Board of Education - USB	1
800-00016	Jumper wires	10
BS2-IC	BASIC Stamp 2 module	1
800-00006	USB A to Mini B programming cable	1

BASIC Stamp Activity Kit (#90005)		
Parallax Part #	**Description**	**Quantity**
-	BASIC Stamp HomeWork Board	1
800-00016	*What's a Microcontroller?* Parts & Text Kit	1
800-00003	Serial cable	1

A

The Basic Analog and Digital experiments require the Analog and Digital Parts Kit (#28128)

The content of the Basic Analog and Digital Parts Kit is listed below. These replacement parts are available from Parallax but may also be sourced from common electronic suppliers. In an effort to provide the best quality and most up-to-date products for our customers, you may find that some of the actual parts received with this kit may differ from those listed in the table.

Basic Analog and Digital Parts and Text Kit (#28155) Parts Only (#28128) Text only (#28129)		
Parallax Code#	**Description**	**Quantity**
150-01011	100 Ω resistors 5%	4
150-01020	1 kΩ resistors 5%	8
150-01030	10 kΩ resistors 5%	3
150-02020	2 kΩ resistors 5%	10
150-02210	220 Ω resistors 5%	2
150-02710	270 Ω resistors 5%	2
150-04710	470 Ω resistors 5%	4
152-01031	10 kΩ potentiometer	2
200-01040	0.1 µF mono-radial capacitor	1
201-01050	1.0 µF electrolytic capacitor	1
201-01062	10.0 µF electrolytic capacitor	1
201-01070	100 µF electrolytic capacitor	1
350-00006	LED, red	3
350-00007	LED, yellow	1
350-00009	photo resistor	2
400-00002	tact switch	2
602-00015	LM 358 Op-amp IC	1
604-00009	555 timer IC	1
800-00016	3" jumper wires (bag of 10)	2
900-00001	piezo speaker	2
ADCO831	ADC 0831 8-bit A/D converter	1

Appendix B: Resistor Color Code

Resistor Color Code

Most common types of resistors have colored bands that indicate their value. The resistors that we're using in this series of experiments are typically "1/4 watt, carbon film, with a 5% tolerance". If you look closely at the sequence of bands you'll notice that one of the bands (on an end) is gold. This is band #4, & the gold color designates that it has a 5% tolerance.

The resistor color code is an industry standard in recognizing the value of resistance of a resistor. Each color band represents a number and the order of the color band will represent a number value. The first two color bands indicate a number. The third color band indicates the multiplier or in other words the number of zeros. The fourth band indicates the tolerance of the resistor +/- 5, 10 or 20 %.

Color	1st Digit	2nd Digit	Multiplier	Tolerance
black	0	0	1	
brown	1	1	10	
red	2	2	100	
orange	3	3	1,000	
yellow	4	4	10,000	
green	5	5	100,000	
blue	6	6	1,000,000	
violet	7	7	10,000,000	
gray	8	8	100,000.000	
white	9	9	1,000,000,000	
gold				5%
silver				10%
no color				20%

B

A resistor has the following color bands:

Band #1. = Red
Band #2. = Violet
Band #3. = Yellow
Band #4. = Gold

Looking at our chart above, we see that Red has a value of 2.

So we write: "2".
Violet has a value of 7.
So we write: "27"

Yellow has a value of 4.
So we write: "27 and four zeros" or "270000".

This resistor has a value of 270,000 Ωs (or 270 kΩs) & a tolerance of 5%.

Index

150-01020
(8) 1 K Ω 1/4 watt 5% resistors
(brown, black, red)

150-01030
(3) 10 K Ω 1/4 watt 5% resistor
(brown, black, orange)

150-02020
(10) 2 K Ω 1/4 watt 5% resistors
(red, black, red)

150-02210
(2) 220 Ω 1/4 watt 5% resistors
(red, red, brown)

150-02710
(2) 270 Ω 1/4 watt 5% resistors
(red, violet, brown)

150-04710
(4) 470 Ω 1/4 watt 5% resistors
(yellow, violet, brown)

150-01011
(4) 100 Ω 1/4 watt 5% resistors
(brown, black, brown)

350-00006
(3) LEDs, red

350-00007
(1) LED, yellow

ADC0831
(1) 8-bit A/D converter

152-01031
(2) 10 K Ω potentiometer

900-00001
(2) speakers

350-00009
(2) photoresistors

200-01040
(1) 0.1 uF monolithic
ceramic capacitor

800-00016
(20) 3" pluggable jumper wires

201-01050
(2) 1 uF electrolytic capacitor

201-01062
(1) 10 uF electrolytic capacitor

201-01070
(1) 100 uF electrolytic capacitor

400-00003
(2) tact switches

604-00009
(1) 555 timer DIP

602-00015
(1) LM358 DIP

The components in your BASIC Analog and Digital Parts Kit may look different than those pictured here. Parts and quantities are subject to change without notice.

OCEAN CHIPS
Океан Электроники
Поставка электронных компонентов

Компания «Океан Электроники» предлагает заключение долгосрочных отношений при поставках импортных электронных компонентов на взаимовыгодных условиях!

Наши преимущества:

- Поставка оригинальных импортных электронных компонентов напрямую с производств Америки, Европы и Азии, а так же с крупнейших складов мира;
- Широкая линейка поставок активных и пассивных импортных электронных компонентов (более 30 млн. наименований);
- Поставка сложных, дефицитных, либо снятых с производства позиций;
- Оперативные сроки поставки под заказ (от 5 рабочих дней);
- Экспресс доставка в любую точку России;
- Помощь Конструкторского Отдела и консультации квалифицированных инженеров;
- Техническая поддержка проекта, помощь в подборе аналогов, поставка прототипов;
- Поставка электронных компонентов под контролем ВП;
- Система менеджмента качества сертифицирована по Международному стандарту ISO 9001;
- При необходимости вся продукция военного и аэрокосмического назначения проходит испытания и сертификацию в лаборатории (по согласованию с заказчиком);
- Поставка специализированных компонентов военного и аэрокосмического уровня качества (Xilinx, Altera, Analog Devices, Intersil, Interpoint, Microsemi, Actel, Aeroflex, Peregrine, VPT, Syfer, Eurofarad, Texas Instruments, MS Kennedy, Miteq, Cobham, E2V, MA-COM, Hittite, Mini-Circuits, General Dynamics и др.);

Компания «Океан Электроники» является официальным дистрибьютором и эксклюзивным представителем в России одного из крупнейших производителей разъемов военного и аэрокосмического назначения «JONHON», а так же официальным дистрибьютором и эксклюзивным представителем в России производителя высокотехнологичных и надежных решений для передачи СВЧ сигналов «FORSTAR».

«JONHON» (основан в 1970 г.)

Разъемы специального, военного и аэрокосмического назначения:

(Применяются в военной, авиационной, аэрокосмической, морской, железнодорожной, горно- и нефтедобывающей отраслях промышленности)

«FORSTAR» (основан в 1998 г.)

ВЧ соединители, коаксиальные кабели, кабельные сборки и микроволновые компоненты:

(Применяются в телекоммуникациях гражданского и специального назначения, в средствах связи, РЛС, а так же военной, авиационной и аэрокосмической отраслях промышленности).

Телефон: 8 (812) 309-75-97 (многоканальный)
Факс: 8 (812) 320-03-32
Электронная почта: ocean@oceanchips.ru
Web: http://oceanchips.ru/
Адрес: 198099, г. Санкт-Петербург, ул. Калинина, д. 2, корп. 4, лит. А