Earth Measurements

Student Guide for Experiments #1 through #6

Version 1.2

Note regarding accuracy of this text:
Many efforts were taken to ensure accuracy of this text and the experiments, but the potential for errors still exists. If you find errors or any subject requiring additional explanation please report this to stampsinclass@parallaxinc.com so we can continue to improve the quality of our documentation.

PARALLAX INC

Warranty

Parallax warrants its products against defects in materials and workmanship for a period of 90 days. If you discover a defect, Parallax will, at its option, repair, replace, or refund the purchase price. Simply call for a Return Merchandise Authorization (RMA) number, write the number on the outside of the box and send it back to Parallax. Please include your name, telephone number, shipping address, and a description of the problem. We will return your product, or its replacement, using the same shipping method used to ship the product to Parallax.

14-Day Money Back Guarantee

If, within 14 days of having received your product, you find that it does not suit your needs, you may return it for a full refund. Parallax will refund the purchase price of the product, excluding shipping / handling costs. This does not apply if the product has been altered or damaged.

Copyrights and Trademarks

Disclaimer of Liability

Parallax, Inc. is not responsible for special, incidental, or consequential damages resulting from any breach of warranty, or under any legal theory, including lost profits, downtime, goodwill, damage to or replacement of equipment or property, and any costs or recovering, reprogramming, or reproducing any data stored in or used with Parallax products. Parallax is also not responsible for any personal damage, including that to life and health, resulting from use of any of our products.

Internet Access

We maintain internet systems for your use. These may be used to obtain software, communicate with members of Parallax, and communicate with other customers. Access information is shown below:

E-mail:	stampsinclass@parallaxinc.com
Ftp:	ftp.parallaxinc.com and ftp.stampsinclass.com
Web:	http://www.parallaxinc.com and http://www.stampsinclass.com

Internet BASIC Stamp Discussion List

We maintain two e-mail discussion lists for people interested in BASIC Stamps. The BASIC Stamp list server includes engineers, hobbyists, and enthusiasts. The list works like this: lots of people subscribe to the list, and then all questions and answers to the list are distributed to all subscribers. It's a fun, fast, and free way to discuss BASIC Stamp issues and get answers to technical questions. To subscribe to the BASIC Stamp list, send e-mail to *majordomo@parallaxinc.com* and write *subscribe stamps* in the body of the message. This list generates about 40 messages per day.

The Stamps in Class list is for students and educators who wish to share educational ideas. To subscribe to this list go to http://www.stampsinclass.com and look for the E-groups list. This list generates about 5 messages per day.

Table of Contents

Contents

Preface

The subject of these six experiments is Earth Measurements. Think of yourself as a geologist, wanting to know more about El Niño, the famous effect in the waters off the coast of South America that changes weather patterns all over the world. You are going to need lots of measurements. Or think of yourself as the operator of a water treatment plant, where a city full of people is counting on you to deliver pure water day and night. You are going to have to monitor the water and operate a computer-controlled plant to pump it across the city. Or, think of yourself as responsible for an orchard of apples. You need to keep close track of the weather so that you will keep one step ahead on irrigation, pest control and bringing your crop to market. These are just a few examples of what we mean by Earth Measurements.

Of course Earth Measurements as a human interest were around long before microcomputers were ever imagined. Whether you think of yourself back 50 years or 500, or even 5,000 years, you can picture humans looking around and testing the wind. Computers, especially small, specialized ones allow measurements to be taken at points where no human can go, and more importantly, lots of measurements, tirelessly. Computers can summarize all that data and even put it up for a view on a worldwide computer network. They can also be programmed to close the loop, to do things like turn on a pump when a field needs to be irrigated, automatically. These were things that were hardly imagined 50 years ago.

Earth Measurements is not much different from measurement in other settings. For example home appliances such as clothes dryers, ovens and room thermostats use microcontrollers for measurement and control, as do instruments in the factory, the laboratory, the hospital, and beyond earth out into space. The techniques of measurement in these different settings are all similar. What you learn here will generalize to many fields outside of earth measurements. But let's recognize that the health of our planet depends a lot on understanding that can come from measurements. There are many interesting careers that can combine knowledge of electronics with a love for the earth environment, like being a scientist, engineer, or a modern farmer.

Audience and Teacher's Guide

The Earth Measurements curriculum was created for ages 17+ as a subsequent text to the "What's a Microcontroller?" guide. Like all Stamps in Class curriculum, this teaches new techniques and circuits with only minimal overlap between the other texts. New topics introduced in this series are a closed-loop feedback control system, serial communication, use of the BASIC Stamp's EEPROM, calibration of sensors, conductivity in water, and the use of a sound transducer for human feedback.

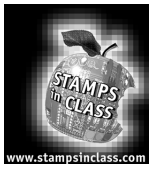
The depth and availability of a Teacher's Guide varies between the Stamps in Class curriculum. Because experts in their field independently author each set of experiments, and they are provided leeway in terms of format and content in the Teacher's Guide. The Earth Measurement Teacher's Guide will be available by e-mail request to stampsinclass@parallaxinc.com. As of August 1999 the probable date of release for an Earth Measurements Teacher's Guide is December, 1999.

Copyright and Reproduction

Special Contributors

Tracy Allen Ph.D. with Electronically Monitored Ecosystems, located in Berkeley, California wrote this curriculum (http://www.emesystems.com). EME Systems designs and manufactures instruments for environmental science. Some of their products are off-the-shelf, and others are customized systems for individual clients. The commercially available OWL2C uses a BASIC Stamp II or IISX microcontroller, providing programmable capabilities for a customer who doesn't want to use the default program. Dr. Allen has particular interest in programs that deal integrated pest management on the farm, efficient use of natural resources, and understanding of endangered species or ecosystems. A recent project of Dr. Allen's consists of measuring the surface temperature on dairy cows to evaluate milk productivity. Dr. Allen is a frequent contributor to the Parallax BASIC Stamp and Stamps in Class list servers. Parallax is very appreciative of his involvement with the Stamps in Class program.

Special thanks also to the Parallax team who provided ideas and content for the program. The Parallax staff who designs, manufacturers, and accepts orders and packages the Stamps in Class products is a key part of the Stamps in Class program.



Experiment #1: Piezo and Temperature Transducer

Temperature is the number one variable in Earth Measurements. We need only a transducer to measure the temperature, and another transducer to convey the temperature readings to our eyes and ears.

In this first experiment, you will start off by experimenting with a transducer that converts electrical impulses from the BASIC Stamp into musical tones. This audio annunciator will provide useful feedback about the operation of the BASIC Stamp throughout this series of experiments. The main project in this experiment will be to install a digital temperature sensor on your Board of Education. This too is a transducer that converts temperature into a coded form that the BASIC Stamp can understand. The BASIC Stamp will take temperature readings and display them on the computer screen. In this series of experiments, we are going to measure temperature in two quite different ways, to help understand how the versatile BASIC Stamp is used to approach the task.

Temperature is of the first importance in Earth Measurements. You are probably sitting in a comfortable room, in the range of 17 to 30 degrees Celsius (63 to 86 degrees Fahrenheit). There may be a thermostat in the room that holds the temperature at that comfortable value, using a heater or an air conditioner (or maybe not!?) What do you think the temperature is right now where you are? How about outside? If you don't have a thermometer, don't worry; you will have one before this experiment is over. We all know from personal experience that temperature is important to our well being.

We live on a planet that is just the right distance from the sun and has the right kind of atmosphere to offer temperatures conducive to life, as we know it. Through our human technology and industry, from clothing and housing all the way through to modern electronic environmental controls, we have extended the range of temperatures where we can live.

It is not far-fetched to say that every process on earth depends on temperature in some way. Think of erosion of mountains. Every year water seeps into the cracks, it freezes, expands, and breaks off pieces. Snow, rain, clouds, wind. Nearly every aspect of the weather depends critically on temperature. A few tenths of a degree change in the temperature of the water in the South Pacific Ocean (El Niño) can affect the weather all over the world. How apples grow on trees, how the worms grow in the apples, how mosquitoes thrive in stagnant pools, how tadpoles survive to eat the mosquitoes, everything relating to agriculture and biology is dependent on temperature. Add to that the environment in factories, hospitals, laboratories, schools, homes, museums, and on and on. Suffice it to say that if you want to go into any career related to the environment and microcontrollers, you are going to have to know how to measure temperature.

Parts Required

In this experiment you will need the following parts in addition to a BASIC Stamp II and Board of Education:

(1) piezo transducer
(1) DS1620 Temperature Sensor
(1) 1K Ω resistor (brown black red)
(1) 0.1 µF capacitor
(several) jumper wires

Build It!

It's always good to start out with a simple project, just to get into the swing of things. That is going to be the pattern in this series of experiments. You will start out with a warm-up project, and then move on to the main focus of the experiment. The warm-up to start this experiment is simply a buzzer, a sound output device. In fancy terms, it is an "annunciator", or a "**piezoelectric transducer**". It will be a big part of our user interface in the projects to come in this set of these experiments on Earth Measurements. Sure, we can also see results on the computer screen when the Board of Education is hooked up to it via its umbilical cord. Having the annunciator will allow us to stand up and walk away from the computer, around the room, into the dark, outside into the sunlight, and still be able to "hear" what is going on.

The piezoelectric transducer you will find in your parts kit is a black plastic cylinder with two pins sticking out the bottom and a sound hole in the top. The top of the case above one of the pins is labeled with a + sign. Hook it up to your Board of Education as shown in the pictorial in Figure 1.1.

What's a...

Piezoelectric transducer:

Piezo comes from a Greek word that means "to squeeze or press", and electric comes from a Greek word that refers to amber, a mineral that can accumulate a charge of static electricity when rubbed. Crystals, such as quartz and also some ceramic and plastic materials generate electricity when they are flexed back and forth. This is the piezoelectric effect. Electrical wires attached to the surface of the material can pick up that electricity. This is the basis of some kinds of microphones. A microphone is a transducer (Latin for "lead across") that transforms sound into electricity. The piezoelectric effect works in reverse too. If electricity is applied across some piezoelectric materials, they bend in response. They can be fabricated as a thin disk, with electrical connections on both faces, and wires attached. The disk is like a tiny drumhead. When connected to an rapidly alternating electrical voltage, it flexes back and forth, compresses the air and emits sound waves, and becomes a piezo transducer. It turns electricity into sound. The electrical voltage has to be in the right frequency range to resonate with the natural tone of the tiny drumhead. Sometimes a piezo transducer is packaged along with some electrical circuitry, so that all you have to do is connect it to a battery or to a power supply and it buzzes at one preset pitch. Such a device is called a piezo buzzer. The device we are using here is a simple piezo transducer. It will not buzz if we connect it directly to a battery. It will only produce sound when we provide audio frequency electrical impulses from the BASIC Stamp.

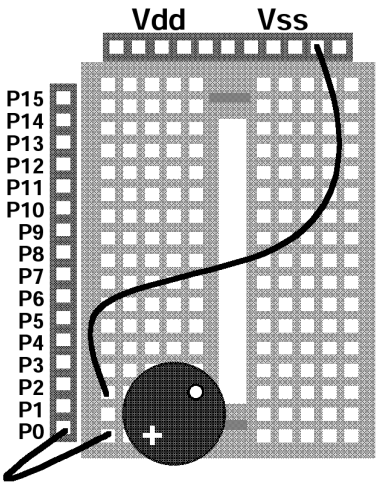


Figure 1.1: Piezo Transducer Pictorial

- Piezo on an angle as shown
- Pin under + mark wired to P0
- Other pin wired to Vss

Please note. The six experiments in this Earth Measurements series will progress from unit to unit by adding new circuits onto the old ones already built on the Board of Education. To avoid having to rewire things later, please follow the suggested parts placement.

The schematic is shown in Figure 1.2.

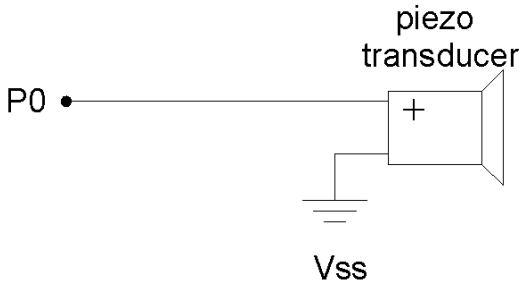


Figure 1.2: Piezo Transducer Schematic
Electrical schematic of pictorial shown in Figure 1.1

Program it! This experiment consists of three smaller sections: piezo transducer, Morse code, and temperature measurement. The project is progressive.

Temperature Transducer

Now, to make noise with the piezo transducer, the BASIC Stamp has to supply a high frequency signal from P0. The PBASIC command to do this is `freqout`. That's short for "frequency output". Start the BASIC Stamp editor on the PC and type in the following PBASIC program:

```
freqout 0,1000,1900
```

In case of difficulty during download:

In the Windows and DOS versions of the software, if RUN gives you a message about "hardware not found" or "communication error", then check to be sure that the cable that connects the PC to the Board of Education is okay. Also check to be sure that the Board of Education has a good power supply and that the power supply indicator light on the Board of Education is glowing.

If you see a message indicating an error in your program, then check your typing. If the program is okay and ALT-R is accepted without an error message, but it simply won't work, then check the wiring on your Board of Education. Compare it to the wiring shown on the pictorial.

That's it. A one-line program.

Now be sure the Board of Education is connected by its cable to the PC. While holding the "ALT" key down, press the letter "R", for "run" (DOS version - in Windows it's ALT/R and "enter"). This should be familiar if you went through the "What's a Microcontroller?" experiments. If all is well, you should hear a high pitch beep. Each time you press the reset button on the Board of Education, you will hear it again. The reset button is found on the Board of Education near one corner, and is clearly labeled Reset. You can press it as often as you want, no worries. Pressing the button starts your program over again but will not erase it.

There are three **parameters** in the `freqout` command:

```
freqout 0,1000,1900
               ^^^^-selects an output frequency of 1900 hertz
           ^^^^-----makes the tone last 1 second=1000 milliseconds
        ^^----------uses P0 signal line for the tone
```

If we could observe the voltage on P0 during the `freqout` command, we would find that it goes back and forth from 0 to 5 volts very rapidly, and what comes out is fundamentally a 1900-hertz sine wave that lasts for 1 second. If you have a handheld multimeter then try to measure the frequency. For more information about how it works, see the explanation of the `freqout` and `PWM` commands in the BASIC Stamp Manual Version 1.9.

What's a... Parameter:

A parameter is a number that governs the behavior of a command or a process. In the freqout command, the parameters of the command specify what pin to use, how long the sound will be, and what the frequency will be. The word parameter is one you will often hear in the fields of science and engineering. For example: air temperature is a parameter that determines how fast paint will dry. Or, the global average temperature of the earth's atmosphere is one parameter that determines how much water is in the ocean versus locked in the ice caps, and how much coastline is exposed or submerged.

Now it's time to experiment. Modify the program by replacing the 1900 with 3800 to give it a higher pitch:

```
freqout 0,1000,3800
```

Download the program to your BASIC Stamp. Pay attention that what you hear is a higher pitch. Try it a couple of times, one way and then the other. Don't be afraid you are going to wear out the BASIC Stamp by reprogramming it lots of times. You can reprogram the BASIC Stamp at least a million times.

```
freqout 0,2000,3800
```

And this, to make the `freqout` command play two tones at once:

```
freqout 0,2000,1900,2533
```

The number 2533 is equal to 1900 times 4/3, the musical interval "fourth". The BASIC Stamp can play two tones at once, but that's it, no three part harmony.

And try the following:

```
freqout 0,2000,1900,1903
```

How do you explain what you hear?

And try a very short duration, to make a click 2ms long:

```
freqout 0,2,1900,3804
```

Feel free to experiment. By experimenting with individual BASIC Stamp commands, you can become aware of possibilities that may be of use in programs later on.

Morse Code

An "audio annunciator" is a device that gives sound feedback about what is going on in a system. Having an audio annunciator on the Board of Education is going to be very useful throughout these experiments on Earth Measurements. In Experiment #2, we will program it to send numbers using Morse code, and use the code to annunciate the temperature readings. Morse code is a fine way to send messages using sound. Here are the Morse code numerals from 0 to 9:

Numeral:	Morse Code	Binary
0	dah dah dah dah dah	11111
1	dit dah dah dah dah	01111
2	dit dit dah dah dah	00111
3	dit dit dit dah dah	00011
4	dit dit dit dit dah	00001
5	dit dit dit dit dit	00000
6	dah dit dit dit dit	10000
7	dah dah dit dit dit	11000
8	dah dah dah dit dit	11000
9	dah dah dah dah dit	11110

The Morse code is based on sending patterns of short and long sounds. The long sound is always three times as long as the short sound. The short sound is called "dit" and the long sound is called "dah". The numerals are all made up of five dits and dahs. The letters of the alphabet have from one to four sounds, and the most common letters have the shortest patterns (for example, e=dit, t=dah, s=dit dit dit, q=dah dah dit dah). Punctuation has six sounds, e.g. period=dit dit dah dah dit dit. Within one letter or numeral, the time between sounds is supposed to be the same length as the dit. And the time between different digits in a sequence like "50" is supposed to be the same length as a dah. The "binary" column is there just to show how you might think of Morse code as a binary number.

In these experiments, we will use only the numerals. Try this simple program that sends the two-digit number "50" as Morse code. You do not have to type in the remarks. Recall that remarks are the apostrophe " ' " and everything that follows it on the line.

```
dit          con    70     ' a short span of time in milliseconds
dah          con    3*dit  ' a longer time, 3 times the above
i            var    nib    ' index
for i=1 to 5               ' send 5 sounds
  freqout 0,dit,1900       ' send a dit
  pause dit                ' short silence
next
pause dah                  ' a longer silence between digits
```

```
for i=1 to 5              ' send 5 sounds
  freqout 0,dah,1900      ' send a dah
  pause dit               ' short silence
next
```

Run the program. Press the reset button on the Board of Education if you want to hear the number 50 again. How could you modify your program to send the most famous Morse code message of all, SOS?

You should already be familiar with the `for. . . next` loop from the "What's a Microcontroller?" text. Think about how the program incorporates the rules of the Morse code. Note how it starts off by defining a constant dit in milliseconds, and then dah is defined as three times dit. PBASIC allows you to do that, to define one constant mathematically in terms of another. That's convenient, because it allows you to change the overall speed by changing only the "dit" constant, and "dah" will fall into place.

In your program, change the dit constant from 70 to some other value, twice or half as long, and listen to the effect on the overall speed. The important thing is that the ratio between the dit and the dah is always going to be 1:3.

This is only an introduction. We will write a serious Morse program in experiment two, to annunciate temperature readings.

Temperature Readings from the DS1620

Now for a complete change of pace. Let's move on to the main topic, to acquire some temperature readings. In engineering, we usually use the word acquire, instead of get when we refer to data or readings. The Board of Education is going to become our data acquisition system.

The DS1620 is a modern temperature transducer (portions of the DS1620 data sheet are included in Appendix D). There is that word, transducer again. Here, it refers to a device that transforms temperature into an electrical signal. The DS1620 takes temperature as its input, and transduces that value into a digital code that the BASIC Stamp can understand. The digital code represents the temperature of the DS1620 chip.

The DS1620 comes in an 8-pin plastic package. Plug it into the Board of Education and hook it up as shown in the pictorial in Figure 1.3. A word to the wise--When you change the wiring on the Board of Education, it is a good idea to disconnect the battery or power supply. It is all too easy to touch a wire in the wrong place and risk burning something out. Double check your wiring, or better yet, have someone else check it, before you reconnect the power. The schematic for the DS1620 is shown in Figure 1.4.

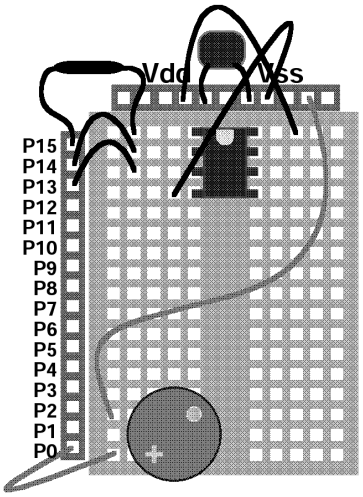


Figure 1.3: DS1620 Pictorial
Plug the DS1620 in at the very end of the BOARD OF EDUCATION. Observe that there is a notch at one end of the DS1620 package to indicate the polarity. Be careful not to reverse the power supply.

- 0.1 uF capacitor from Vdd to Vss
- DS1620 pin 4 wired to Vss.
- DS1620 pin 8 wired to Vdd
- DS1620 pin 1 wired through 1K ohm resistor to BASIC Stamp P15
- DS1620 pin 2 wired to BASIC Stamp P14
- DS1620 pin 3 wired to BASIC Stamp P13

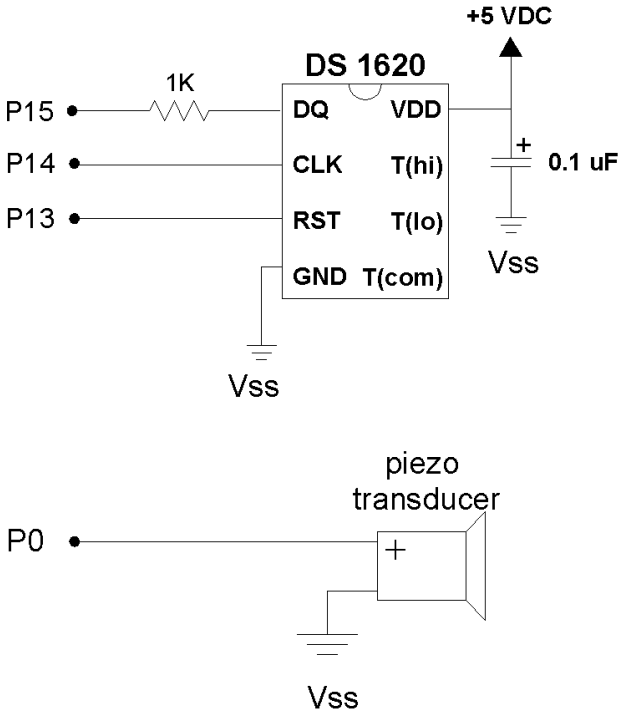


Figure 1.4: DS1620 Schematic
Schematic of the circuit pictured above. Remember – the piezo transducer portion of the circuit has already been built.

Plug the DS1620 in at the very end of the Board of Education. Observe that there is an **indicator** at one end of the DS1620 package, to indicate where pin 1 is connected. Be careful not to reverse the power supply connections!

Now it's time to program the DS1620, literally. The DS1620 is itself a little computer. More accurately, it's a smart sensor. It can remember certain settings and do some pretty nifty tricks all on its own. Smart sensors are being used more and more in electronics and in the field of environmental monitoring and control.

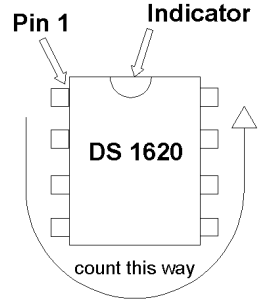
Enter the following program. Again, you don't have to enter the remarks after and including the '.

```
low 13                              ' Puts the DS1620 in the waiting state
freqout 0,1000,3800                 ' sound shows us the program is running
high 13                             ' Tells the DS1620 that a command is coming
shiftout 15,14,lsbfirst,[12,2]      ' Command  to set DS1620 configuration 2.
low 13                              ' Completes the command cycle.
end                                 ' end of program
```

Double check your typing. Download your program.

What's a... Indicator:

The DS1620 is an 8-pin DIP package. The indicator denoting the DS1620's pin 1 is a small notch on top of the chip. On parts like the DS1620 and BASIC Stamp, the pins are always counted counterclockwise starting from the mark. The mark can be a bump, round depression, notch, beveled edge, etc.



You will hear the one-second tone. That's all. But a lot has happened. The `shiftout` command sends two bytes 12 and 2 to the DS1620. The 12 is a command to the DS1620 to get ready for the configuration, and the 2 is the actual configuration. Here are the four possible configurations:

0: No CPU, continuous conversion
1: No CPU, one-shot conversion
2: Yes CPU, continuous conversion
3: Yes CPU, one-shot conversion

What? By selecting configuration 2, we are telling the DS1620 that we want it to send its readings to a CPU (Central Processing Unit--the BASIC Stamp). The alternative is for it to sit there and monitor temperature on its own, and not send back any readings. What good would that be? We asserted that the DS1620 is a smart sensor. Those other pins we are not using on the DS1620 could be wired up to a fan or heater, and set to regulate the temperature in a room or in a terrarium. The DS1620 also has a command that allows you to set a desired temperature. You will hear more about regulation of temperature in Experiment #6. But that is the way we are using it here, and we have not connected anything to those pins.

By setting the CPU option, the DS1620 will send data back on the serial line when it receives commands. The term "continuous conversion" means that it will read temperature over and over and always have a current value available. The term "one-shot" (which we are not using) means that it will read the temperature once and then stop until it receives a new command. The one-shot mode is used when an engineer needs to get the best battery life.

Now that we have sent the configuration, the DS1620 will not forget the setting. It is stored in memory inside the DS1620 in a kind of memory (EEPROM, like the BASIC Stamp program memory) that is not lost when the power supply is turned off.

The heart of the above PBASIC program is the `shiftout` command. The sequence is an example of synchronous serial communication. It will pay for you to understand how it works. Lots of modern electronics, found in everything from pagers to satellites use these ideas. One main reason for this popularity is that devices that use serial communication can be made very small, because there don't have to be many wires connecting them. Here are the parameters of the command.

```
high 13                      <---   this is really part of it, the chip select.
shiftout 15,14,lsbfirst,[12,2]
                           ^^^^---  two bytes sent from the Stamp to the DS1620
                    ^^^^^^^-------  the bytes are sent least significant bit first
                ^^------------------ P14 on the BASIC Stamp is the clock
           ^^---------------------- P15 on the BASIC Stamp is used to send the data bytes
low 13                       <---   this is part of it too, ends the session.
```

To explain how it works, I'll try an analogy using a stick figure dance. Please refer to Figure 1.5. The BASIC Stamp is the one with the round head. The DS1620 is the blockhead. The DS1620 starts off with a zero as its configuration in memory.

The BASIC Stamp starts the SHIFTOUT dance by raising the left hand. That is a wake-up call to the DS1620, and it means get ready, this message is for you. Then the BASIC Stamp taps out the first 8 beats on the clock pin with its foot. On each tap, the BASIC Stamp holds his right hand either low to signal a zero, or high to signal a one. Those are the digits of a binary number, sent out, least significant bit (lsb) first on the data pin.

The DS1620 watches BASIC Stamp's right hand at each tap. After eight taps, DS1620 has the binary number 12 and recognizes it as a command. The BASIC Stamp knows in advance that DS1620 will interpret 12 as a command. (The command set is determined by the engineers at Dallas Semiconductor, the manufacturer of this part).

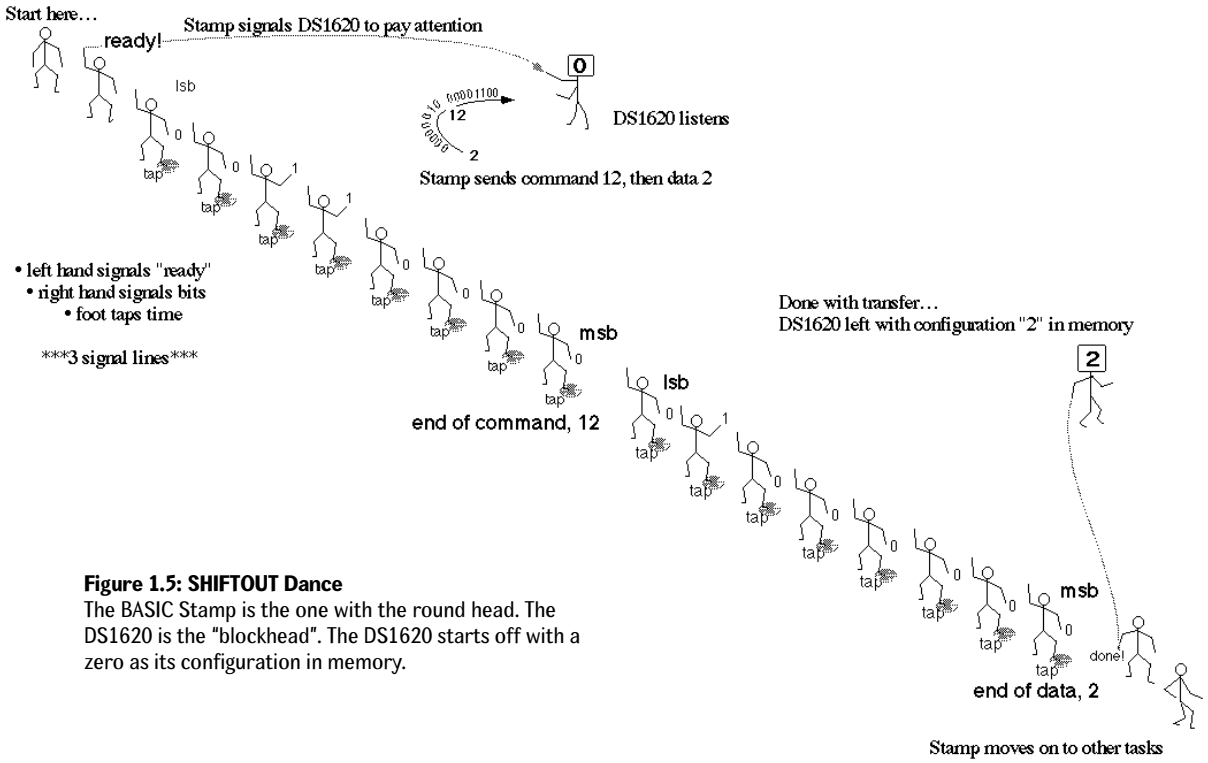


Figure 1.5: SHIFTOUT Dance
The BASIC Stamp is the one with the round head. The DS1620 is the "blockhead". The DS1620 starts off with a zero as its configuration in memory.

It isn't over yet. The DS1620 is now waiting for another binary number to follow the 12. The BASIC Stamp taps out 8 more beats, and DS1620 watches BASIC Stamp's right hand at each tap. This time it gets the number 2. The DS1620 stores the 2 in its EEPROM memory. Now the DS1620 is configured. The BASIC Stamp puts down its left hand to signal that the sequence is finished. The BASIC Stamp and the DS1620 are no longer in communication. All that signaling is taken care of automatically, in less than 1/1000 second, by the `shiftout` command. Figure 1.6 shows the same thing as an engineer might draw it.

Figure 1.6: Timing Diagram
Engineer's timing diagram of the SHIFTOUT command as executed from the BASIC Stamp.

```
time: ------------------------one millisecond---------------------->
P13: 0000555555555555555555555555555555555555555555555555555555555000 CS
P14: 0000005005005005005005005005000000500500500500500500500500000 Clock
P15: 0000000000055555500000000000000000005550000000000000000000000000 Data
          0  0  1  1  0  0  0  0      0  1  0  0  0  0  0  0
         ^-command 12---------->    ^-data 2--------------->
```

Note that 12 decimal = 00001100 binary, and 2 decimal is 00000010 binary.

When reviewing the timing diagram from Figure 1.6 consider the following:

- P13 starts the exchange by going from 0 to 5 volts. The command ends when P13 goes back down from 5 to 0 volts. P13 is often called the chip select or chip enable.
- P14 is the clock and puts out a series of 16 pulses, 0 to 5 volts, in two groups of 8.
- P15 is the data line and puts out either 0 or 5 volts in each time slot, synchronized with the clock pulses on P14. The first group forms the 12 (00001100 in binary), and the second group forms the 2 (00000010) in binary.
- Note the "lsbfirst" parameter for the `shiftout` command. The least significant bit comes first in the time sequence.
- This whole transmission of 16 clock cycles takes about 1 millisecond, 1/1000 of a second, and it happens automatically under the `shiftout` command.

If you want more explanation, please refer to the BASIC Stamp Manual Version 1.9 where it describes the `shiftout` command, where there is also an application note. This is called synchronous serial communication, because the data is synchronized with the clock ticks that come from the BASIC Stamp. The BASIC Stamp is commonly referred to as the master and the DS1620 as the slave. That is because the clock pulses and commands originate in the BASIC Stamp.

Now for the main event: to read the temperature from the DS1620. Enter the following program.

```
x     var byte                  ' define a general purpose variable, byte
degC  var byte                  ' define a variable to hold degrees Celsius
                                ' note: DS1620 has been preprogrammed for mode 2.

outs=%0000000000000000          ' define the initial state of all pins
     'fedcba9876543210
dirs=%1111111111111101          ' as low outputs

freqout 0,20,3800               ' beep to signal that it is running
high 13                         ' select the DS1620
  shiftout 15,14,lsbfirst,[238] ' send the "start convertions" command
low 13                          ' do the command
  loop:                         ' going to display once per second
  high 13                       ' select the DS1620
  shiftout 15,14,lsbfirst,[170] ' send the "get data" command
  shiftin 15,14,lsbpre,[x]      ' get the data
  low 13                        ' end the command
  degC=x/2                      ' convert the data to degrees C
```

```
  debug ? degC                  ' show the result on the PC screen
  pause 1000                    ' 1 second pause
goto loop                       ' read & display temperature again
```

Download the program to your BASIC Stamp.

The debug screen should appear, and you should see the current temperature readings appear once per second. The readings are in units of degrees Celsius. If you hold your finger on top of the DS1620 chip, you should see the temperature rise.

In case of difficulty from an error in your program:

If you get a message about an error in your program, you may have typed something wrong. The Stamp editor program will position the cursor near where the error occurred. Do not take the wording of the error message literally. Sometimes the wording of the error message is not appropriate. Look for any error near the cursor. If the error message you see is about "hardware not found" or "communication error", then be sure your Board of Education is powered on (green light on the Board of Education?!) and that cable to the PC is connected properly. If all that goes okay, but the program does not work, then you will have to decide whether the problem is in the program or in your wiring of the DS1620.

Now, what is the room temperature where you are working?

Observe that when you hold your finger on the chip or when you put it under a lamp or in the sun, it takes some time for it to heat up and for it to cool down. Once you heat it up, observe that you can cool it down faster by fanning air across it. What is the temperature near the floor? On top of your PC? Next to your body? Is it different from the temperature up high? Try to experiment.

Which one of these temperatures (if they are different) will be the one you call the room temperature? Usually, HVAC engineers (Heating, Ventilation and Air Conditioning) prefer to use a temperature reading that is taken in the shade at a position not too close to sources of heat, like computers and bodies. This is called a representative temperature. In the real world, there can be lots of variation over even small distances and short times. You always have to make some choice about where and when is the best place and time to make a measurement.

What is going on in the program? First a word about the outs and dirs statements:

```
outs=%0000000000000000          ' define the initial state of all pins
     'fedcba9876543210
dirs=%1111111111111101          ' as low outputs
```

When using the BASIC Stamp, or any microcontroller, there will be pins connected to the outside world, and those pins can be either an input or an output, and if it is an output it can be either output high or output low. You are already familiar with the `out` and `dir` variables from the "What is a Microcontroller?" series. Here, with an "s" on the end, the statements control all 16 I/O pins, numbered from 0 to f (Note the apostrophe in front of the "f"--above that makes it a remark – and it is just there for reference.) The BASIC Stamp I/O pins are numbered from P0 to P15, where a=10,...,f=15.

It is good programming practice to start off every serious program by putting all of the microcontroller pins into a known, desirable state. When the BASIC Stamp is first turned on or reset, all of the pins are configured by default as inputs. This is a fine state for a microcontroller to start up in. You, the designer, are in charge of making the pins outputs as needed. On the other hand, if a pin is not connected to anything, it is not a good idea to leave it as input. Unconnected inputs may cause the microcontroller to behave erratically or to draw excessive power from the battery. The above instructions turn all of the pins on the BASIC Stamp into LOW outputs. That is what we want at first for the piezo transducer and for the DS1620. All the other pins are made low outputs just as a matter of principle. Reasons to do otherwise will arise we progress through these experiments. For more information on the `dirs` and `outs` command, please refer to the BASIC Stamp Manual Version 1.9, page 216.

The main action in the temperature program comes from the `SHIFTOUT` and `SHIFTIN` commands.

The first `shiftout` should look familiar. You see the familiar sequence: It sets P13 high, and then sends one byte, 238, out to the DS1620, and then sets P13 low again to end the sequence. Inside the DS1620, the 238 is a command that tells it to start converting temperature into digital codes. The 238 command needs to be sent at least once after the DS1620 is powered on. Unlike the configuration command, this one is not stored in the permanent memory of the chip.

Next comes the heart of the routine, to read the temperature from the DS1620. Again you see the familiar sequence: It sets P13 high, and then sends one byte, 170, out to the DS1620. So far so good. The DS1620 interprets the 170 as a command for it to get the current temperature reading and send it back to the BASIC Stamp. Now things get interesting. The DS1620, in response to the 170 command, takes control of the data line. The BASIC Stamp moves on to the `shiftin` command. Here are the parameters:

```
shiftin 15,14,lsbpre,[x]
                      ^--------   name of the variable to receive the variable
               ^^^^^^------------ the bytes are received least significant bit first
             ^^------------------ P14 on the BASIC Stamp is the clock
          ^^-------------------- P15 on the BASIC Stamp is used to receive the data bytes
low 13                    <--    end the command
```

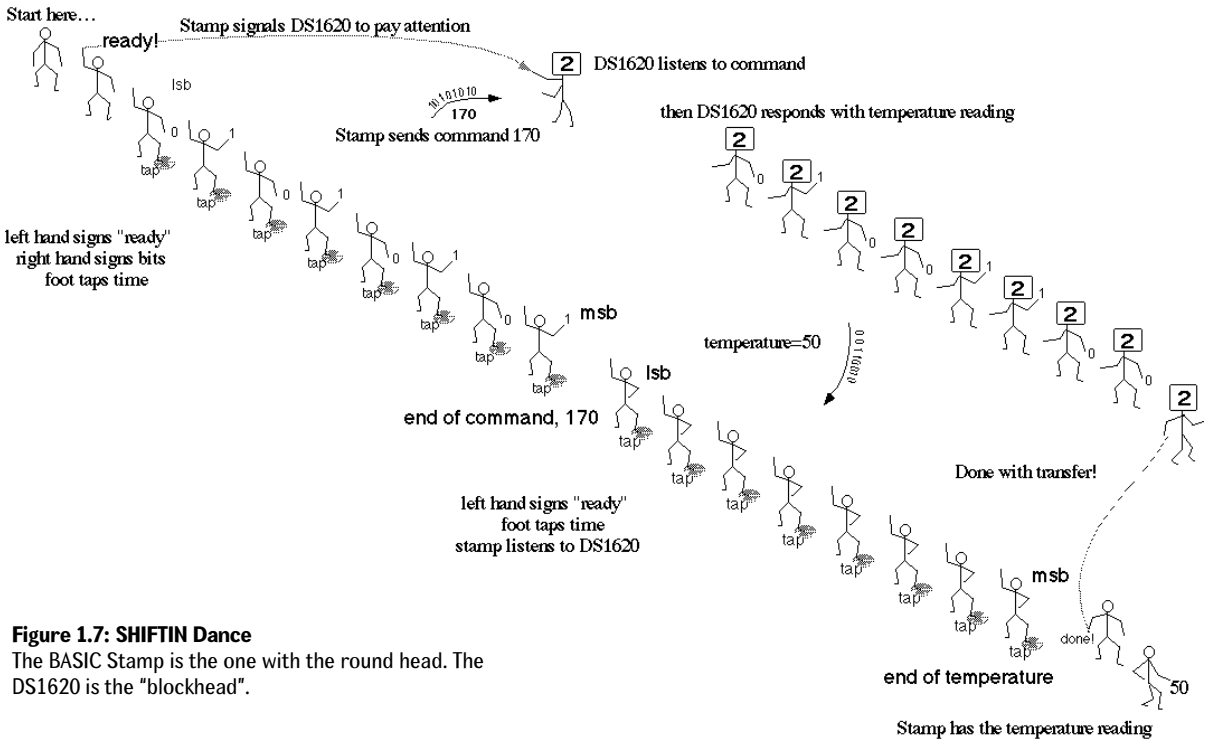


Figure 1.7: SHIFTIN Dance
The BASIC Stamp is the one with the round head. The DS1620 is the "blockhead".

P15 on the BASIC Stamp is now an input, whereas for `shiftout` it was an output. The BASIC Stamp is now ready to receive data from the DS1620. This is diagrammed in Figure 1.7, and as an engineering timing diagram in Figure 1.8.

Observe that the BASIC Stamp is still in charge of the timing. The BASIC Stamp is still the master and the DS1620 is the slave. Here is the timing diagram:

Figure 1.8: Timing Diagram for SHIFTIN

```
time: ------------------------one millisecond---------------------->
P13: 00005555555555555555555555555555555555555555555555555555555000  CS
P14: 00000050050050050050050050050000005005005005005005005000000  Clock
P15  00000000555000555000555000555000000005550000005555550000000000  Data
          0  1  0  1  0  1  0  1      0  1  0  0  1  1  0  0
         ^-command 170---------->    ^-data 50-------------->
          BASIC Stamp to DS1620         DS1620 to BASIC Stamp
```

Each time the BASIC Stamp sends out a pulse on the clock line P14 (taps its foot), the DS1620 signals the next bit of the temperature byte. It starts with the least significant bit first. The `lsbpre` means that the BASIC Stamp looks for the least significant bit before it sends out the first clock pulse. It goes like this, get 1st bit, pulse clock, get second bit, pulse clock, and so on until it has all 8 bits. The BASIC Stamp stores the data it receives from the DS1620 in the variable, `x`.

If the temperature is 25 degrees Celsius, the DS1620 sends back the value 50, which is two times the temperature. In binary, 50 is 00110010. The bytes that the DS1620 sends out are always two times the temperature. If the temperature is 25.5 degrees C, then the byte that the DS1620 sends back will be 51. Each step in `x` represents 0.5 degrees C. That is the resolution, the smallest change in temperature that the sensor detects.

Our program then converts the raw value of `x` to temperature:

```
degC=x/2                        ' convert the data to degrees C
```

The BASIC Stamp uses integer arithmetic. It throws away the 0.5 degree remainder. Both 50/2 and 51/2 come out as degC=25, and 52 and 53 both come out as degC=26, and so on. There are ways to keep the half degree resolution, but we won't pursue that here. (But you can do so as a challenge!)

The temperature is sent to the debug screen by this command:

```
debug ? degC                    ' show the result
```

The "?" makes the BASIC Stamp send "`degC=`" and then the actual value of `degC` to the debug display screen, with each entry on a new line.

What's a...

Operational limit:

The DS1620 is perfectly capable of measuring temperatures below zero, down to -25. That would be important if you were out doing research on snow in Alaska, or if you were designing a control system for a freezer. The trouble is, the program we just wrote does not handle negative temperatures correctly. I.e., when the temperature goes to -1 degrees C, our reading would be degC=127 instead of degC=-1. In order to read negative temperatures, we would have to take a couple more steps, that would complicate the program more than we want to get into at this time. As it stands, zero degrees is the operational limit on the low end. Operational limits are everywhere in engineering, and they come up for all kinds of reasons, both in the software and hardware and in the properties of materials. This particular operational limit comes from a short cut we have taken in writing the software. That will be justified so long as the temperature is above freezing, but becomes a "bug" if we try to go below freezing. A famous software operational limit is (was!) the Y2K bug, where a software shortcut taken in the latter half of the 20th century led to an operational failure or glitches in the year 2000.

Now for a valuable experiment, you should save the program you have just typed in and debugged. In this series of experiments, we are going to build up a large program, one piece at a time. This is the first piece you will be able to reuse.

If you didn't do so already, you may want to enter the remarks attached to the program. That will reinforce your understanding, and it will also make it easier for you to pick up the program the next time you look at it, in Experiment #2.

Decide what you want to name the program. Your instructor will have directions, depending on how your class is set up to share the PCs. The program will have an extension of "BS2". Let's say you decide to name the program "DS1620.BS2"

This is how you save the program in the DOS and Windows versions of the Parallax BASIC Stamp editor:

STAMP2.EXE (DOS):

Type ALT-S, holding down the ALT key while tapping "S". A dialog box will appear for you to enter the program name. Type in the name, and press ENTER. That's it.

STAMPW.EXE (Windows):

Go to File/Save, then navigate to the directory where you want to save the program, type in the name, and press enter or click Save.

Challenge!

1. Write a program using a sequence of `freqout` commands to plays a simple tune. Look up the `freqout` command in the BASIC Stamp Manual Version 1.9. You will find an example of how to play Mary Had a Little Lamb. Okay, you can try Stairway to Heaven, or Beethoven's 5th, if you prefer. You will discover some of the high fidelity limitations of the piezo transducer.

2. Define a variable degF for Fahrenheit. Display both degrees Celsius and degrees Fahrenheit on the debug screen. Use either formula:

   ```
   degF = degC * 9 / 5 + 32    or            degF = x * 9 / 10 + 32
   ```

   Is one formula better than the other? Why? Observe how the readings change as you gradually change the temperature of the DS1620 chip.

3. Display degrees Celsius resolved to 0.5 degrees. Recall that the result that comes from the DS1620 is a binary number where each bit represents 0.5 degrees. To get degrees, we divided by 2 and lost a bit of information (literally, one bit). You can display the result as 205 to represent 20.5 degrees C. Hint: multiply by 5 instead of divide by 2.

4. If the temperature is greater than (you choose a value), play an alarm tone on the piezo transducer. Make the alarm stop when the temperature goes back down. Then modify it so that the alarm continues, even when the temperature goes back down. Under what circumstances would each kind of alarm be appropriate?

Experiment #2: Data Logging

The theme of the Data Logging experiment is best answered by the question: What is data logging and why it is important in earth measurements? The activities of this experiment are: (1) Design a user interface by adding a pushbutton to your existing setup on the Board of Education, then implement single click, double click and long click to do different things; (2) Learn the basics of `read` and `write` with the BASIC Stamp's EEPROM; and (3) Implement a "talking (Morse code) thermometer".

Constancy punctuated by change: that is one prevalent view of the natural world. In order to understand and predict events, people often need to keep a record of variables that affect the action. In the field of earth measurements, the data logger, or data acquisition system, or DAQ for short, is an essential tool. It is a machine that automatically takes readings and stores them at regular intervals of time (or on some other basis) into the memory of the machine for later retrieval.

Data is stored in a log file. The term comes from nautical history, where readings of position and depth soundings on a ship were regularly noted in the Captain's log-book. In fact, some data was collected by throwing a log (not the book!) off the bow of the boat and counting the time it takes to reach the stern of the boat. Then they could calculate speed.

These days, much logging is done by computers with sensors attached. Computers are well suited to data logging - they never get bored or tired, and they can work reliably and very rapidly if required. It can be difficult, boring, or downright impossible for a real human being to exist in the place and time where data needs to be collected. Data loggers are found out on buoys floating in the ocean, high on windy mountaintops, on spacecraft, in collars on grizzly bears, in the stomachs of whales, out in orchards and vineyards, and in innumerable industrial settings.

Another "buzz word" these days is SCADA, for Small Computer Aided Data Acquisition. That usually refers to something fancier, a network of sensors and computers, but the general idea is the same. Data loggers may even communicate to a central hub via TCP/IP connections to the internet, or via long-distance radio links.

In this experiment you will learn important details about the EEPROM memory in the BASIC Stamp II. This is in preparation for logging readings of temperature, light and water level in the experiments to come. Also, you will improve on your DS1620 thermometer from the previous experiment, and make it talk (in Morse code). And as a warm-up, you will work with one pushbutton and the piezo beeper, to make a user interface.

Everyone who has a computer understands what you mean by a mouse, and the actions of click, double-click, and click-and-hold. These actions are central to the modern computer's user interface. Have you ever wondered how a program implements those actions? How hard would it be to implement them on the BASIC Stamp? Well, it is not too hard at all, and we are going to do it, to enable one button on the Board of Education to perform multiple tasks. There is not going to be room for multiple pushbuttons. One button, along with feedback from the piezo transducer, is going to have to do it all for our user interface when the Board of Education is not docked to the PC.

One button, one buzzer.

Parts Required

The Earth Measurements experiments are progressive and build on the previous projects. Therefore, you'll be adding parts to your Board of Education. This experiment requires the following parts:

- pushbutton
- 10K ohm resistor
- jumper wires

Build It

In Experiment #2 of What's a Microcontroller, "Detecting the Outside World", you learned how to use two buttons to make decisions, to control two light emitting diodes. In this experiment you will build on that project and on the previous Earth Measurements experiment. You already have an audio annunciator for output. Now, install a pushbutton for input, as shown in Figure 2.1. The schematic is shown in Figure 2.2.

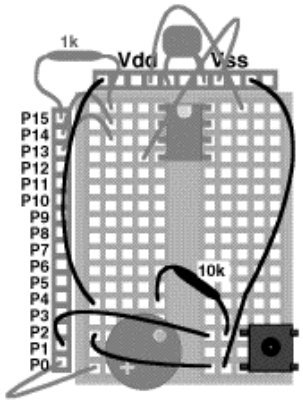


Figure 2.1: Pictorial
Install a pushbutton (PB) at the very end of the Board of Education, across from the piezo transducer. Two of the pins will hang over the edge of the plastic terminal block, so as to leave a couple of holes free for wiring, as shown. If you straighten the button's pins you may be able to fit it in the entire lower right-hand 3x3 square of holes.

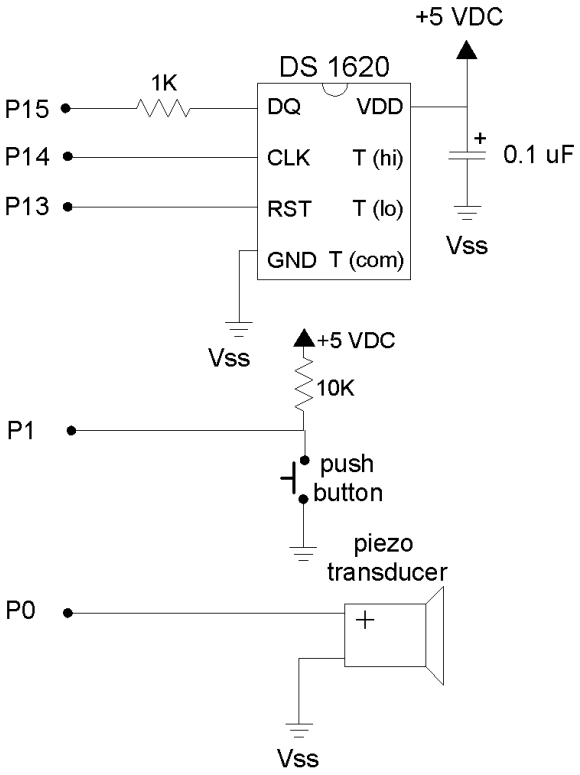


Figure 2.2: Schematic
Install a pushbutton (PB) at the very end of the BOARD OF EDUCATION, across from the piezo transducer. Two of the pins will hang over the edge of the plastic terminal block, so as to leave a couple of holes free for wiring, as shown. Hold the loose end of the pushbutton in place with a piece of strong plastic tape. Wire it up as follows.

- Vss to pushbutton (moved from piezo -)
- pushbutton wired to piezo (-)
- Vdd (+5 volts) wired to row next to piezo
- 10k ohm from Vdd row to pushbutton (+)
- pushbutton (+) to P1.

Note: If you straighten the pushbutton pins you won't need to hang it over the edge of the breadboard.

Program It

The wiring has a pushbutton connected to a pull-up resistor, and the junction between the resistor and the switch connected to P1 on the BASIC Stamp. When the pushbutton is not pressed, the voltage at the BASIC Stamp pin is 5 volts (=Vdd) through the pull-up resistor. But when the button is pressed, the voltage at the BASIC Stamp pin is low, zero volts (=Vss). Try the following test program.

```
' Earth Measurements program 2.1
' test pushbutton
loop:
debug bin in1
goto loop
```

What's all this DEBUG stuff?

In these experiments, you will be seeing the DEBUG statement very often, to put data on the computer screen. It's name comes from the notion of debugging. You can put information on the screen that helps you see what is going on in your program. Moreover, you can ask the DEBUG command to send any messages or data you want to the screen using a command called SEROUT. It does not have to have be especially for debugging.

The debug command lets you display the data on screen in quite a few different ways, using modifiers and screen control commands. In experiment 1, we used commands like this to display temperature data:

debug ? degC

That is a combination command that does 3 things: it prints the variable name and an equals sign; it prints the decimal value of the variable; and it moves the cursor down to a new line. The result looks something like this:

degC=25

The current little program has a different form of the debug statement:

debug bin in1

This prints the binary value of the variable in1. Yes, in1 is a variable, the state of input pin P1, either low or high, 0 or 1. This form of the debug command prints only the "0" or the "1", and not the name "in1", nor the "=", nor any spaces between the 1s and 0s, nor does it move down to a new line (until it hits the full width of the screen). The result looks something like this:

1111111100000000000111111111111111000000

00000011111111111111111111000000001111 1...

As we come to new forms, we will describe them briefly, and refer you to the BASIC Stamp manual, v1.9 pp 253-256.

Run it and observe the debug screen as you push and release the pushbutton. The program is going around and around the loop, spewing out the level that it finds at the input. The variable is `in1`. It is either high=1, or low=0. The reading should go to zero immediately when you press the pushbutton, and it should go to one when you release it. Yes? Go on to the next step. No? Is the problem in the program, the connection to the BASIC Stamp, or in the wiring of your pushbutton?

Now let's make the button produce a continuous sound while it is pressed down.

```
' Earth Measurements program 2.2
' buzzer
clik:            ' loop here while button is up
  if in1=1 then clik ' decide if it is up or down
  freqout 0,8,2500   ' play the tone while it is
'down
goto clik
```

Run this. When you press the button, you should hear a sound that may remind you of a cricket chirping. What is going on? If the button is up, nothing happens, because the `if` statement sees a 1 on the input pin and simply sends the program back to the top of the loop. If the button is down, the `if` statement sees a zero on the input pin. The program falls through and executes the `freqout` statement. Then it loops back to the top. So long as the button stays down, the loop with the `freqout` is executed over and over.

Recall that the parameter 8 in the `freqout` command is the duration of the tone in milliseconds. The tone is 2500 hertz, so in 8 milliseconds, there are 20 cycles of the tone (0.008 seconds * 2500 cycles per second = 20 cycles). Then the tone stops briefly, while the program goes back up to the top and tests the state of the P1 pin again. The tone is not produced during that time, because the BASIC Stamp can only execute one command at a time (This is an important fact to remember!), If the pin is still low, though, it soon is back to the `freqout` command.

So the sound looks something like this: ||||||||.||||||||.||||||||.||||||||.||||||||. What you hear is not the pure 2500 hertz tone, but a tone with repeated brief interruptions. These add the low sub-tone you hear in the sound, at about 110 hertz (about 9 milliseconds for the loop, 1/.009=111). This is indeed kind of like a cricket's stridulation (song), which is produced when the insect rubs a file on one of its forewings against a ridge on the other forewing, producing a high pitch, with brief pauses in the back and forth motion of the wings.

As a variation on the above program, try substituting the alternate values 1, 4, 50, 500 and 5000 for the duration parameter. Run the program each time and listen, and explain to yourself why it sounds as it does. At the long interval, 500 and especially 5000, note that the tone can go on long after the pushbutton is released. Why is that? Why doesn't the tone stop immediately when you release the pushbutton?

With the duration set back to 8, tap on the button to send the number "50" or "SOS" in Morse code. Refer to experiment 1 for the code listing. dit dit dit dit dit="5" and dah dah dah dah dah="0", dit dit dit="S", dah dah dah="O". It is already a useful program - a Morse code keyer!

Try inserting a `pause 6` command on the line after the `freqout` command. That gives a ||||||||.....||||||||.....||||||||.....|||||||| pattern that may seem even more cricket-like. Crickets, in addition to their "output transducer", the wings, also have an "input transducer", an ear. It is a membrane located on their front legs! Crickets are very sensitive to repeating patterns and pulses of sounds. It is their "Morse code". Their songs are part of their courtship and male rivalry. Entomologists have studied insect stridulations by reproducing sounds on speakers, and watching what parameters of the sound evoke what behaviors from the crickets.

Sometimes you don't want an action to keep going all the time the button is down. You want it to happen once and only once each time the button is pressed. Modify the program so that is reads as follows. (Here is a new convention to make your task easier - modified lines will be remarked with a ▲, and new lines will be remarked with a ❒. Other lines stay as they are.)

```
' Earth Measurements program 2.3
' single click on pushbutton, action on button down
clik:                       ' loop here while the button is up
  if in1=1 then clik        ' decide if the pushbutton is up(1) or down(0)
freqout 0,100,3800          ' ▲ play the tone once on buttonDown
clik1:                      ' ❒ loop here until buttonUp
  if in1=0 then clik1       ' ❒ decide if pushbutton is down(0) or up(1)
goto clik
```

As in the previous program, nothing happens until the button is pressed down. Then the tone plays for 100 milliseconds. Then there is a second holding loop, where the program stays looping until the pushbutton is released. When that occurs the program goes back up to the top, ready for the button to be pressed again. One press, one action.

That is fine, but think about how a mouse click usually works. Most mouse clicks do not perform their action until you release the mouse button. That's easy. Move the `freqout` down after the `clik1` loop:

```
' Earth Measurements program 2.4
' single click on pushbutton, action on button up
clik:                    ' loop here while the button is up
  if in1=1 then clik     ' decide if it is up(1) or down(0)
clik1:                   ' loop here until pushbutton goes back up
  if in1=0 then clik1    ' decide if it is down(0) or up(1)
  freqout 0,50,1900      ' ▲ play tone once on "buttonUp"
  freqout 0,100,3800     ' ▲ and while we're at it, a better sound!
goto clik                ' wait for next keypress.
```

Now a rising note should occur when the button is released. Logical, right? Be sure you understand totally how this works.

Now let's make the button take one action if you click it, and a different action if you hold it down for a long time. This is similar to the action of some computer menus that only appear if you hold the mouse button down for a longer period of time. Or you may have seen this in a car radio, where you press a preset button briefly to select a station, but you hold the button down for a longer time (until you hear a beep), to program a station you want into the preset memory. Appliances from wristwatches to VCRs, and yes, instruments sold in catalogs that cater to earth scientists, all of them use tricks like this to get to the configuration menus.

The program needs a variable to keep track of the time you hold down the button. Try this: (New code is shown with a ❒)

```
' Earth Measurements program 2.5
' pushbutton, action on click and hold
n  var  word                    ' ❒ variable to keep track of time
clik:                           ' loop here while button is up
  if in1=1 then clik            ' decide if it is up(1) or down(0)
  n=0                           ' ❒ zero the timer
clik1:                          ' loop here while button is down
  n=n+1                         ' ❒ increment the timer
  if n>500 then longclik        ' ❒ branch out after a certain time
  if in1=0 then clik1           ' or loop until buttonUp
  freqout 0,50,1900                   ' play tone once on buttonUp
  freqout 0,100,3800                  '
goto clik                       ' back to the top

longclik:                       ' ❒ come here if pushbutton is held down.
  freqout 0,5,3800,2533         ' ❒ sound to show the time has passed
longclik1:                      ' ❒ loop here while button is down
```

```
  if in1=0 then longclik1          ' ❒ decide if it is up(1) or down(0)
goto clik                          ' ❒ back to the top
```

The program arrives at `clik1` when you press the button. While the button is down, the program goes around and around the `clik1` loop. The statement with the `in1=0` keeps the loop going repeatedly back to `clik1` so long as the pushbutton remains down. Each time around the loop, the timer variable n increases by one. It is a race to see which happens first. Do you release the button first, or does the timer reach 500 first? If the button is released first, well, that is just a single click, as above. The program plays the tone and goes back up to the top to await another button action. But if the timer n reaches 500 before you release the button, the program branches to the `longclik` routine. There it plays one short chirp, to let you know that you've gotten there, and then waits for you to release the button. And then it goes back to the top.

Where does the magic number 500 come from? The simple answer is "trial and error". The programmer (you!) tries different numbers until it feels right. Approximately how long (in milliseconds) do you have to hold the button down before it branches to the `longclik` routine? Try experimenting - substitute different values in place of 500.

Think about the order of these two statements in program 2.5:

```
    if n>500 then longclik         ' ❒ branch out after a long time
    if in1=0 then clik1            ' loop until the button goes back up
```

What would happen if the order of the two statements were switched? If you aren't sure, try it.

Advanced Topic: Detecting a Double-click with the BASIC Stamp

Can the BASIC Stamp detect a double click? Sure, it's not too hard. At the end of a single click, the program has to wait a fraction of a second to see if you are going to press the button again. If you do, then it is a double click. If you don't, it is a single click. The interval of time is so short that you don't really notice it. The actual interval is determined by trial and error, a "user preference".

This too needs a timer variable. We will recycle the same timer variable, `n`, from the last routine. Try this: (lines remarked with a ❒ are the ones you need to add). Just for fun, we also modified the `longclik` routine too, to so that it plays a constant chirp that continues until the button is released.

```
' Earth Measurements program 2.6
' pushbutton, action on double click
n  var  word                    ' variable to keep track of time
clik:                           ' loop here while the button is up
  if in1=1 then clik            ' decide if it is up(1) or down(0)
  n=0                           ' zero the timer
clik1:                          ' loop here while the button is down
  n=n+1                         ' increment the timer
  if n=500 then longclik        ' branch out if it reaches 500
  if in1=0 then clik1           ' or loop until the button goes back up(1)
  n=0                                ' ❒ rezero the timer
clik2:                          ' ❒ loop here until time runs out
  n=n+1                         ' ❒ increment the timer
  if in1=0 then doubleclik      ' ❒ branch out if the button goes down soon
  if n<150 then clik2           ' ❒ loop while timer is less than 150
  freqout 0,50,1900             ' here for single click-play tone
  freqout 0,100,3800            '
goto clik                       ' back for next keypress
end

doubleclik:                     ' ❒ button is down for second click
    if in1=0 then doubleclik    ' ❒ loop until the button goes back up
    freqout 0,50,3800           ' ❒ play a unique falling sound
    freqout 0,50,2533
    freqout 0,50,1900
goto clik                       ' ❒ back to the top

longclik:                       ' come here if pushbutton is held down.
  freqout 0,5,3800,2533         ' ❒ play a chirp in a loop
  if in1=0 then longclik        ' ❒ loop here until the button goes back up
goto clik                       ' back to the top
```

If you press the pushbutton once and quickly release it, the program arrives at `clik2`. Now there is another race between the button and the timer. This time the button is up to begin with. If you quickly press the pushbutton a second time before the timer reaches 150, that means you intend a double click. But if the timer reaches 150 first, that means you just want a single click (or you have slow fingers and need to reset the preference to a longer time, say 200).

The `clik1` and `clik2` routines are similar, but observe that they are not identical. What would happen if the order of these two statements in the program were switched? If it is not obvious, try it, and think it through.

```
if in1=0 then doubleclik      ' ❒ branch out if the button goes down soon
if n<150 then clik2           ' ❒ loop while\ timer is less than 150
```

Snippet:

You can "snip" an action from one program, and use it (with changes?) in another. Pieces of programs that perform specific actions are called snippets. Snippets often do not stand on their own as complete programs. Programmers often exchange ideas in the form of snippets.

If you want, you could extend this logic to make a routine respond to a triple click, like some word processing programs use to select an entire paragraph. We'll let that be a challenge!

Now, let's move on.

Please save this program you have just built on disk. You can experiment with it later on in the challenges. We will be using **snippets** of what you learned here in programs to come. Use the name "cliks.bs2", or a name suggested by your instructor.

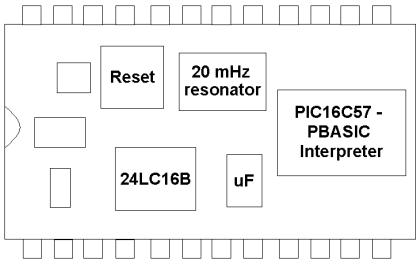
Learning to READ and WRITE, The Basics.

In this series of experiments, we are going to program the BASIC Stamp to collect readings of temperature and other variables. We want to log them, that is, collect them at regular intervals of time and store them in a file, and read them out later for comparisons, charts and graphs. We'll take this a step at a time. First, it is important to understand how the memory on the BASIC Stamp is organized.

You know from "What's a Microcontroller" that the memory available in the BASIC Stamp II is of two kinds, RAM and EEPROM.

It may help you to think about these kinds of memory if you know where they are located physically. Take a look at Figure 2.3, which shows a top view of the BASIC Stamp II.

Figure 2.3: BASIC Stamp Memory
The PIC16C57 chip is the BASIC Stamp's RAM and central processor. The 24LC16B is the EEPROM, which holds your PBASIC program and data we will be storing.



Variables are created in the RAM (Random Access Memory). You store numbers in RAM with statements like this, using named variables:

```
x       var     byte
x=76
```

Variables are very versatile. They can be added and subtracted and used in lots of other kinds of arithmetic, and they can be parameters in all sorts of commands that are described in the BASIC Stamp Manual Version 1.9. It is very fast to manipulate data in RAM (~200 microseconds per operation), and RAM does not wear out with use. Trouble is, there isn't very much RAM available on the BASIC Stamp, only 26 bytes. It is not suitable for storing lots of data. Also, the contents of RAM are lost when the BASIC Stamp loses power, or when the reset button is pressed. RAM is not suitable for storing "valuable" data that you want to survive when the power is disconnected.

Then there is EEPROM. A greater amount of EEPROM memory is available on the BASIC Stamp, 2048 bytes. Although part of the EEEPROM is used for your PBASIC program code, there will be some left over for data storage. One great advantage of EEPROM is that it is semi-permanent. The EEPROM memory retains its contents with or without power and through resets. The disadvantages of EEPROM are that it is relatively slow (~10 milliseconds to save a byte of data), and, it will wear out after something like 1,000,000 changes at one spot. To put this in perspective, if one certain location in EEPROM is reprogrammed over and over, once per second, it would take you about 11 days to get near the 1,000,000 mark. How many seconds are there in 11 days? On the other hand, at once per hour, it would take 114 years to reach that same mark. (How many hours are there in 114 years?) It is something to think about in planning. In Earth Measurements we may write to a single location a hundred times at most, nowhere near 1,000,000.

A final disadvantage of EEPROM is that there are only two instructions that can manipulate the data stored there. `Read` retrieves a byte, and `write` stores a byte. That's it. You cannot do arithmetic directly on the data stored in EEPROM, nor use it as directly as a parameter in a command. You first have to `read` it into a variable in RAM, and then manipulate it. And when you are ready, you can `write` the value of a variable into EEPROM. With that in mind, the main reason we use EEPROM is to store larger quantities of data, (if we won't have to change them too often) and where they will stay permanently if we do change them.

In PBASIC, the `data` statement reserves an area in the EEPROM, and gives it a name:

```
log     data  7
               ^-----the value 7 is loaded into EEPROM at address, "log"
 ^^^-------------------the name for the address in EEPROM where the data is located.
```

`Read` retrieves a byte from an address (in EEPROM) and copies its value to a variable (in RAM). The value of the byte in EEPROM is not affected by reading it.

```
read    log, x
             ^-----RAM variable to receive the data
          ^^^---------where in EEPROM to get the data
```

`Write` may be used in a program to change the byte stored at an address in EEPROM.

```
write   log, 25
             ^^----byte size constant
          ^^^---------where in EEPROM to put it
```

or, with a variable,

```
write   log, x
             ^-----RAM variable
          ^^^---------where in EEPROM to put it
```

Data in EEPROM is stored as bytes only. (Advanced topic: The RAM variable in the above statements can be a word, byte, a nib or a bit, but extra bits are padded or dropped on the left end if necessary to fit into a byte-size EEPROM cell.)

Do not confuse the address, "log" in this case, with the data that is stored there. Try the following program:

```
' Earth Measurements program 2.7
' distinction of constant, data and variable
dit     con     70                      ' define a constant
log     data    7                       ' reserve a byte in eeprom, initially 7
worm    data    240                     ' reserve a byte in eeprom, initially 240
x       var     byte                    ' define two variables
y       var     byte
read log,x                              ' read data from eeprom into the variables
read worm,y
debug ? dit, ? log, ? x, ? worm, ? y    ' show all quantities
```

The value of `dit` is 70, an ordinary constant. The name `dit` refers to the value itself. The values of `log` and `worm` are constants too, but they have values of 0 and 1, not 7 and 240. The names `log` and `worm` refer indirectly to the data. To read the 7 and the 240, there are two read commands in the program. One read gets the 7 from EEPROM address `log=0` and puts it in the RAM variable `x`, and the second read gets the 240 from EEPROM address `worm=1` and puts it in RAM variable `y`. The labels `log` and `worm` have the addresses 0 and 1 because PBASIC assigns addresses for data statements starting at 0.

Now modify the above program by adding four more lines at the end.

```
' Earth Measurements program 2.8
' writing a variable
dit     con     70
log     data    7
worm    data    240
x       var     byte
y       var     byte
read log,x
read worm,y
debug ? dit, ? log, ? x, ? worm, ? y
x=x+1                                   ' ❒ make a new value for x
```

```
y=y/2                                   ' ❒ make a new value for y
write log,x                             ' ❒ change the value stored at log
write worm,y                            ' ❒ change the value stored at worm
```

Run this and press RESET on the Board of Education a couple of times with the debug screen active. You should see the values of x increase by 1 each time, and the value of y halved each time. Then disconnect the power momentarily, and reconnect it. The first value you see on the debug screen should be the next one in the series, showing that the EEPROM retains its data when the power is off. What happened to the 7 and the 240 that were there when you first ran the program? They are gone. The `write` statement changed those values. The only way to restore the initial condition is to RUN the program again from the PC. Try it.

There is additional information about the `data` directive on pages 228-230 of the BASIC Stamp Manual Version 1.9, and also discussion of `read` (p. 302) and `write` (p. 341). You will be seeing more of this in experiments to come!

The EEPROM is often used to store settings and calibration constants that may need to be changed occasionally. It might be a parameter that tells how hot the temperature has to be before turning on a fan, or how many seconds have to pass before recording data in a log file. Here is a fun demo program that plays a musical scale when you single-click the button. How many notes it plays depends on a parameter that is stored in the EEPROM. If you hold down the button, instead of single-clicking it, the program enters a calibration routine where you will hear a series of ticks. Release the button after a few ticks, and then single click the button again.

```
' Earth Measurements program 2.9
' saving a setting in eeprom
dit           con    70                 ' length of a dit, milliseconds
how           data  1                   ' initial number of sounds
many          var     word              ' RAM variable for number of sounds
n             var     word              ' multipurpose variable
tone          var     word              ' the frequency of the sound

clik:                                   ' loop here while the button is up
  if in1=1 then clik                    ' decide if it is up(1) or down(0)
  n=500                                 ' initialize the timer for longclik
clik1:                                  ' loop here until buttonUp or timeout
  n=n-1                                 ' decrement the counter for long click
  if n=0 then longclik                  ' timeout!--goto longclik with n=0
  if in1=0 the clik1                    ' decide it the button is down(0) or up(1)
  tone=4519                             ' play tones, this is the first tone
  read how, many                        ' get how many to play from eeprom
  for n=1 to many                       ' and play them
  freqout 0,dit,tone                    ' sound, duration dit, frequency tone
  pause dit                             ' brief silence
```

```
  tone = tone**61858                    ' next note of chromatic scale
next                                    ' loop back to for if not done.
goto clik                               ' from the top, await keypress
end

longclik:                               ' enter here with n=0
  freqout 0,2,3800                      ' short tick
  pause 400                             ' short delay (time for response)
  n=n+1                                 ' increment n
  if in1=0 then longclik                ' loop until buttonUp
  write how,n                           ' store the new parameter
goto clik
end
```

Try to figure out how it works in detail. It consists of snippets from the foregoing button and memory routines. (The mathematical formula, tone = tone**61858, generates the chromatic scale, but you don't have to understand that here.) Do understand the role of `read` and `write`. There is one read command to fetch the number of notes to play, and one write command to store the new number selected by the user.

To test your understanding, modify the program as follows:

1. Add a data statement with a label of "`dur`" and make 70 milliseconds it's initial value.
2. Change "`dit`" from a constant to a byte variable.
3. At the outset of the program read the value from "`dur`" into the variable "`dit`". At this point, the program should run, just as it does now.
4. At the end of the `longclik` routine, before it goes back to "`clik`", have it wait for you to press and release the button a second time.
5. During this second time the button is down, have it increment the value of "`n`" each time around a loop.
6. When the button is released, write the value of n into the address "`dur`".
7. Verify that the program runs, and that the `longclik` routine allows you to change both the number of notes to be played, and also the duration of the notes.

Talking Thermometer, Morse Code Revisited

Now load in the program that you saved in Experiment 1. To do this, press ALT-L if you are using the DOS version of STAMP2.EXE, or press CTRL-O or use the mouse if you are using the Windows version, STAMPW.EXE.

The program from Experiment #1 reads the temperature from the DS1620 chip and displays it on the debug screen. After you load the program, run it to make sure that it still works. You never can be sure, maybe you accidentally bumped a wire on your Board of Education, or maybe someone was fooling around with your program on disk. It is a wise practice to start each step of building a complex system at a point where you know everything is working.

As it stands, the program displays the temperature on the debug screen once per second. Let's modify it, to make the piezo transducer send the temperature using Morse code. The Morse code in the first experiment of Earth Measurements was an introduction - it only sends the number 50. We need a subroutine that can sound out any arbitrary two-digit number we throw at it. And we'll change the program so that your new pushbutton will initiate the temperature reading. Starting with the program from Experiment #1, the new lines are remarked with a ❒, and the changed lines with a ▲.

```
' Earth Measurements program 2.10
' talking thermometer, using morse code.
dit     con   70                ' ❒ milliseconds for Morse dit
dit2    con   2*dit             ' ❒ constants related to dit
dah     con   3*dit             ' ❒ ditto
mc      var   byte              ' ❒ temporary for Morse pattern
xm      var   byte              ' ❒ morse input variable
j       var   nib               ' ❒ index for digits to send
i       var   nib               ' ❒ index for dits and dahs
x       var   byte              ' define a general purpose variable, byte
degC    var   byte              ' define a variable to hold degrees Celsius
                                ' note: DS1620 preprogrammed for mode 2.
                                ' high 13:shiftout 15,14,[12,2]:low 13

outs=%0000000000000000          ' define the initial state of all pins
     'fedcba9876543210
dirs=%1111111111111101          ' ❒ as low outputs
                                ' ❒ except P1, an input for a pushbutton

freqout 0,20,3800               ' beep to signal that it is running
high 13                         ' select the DS1620
shiftout 15,14,lsbfirst,[238]   ' send the "start conversion" command
low 13                          ' finish the command
clik:                           ' ❒ loop here while the button is up
  if in1=1 then clik            ' ❒ decide if the button is up(1) or down(0)
clik1:                          ' ❒ loop here while the button is down
  if in1=0 then clik1           ' ❒ decide if the button is down(0) or up(1)
high 13                         ' select the DS1620
  shiftout 15,14,lsbfirst,[170] ' send the "get data" command
  shiftin 15,14,lsbpre,[x]      ' get the data
```

```
low 13                          ' end the command
degC=x/2                         ' convert the data to degrees C
debug ? degC                     ' show the temperature on the debug screen
xm=degC                          ' ❒ morse routine expects data in variable xm
gosub morse                      ' ❒ to the subroutine
goto clik                        ' ❒back to wait for button again

morse:                           ' ❒ enter here to send byte xm as morse code
  for j=1 to 0                   ' ❒ send 2 digits, tens then ones.
    mc = xm dig j                ' ❒ pick off the (j+1)th digit
    mc = %11110000011111 >> mc   ' ❒ set up pattern for morse code
    for i=4 to 0                 ' ❒ 5 dits and dahs
      freqout 0,dit2*mc.bit0(i)+dit,1900      ' ❒ send pattern from bits of mc
      pause dit                  ' ❒ short silence
    next                         ' ❒ next i,dit or dah of five
    pause dah                    ' ❒ interdigit silence
  next                           ' ❒ next j,digit of two
return                           ' ❒ back to main
end
```

Run the program and try it by clicking the button. Listen to the Morse code as you make the temperatures go up and down. If you are not a ham or Navy radio operator, you may need a little practice to hear the numbers of the Morse code. But it shouldn't take long. You can read them on the screen as you listen. You can heat up the DS1620 temperature sensor with you finger, or by placing it under a lamp or in the stream of hot air from a hair dryer.

This talking thermometer is a useful instrument already. A visually impaired person could use it. Or how about a biologist doing research on bats in a dark cave? (Listening on an earphone-bats are very sensitive to high-frequency sound.) Can you think of other situations where this device might be useful?

Please save this program as it stands now on disk. Name of program? (degCtalk.bs2)

Now let's look at the program step by step. (The remainder of this experiment will be detailed explanation of the Morse code routine - no more programs for you until the challenges!)

Several variables and constants are defined at the top of the program. Some of these you will recognize from Experiment #1, where they appeared in the routine to send the number 50 as Morse code. There is the basic length of the dit in milliseconds, and the dah, which is defined as three times the length of the dit, and a new one, dit2, which is defined as twice the length of the dit. There are a couple of other variables, too, `xm` and `mc`, that we'll talk about in connection with the Morse code routine below.

P1 is now an input, for the pushbutton. P1 is set to input by making its bit in `dirs` equal to zero. The following statements fix the input and output state of all 16 pins of the BASIC Stamp.

```
outs=%0000000000000000          ' define the initial state of all pins
     'fedcba9876543210
dirs=%1111111111111101          ' as low outputs
                     ^----------' this is now an input for the pushbutton
```

Note the single change from the original program. If we do not set that bit in `dirs` equal to zero, then the program cannot read the pushbutton. If you don't believe it, try it and see what happens. You may wonder about the programs in the first part of this experiment, where we were reading the state of the pushbutton very well with neither a `dirs` nor an `outs` command. The reason is that the BASIC Stamp always starts up with all its pins as inputs. As a matter of good programming, we are turning them all into outputs, except the ones we truly need to be inputs. When we make a pin like P1 into an input, it doesn't matter what the state of the corresponding `outs` bit is. The outs bit has no effect when the pin is defined as an input.

The central idea of the Morse subprogram is held in the binary pattern, %11110000011111. The % sign marks it as a binary number. This is the pattern of zeros and ones as they are actually stored in binary brain of the BASIC Stamp. This binary number does have a standard numerical value (=15391), but the numerical value is not important here. Quite often in computer science, you have to think of computer data as something other than a standard numerical value. Think of this as a pattern on an audio tape. If you put a playback head (by analogy) at the far left and play back 5 bits moving to the right, you come up with 11110. This is going to translate in Morse code to dah dah dah dah dit, a nine. (It is not a binary number nine, which would be 1001 – instead, it is a pattern for Morse code number 9 – there are many ways to represent numbers!) Depending on where you start on the "tape" different code patterns result, in fact, the total pattern is arranged to give the code patterns for the Morse code numerals numbers in order. It's a trick.

```
11110000011111
^^^^^-------------> 11110, dah dah dah dah dit  nine, playing back five bits
 ^^^^^------------> 11100, dah dah dah dit dit  eight
  ^^^^^-----------> 11000, dah dah dit dit dit  seven
   ^^^^^----------> 10000, dah dit dit dit dit  six
    ^^^^^---------> 00000, dit dit dit dit dit  five
     ^^^^^--------> 00001, dit dit dit dit dah  four
      ^^^^^-------> 00011, dit dit dit dah dah  three
       ^^^^^------> 00111, dit dit dah dah dah  two
        ^^^^^-----> 01111, dit dah dah dah dah  one
         ^^^^^----> 11111, dah dah dah dah dah  zero
```

Now let's take the Morse code routine step by step, and really dissect it. First, you have to recognize that this is a subroutine, that starts with the label, "`morse:`", and ends with the "`return`" instruction. The main routine, after it acquires the temperature reading in degC from the DS1620 sensor, and puts it in the variable xm; it executes a `gosub morse` command. The `morse` subroutine does its thing and then returns execution to the instruction just after the `gosub morse` instruction, which is "`goto clik`". By writing the `morse` routine as a subroutine, we will be able to use it over again at different points in a our program, as it develops.

Variable `xm` is the one that will be sounded out as Morse code. In the `morse` subroutine itself there are two `for . . . next` loops, one inside the other. The outside loop has an **index** `j`:

```
  for j=1 to 0                          ' ❒ send 2 digits, msd 1st.
    mc = xm dig j                       ' pick off the jth digit
    mc = %11110000011111 >> mc          ' ❒ set up pattern for morse code
                                        ' more morse here
  next                                  ' ❒ next digit of two
return                                  ' ❒
```

What's a...

Index and Pointer:

An index is a variable that steps through a sequence of values. For example, "j" in the for-next loop steps through the values of 1 and 0. A pointer is a variable that specifies where in memory, or where in some ordered set, to retrieve information. For example, the variable "j" is both an index and a pointer. It points to a digit in the variable xm. The index "I" in this same program is a pointer to the bits (binary digits) of the variable mc. In experiments to come, we will use indices and pointers to refer to the data in the EEPROM log, as in, 1st reading, 2nd reading, and so on.

When the program first arrives at the `morse` routine, it first sets `j` equal to 1, and then continues with j=1 all the way through the loop, including everything in "more Morse stuff". The keyword, next, is the turning point in the for-next loop, and at that point the program jumps back up to the corresponding for, sets j=0, and executes all the way through again, to the next. Note that the BASIC Stamp knows how to count backwards! After j has taken on the values 1 and 0, that's it, the loop ends, and the program returns to the main program, and back to `clik`.

There are two math statements in this outer loop. The first one is:

```
mc = xm dig j.
```

This "`dig`" is an operator, kind of like "plus" or "divided by". It sits between two numbers, `xm` and `j`, and returns the (j+1)th digit of `xm`. It is easiest to illustrate with a specific example. Suppose the value of xm=25. On the first time through the loop, the value of j is 1, and the result of (`mc = 25 dig 1`) will be (`mc=2`), because 2 is the tens digit of 25. On the second time through the loop, the result of (`mc = 25 dig 0`) will be `mc=5`, because 5 is the ones digit of 25.

```
                          25
 j=1 point to the tens digit ----^
 j=0 point to the ones digit -----^
```

The logic of this can be extended to larger numbers, for example, j=3 would point to the thousands digit. However, in this Morse code routine we will only need 2 digits.

Now we have a number between 0 and 9 inclusive in the variable mc. The next statement sets up the pattern for the Morse code.

```
mc = %11110000011111 >> mc  ' ❒ set up pattern for morse code
```

The symbol >> is another operator that goes between two numbers. The constant, %11110000011111, is the binary pattern we were talking about above. The >> operator is one that operates specifically on binary patterns. It is called a shift operator. (Shifts are very important in computer science.) It shifts the binary pattern to the right a certain number of places (mc places) and drops that same number of bits off the right end. Again, to illustrate, the first time through the loop, the digit is 2 when the program arrives at this command:

```
BEFORE    mc=  11110000011111 >> 2
AFTER          111100000111         ' pattern shifted two to the right
                        \11         ' two bits dropped
                      ^^^^^--------- 5 bits are the morse pattern for "2"
```

And the second time through the loop, the digit is 5:

```
BEFORE    mc=  11110000011111 >> 5
AFTER          111100000           ' pattern shifted five to the right
                         \11111    ' five bits dropped
                      ^^^^^--------- 5 bits are the morse pattern for "5"
```

What has happened is that the Morse code pattern has ended up in bits 4 to 0 of the variable mc. In the example, 00111 represents 2 in Morse code, and 00000 represents 5. Earlier we talked about moving a "playback head" over the "tape"; here we have moved the "tape" over the "playback head", ready to play back the five bits on the right.

Now the Morse code pattern is in position, and we come to the inner `for-next` loop:

```
for i=4 to 0                                  ' ❒ 5 dits and dahs
  freqout 0,dit2*mc.bit0(i)+dit,1900          ' ❒ send pattern from bits of md
  pause dit                                   ' ❒ short silence
next                                          ' ❒ next dit or dah of five
pause dah                                     ' ❒ interdigit silence
```

The index here is `i`, and it runs through 5 values, counting backwards from 4 to zero. The `freqout` command plays a dit or dah for each time around the inner loop. Between each sound, there is a short pause equal in width to a dit. After the 5 dits and dahs of one digit are played, there is a longer pause, equal in width to a dah, and then the program loops back to get the ones digit, and play it's five bits in the same way.

The `freqout` command is familiar, except here the duration is neither a constant nor a simple variable. It is an expression. PBASIC lets you do that. The expression is:

```
dit2*mc.bit0(i)+dit
 ^^^^^^^^^^-------this has a value of either 0 or 1.
```

Let's start out by stating that `mc.bit0(i)` is a variable that has a value of either zero or one. So the statement reduces with simple multiplication and addition to either,

```
   dit2 * 0 + dit  ==>  dit
or
   dit2 * 1 + dit  ==>  3*dit ==> dah
```

The `freqout` command plays a dit or a dah, depending on the value of the mystery variable.

So what exactly is `mc.bit0(i)`? One powerful feature of PBASIC is that it allows you easy access to individual bits in that byte. The byte, mc, has 8 bits. The notation, `mc.bit` is called a modifier of the byte variable `mc`. It is really just name for the least significant bit of that byte. The second bit is `mc.bit1`, and so on `mc.bit4`, is the 5th bit. It is simply a way of naming the bits, a syntax that is built into the PBASIC language.

There is still another way to refer to those same bits, using a variable as a **pointer** to bits in the byte. This notation is `md.bit0(i)`. For example, `md.bit0(4)` and `md.bit4` both refer to the same bit. Literally it means, "the fourth bit up from md.bit0". See the BASIC Stamp Manual, v1.9 pp. 221-224 for more explanation.

Here is the way it works:

```
00111    <-- these are the five lower bits of the byte variable mc
    ^---- mc.bit0   or   mc.bit0(0)   different names for the same bit
   ^----- mc.bit1   or   mc.bit0(1)
  ^------ mc.bit2   or   mc.bit0(2)
 ^------- mc.bit3   or   mc.bit0(3)
^-------- mc.bit4   or   mc.bit0(4)
```

The variable `i` is the pointer. The power of this indirect, or array naming, is that the program loop (for i=4 to 0) can step through the bits of the byte variable, `mc`, one by one, and pick off the binary 0 or 1 values of the individual bits. Those are the bits that need to be sounded out as 0=>dit and 1=>dah. Here is another way we could have played the five dits and dahs, without using a for-next loop:

```
freqout 0,dit2*mc.bit0+dit,1900   ' ❒ first bit
pause dit                         ' ❒ short silence
freqout 0,dit2*mc.bit1+dit,1900   ' ❒ second bit
pause dit                         ' ❒ short silence
freqout 0,dit2*mc.bit2+dit,1900   ' ❒ third bit
pause dit                         ' ❒ short silence
freqout 0,dit2*mc.bit3+dit,1900   ' ❒ fourth bit
pause dit                         ' ❒ short silence
freqout 0,dit2*mc.bit4+dit,1900   ' ❒ fifth bit
pause dit                         ' ❒ short silence
```

You see, this refers directly to each bit, one at a time. But it comes out much shorter, and more elegant (?) using the for-next loop and the index as a pointer to the bits.

Whew! That was a lot of explanation for a short stretch of program. But it contains some advanced ideas. How to interpret a number as a pattern. **Index** and **pointer**. How to extract decimal digits. The dig and shift operators, how to use an expression as a parameter. How to use array modifiers of PBASIC variables. These are the stuff of programming a microcontroller.

Challenge!

Hook up an led to P5, so that `high 5` will turn it on. Write a program to turn the led ON when you click the button once, and OFF when you click the button again. (Push on, push off action). Hint: although there are several ways to do this, the `toggle` command may help. See page 329 of the BASIC Stamp Manual Version 1.9.

(A) Make a BS2 program that prints "working" on the debug screen, and plays a sound, once each time you click the button. Hint: print a message on the screen using commands like `debug "working",CR` `' CR` stands for "carriage return".

(B) Then program it so that if you hold the button down while you press and release RESET on the Board of Education, it will not go immediately to the "working" routine. Instead it will print "I await your instructions" on the debug screen, play a different sound, and delay until you click the button again. (Think about printers, how some will print a "test page" if you hold down some button on the front panel as you turn the printer on.)

The program 2.10 measures the temperature in degrees Celsius.

(A) Modify the program so that it displays degrees Fahrenheit, and plays it in Morse code.

(B) Modify the Morse code routine so that it will play three digits instead of just two, in case the Fahrenheit temperature goes above 99. (C-advanced) If you want to get fancy, make it so that it will not play leading zeros, that is, if the reading is 76 degrees F, it will play "7","6", not "0","7","6").

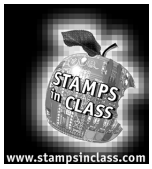
Then try this:

(A) Start with a byte of data, initially zero, stored in EEPROM. Each time the button is single-clicked, increment the byte in EEPROM by one (`read`, increment, `write`), and display the current value on the debug screen.

(B) When the value reaches 7, print the words "access denied" on the debug screen, and make a sound and blink the led on and off repeatedly. At that point, if you reset the stamp or remove the power and then restore it, the "alarm" should come on right away (`read` & decision at top of program.).

(C) (Advanced) Think of a way, using a special action on the button, like holding it down for a long time, to reset the value in EEPROM to zero. That will allow access so you can click the button 7 more times before the alarm re-sounds and locks you out.

Write a program that plays a unique sound if you triple click the pushbutton.



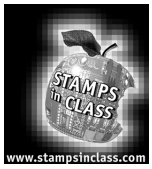
Experiment #3: Temperature Probe for Micro-Environments

The theme of Earth Measurements Experiment #3 is to connect a temperature probe mounted at the end of a cable that can reach out away from the Board of Education, to monitor microenvironments. A well-calibrated sensor, with good resolution, will achieve the most accurate results. The specific activities of this experiment consist of: (1) Capacitor on the BASIC Stamp input, and the `rctime` command; (2a) Temperature measurement using an AD592 probe, with calibration in an ice bath; and (2b) Comparison of calibration with the DS1620 at room temperature; (3) Automatic calibration using the BASIC Stamp's EEPROM; and (4) Talking (Morse code) temperature experiments--solar radiation--wet-bulb/dry bulb--wind chill.

Parts Required

For this experiment leave all parts from the previous experiments in place on the Board of Education. The following parts are required:

(1) AD592 temperature probe. Please see Appendix B if you would prefer to build your own instead of using the probe included with the Earth Measurements Parts Kit.
(1) nylon cable clamp or tie-wrap (to provide strain relief for the probe).
(1) 0.1µF monolithic capacitor
(2) 0.22µF film capacitor
(2) 100Ω resistor
(2) 100KΩ
(3) jumper wires
(1) conductivity probe (device with two 2" screws 0.4" apart mounted on a scrap PCB with wires)
(1) Ice bath. The best ice bath is made with crushed ice and water in a thermos bottle. If you don't have a thermos, use a couple of large nested foam cups, and wrap the whole thing outside with aluminum foil.



Build it!

Analog Temperature Sensor

It is often important to extend sensors out away from the recording instruments, so that measurements can be made in micro-environments. In the natural world there can be much variation from place to place and time to time. For example, the temperature of a leaf on a plant in the sun can be significantly different from the surrounding air temperature. The leaf forms its own micro-environment. And as plants grow, they create a unique micro-environment under their canopy. Often measurements are needed in several places at once and are fed back to one centrally located instrument. For example, an agricultural weather station will measure wind high above the ground, soil moisture underground, and other parameters at points in between. This means that sensors have to be mounted on cables to reach all those separate micro-environments.

In Experiment #1 you learned about the DS1620 smart temperature sensor. One nice thing about that sensor is that it returns readings directly as digital numbers. But one disadvantage it has is that it is a chip with 8 pins, hard to turn into a probe that can be used apart from a circuit board. In this experiment you will learn about a different kind of temperature sensor, the AD592. It is easy to incorporate into a probe mounted on a single pair of wires. This AD592 is an **analog temperature sensor**. Analog means that its signal is a continuous electrical value (microamps), proportional to temperature. Analog is the opposite of digital, which means that readings are returned as a code of discrete values (zeros and ones). The AD592 is a "classic" technology that has been proven through many years. Many of the signals you will encounter in the science of Earth Measurements, or in many fields of engineering for that matter, are analog signals. Chips like the DS1620 have analog sensors at their heart, and engineers have worked very hard to give the DS1620 its digital smarts.

What's a...

Analog temperature sensor:
The microamp current produced by the AD592 is what is called an "analog" of temperature. Microamps is not the same as temperature, just as apples are not the same as oranges. Considering an analogy between a capacitor and a water tank, here the analogy is between the temperature and electrical current. This is the basis of "analog" sensors. Other temperature transducers may transduce temperature to voltage or resistance or capacitance. The signals on both sides of the analogy are of a continuous nature, with infinite gradations of strength from low to high. Analog is the opposite of digital, where the signals are transmitted as digital codes of zeros and ones.

Analog sensors require a different kind of interface to the BASIC Stamp II. In this experiment you will learn about the `rctime` command. You may know about analog-to-digital converters, a kind of electronic chip that is dedicated to doing those conversions. The `rctime` command is a rudimentary analog-to-digital converter that is built into the BASIC Stamp II. To introduce the `rctime` command, you will connect a capacitor to the BASIC Stamp input and review the properties of capacitors. Once you have the `rctime` command reading the temperature sensor, you will learn how to calibrate it, so that it will read the correct temperature, despite variations in the parts that are provided to build the circuit.

Once you have the probe on a cable, you can extend it out to measure temperature in micro-environments in your surroundings.

BASIC Stamp Pins, Capacitors, Review of the BASICs

You probably already understand that the 16 general purpose I/O pins on the BASIC Stamp that can be in one of three distinct states at any given time. As shown in Figure 3.1, this is like three-position switch:

(1) The switch is connected to Vdd=+5 volts, as shown here, output HIGH. Current can flow out of the pin. (sourcing current from the +5 volt (Vdd) power supply).

(2) The switch is connected as an `input`. Zero current flows in or out of the pin. As an `input`, the internal BASIC Stamp circuitry monitors the voltage at the input pin. Less than 1.3 volts is seen as low (0). Greater than 1.3 volts is seen as high (1).

(3) The switch is connected to Vss=0 volts, output `low`. Current can flow into the pin (sinking current to ground - Vss).

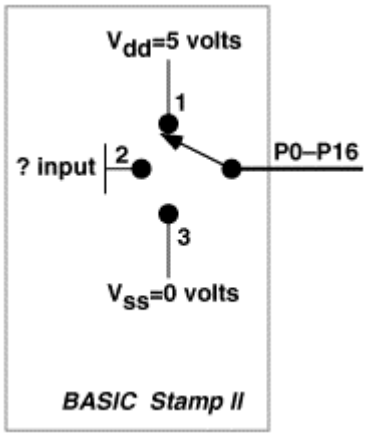


Figure 3.1: BASIC Stamp I/O Pins
Three positions exist in this switch: (1) Vdd +5V; (2) input to act as low or high; and (3) switch is connected to Vss +0V.

Simple commands like `high 5` or `low 5` or `input 5` put the named pin instantly into one of these three states. Many of the commands in the PBASIC language work by playing fancy games with the pins. For example, the `freqout` command makes a sound by flipping the internal switch rapidly between the `high` and `low` output states. The `shiftin` and `shiftout` commands work by coordinating the activity on several pins at once, some as outputs jumping from `high` to `low`, and others as inputs synchronized to the action. Here we will be introducing the `rctime` command, which works by switching a pin from output to input and timing how long it takes for the voltage at the pin to cross the 1.3-volt threshold level.

The change in voltage is brought about by external circuitry, usually a resistor (R) and a capacitor (C). The important point we want to emphasize here is that practically zero current flows when the pin is an input - it just looks.

First, a brief review of capacitors. Please bear with us if you already understand how capacitors work. The analogy is a tank of water, with an inlet pipe and an outlet pipe. The tank stores water, analogous to how the capacitor stores electrical charge. Figures 3.2 and 3.3 demonstrate this point.

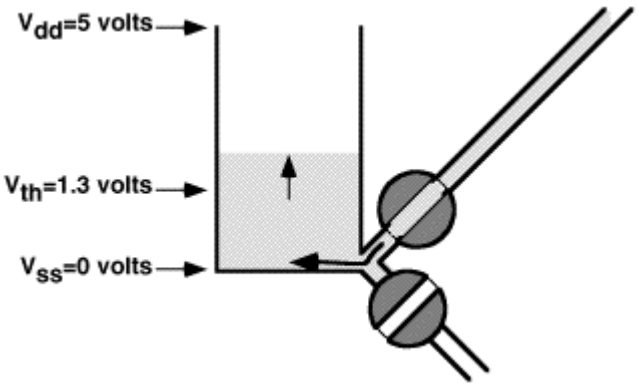


Figure 3.2: Analogy, capacitor charging
Water flows (amps) into the tank and the level (volts) rises. Higher inflow means higher rate of rise. The flow through the pipe can be limited by resistance (ohms) in the pipe, or by the water pressure at the other end of the pipe.

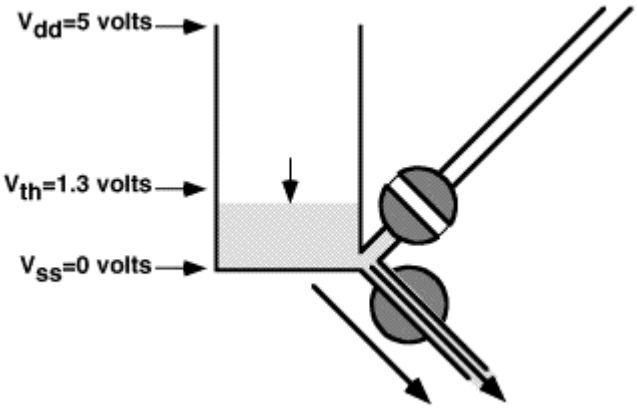


Figure 3.3: Analogy, capacitor discharging
Flow (amps) discharges from the tank and the level (volts) falls. The flow can be slow, a trickle, or fast, a deluge. If both inflow and outflow are zero, then the level stays constant. There can be unintentional flows, called leakage (amps).

Capacitors come in a wide range of sizes, measured in picofarads up to farads. This does not refer to physical size, but to the capacity to store charge, which depends on the material of which the capacitor is composed. Two capacitors of exactly the same physical size can have vastly different capacitances. We will be using values in this experiment that are 0.01 to 0.22 microfarads (uF).

Simple Resistance Detector

Hook up a 0.1 capacitor, a 100Ω resistor, and the conductivity sensor as pictured in Figure 3.4. The schematic is shown in Figure 3.5.

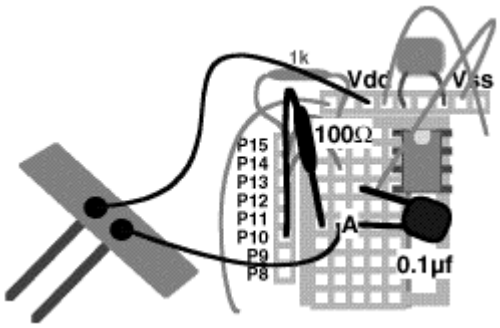


Figure 3.4: Capacitor and sensor on P10
The capacitor will probably be marked "104" or ".1" in tiny lettering. The orientation of this type of capacitor does not matter.

- 0.1 µf from pin 4 of DS1620 to node A
- 100Ω resistor from capacitor node A to P10
- conductivity sensor from node A to Vdd (+5 volts)

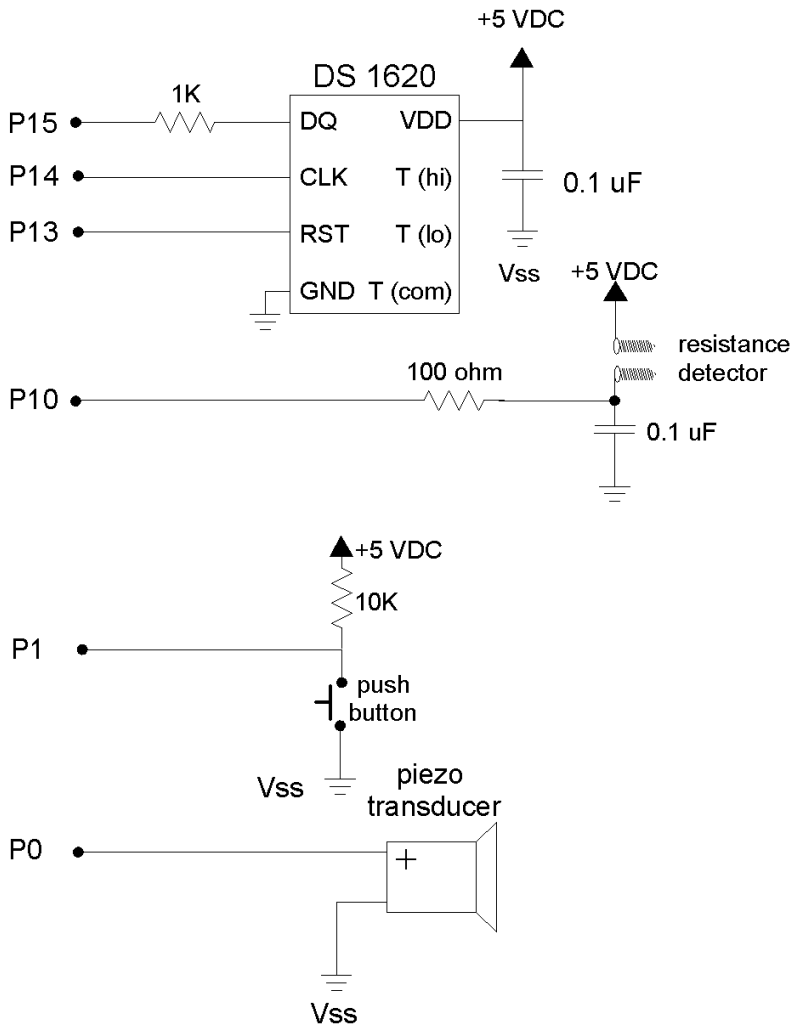


Figure 3.5: Simple Resistance Detector Schematic

Once you've got the circuit built, load the following program:

```
' Earth Measurements program 3.1
' simple demo of a capacitor on a BS2 pin.
  v  var bit              ' a bit-size variable to hold the input state
  loop1:                  ' come here to discharge the capacitor
    low 10                ' discharge the capacitor to 0 volts
    freqout 0,5,3500      ' signal event
    debug CR              ' new line on screen
    input 10              ' make the pin an input
  loop2:                  ' loop here while the input is <1.3 volts
    v=in10                ' read the input
    debug bin v           ' show it
    pause 99              ' 0.1 second pacing
  branch v,[loop2,loop1]  ' back to loop1 if voltage>1.3
```

The first instruction in the program discharges the capacitor to zero volts. The capacitor discharges very fast, like a big pipe dumping water from the tank out onto the earth. Current from the PIC microcontroller on the BASIC Stamp can discharge the capacitor through the 100 ohm resistor in about 25 microseconds, which is much less than the 10 millisecond duration of the `freqout` command. Then comes the `input 10` instruction. The pin instantly becomes an input. Let it sit for a minute or two without touching anything. Do you hear beeps or see any 1s on the screen? No? Don't be surprised if the capacitor stays discharged, because there is no source of current to charge it. All those zeros on the screen mean that the capacitor stays discharged.

Now touch the two leads of the conductivity sensor with your fingers. Experiment! The result should depend on how sweaty or wet your fingers are (a lie detector?), and how hard you pinch. There are leakage paths through the moisture on your fingers, and through your skin and tissue. Try dipping the conductivity sensor in water, or touch it to wet paper, or touch it to a heavy pencil line drawn on paper. You can also substitute 100K ohm resistor. Your finger short-circuits the capacitor as shown in Figure 3.6

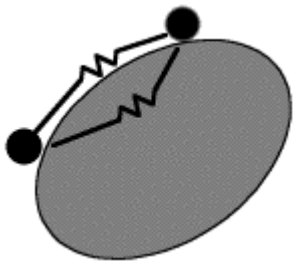
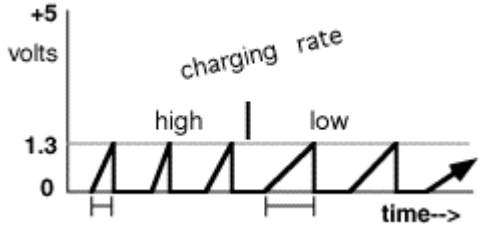


Figure 3.6: Short-Circuit
Touch the two screws that comprise the conductivity probe. Your finger will short-circuit the capacitor.

We need to say a word here about safety. The voltage and current in this circuit are very small, five volts and a few microamps. If you are ever unsure about a circuit, always err on the side of safety!

The input pin on the BASIC Stamp is acting as a "comparator". This is a technical term in electronics for this device that gives a yes or no answer, 1 or 0, to the question, "is the voltage level at P10 greater than or less than 1.3?" This 1.3-volt threshold is fixed by the PIC microcontroller in the BASIC Stamp II, and there is nothing we can do to change it. Figure 3.7 shows how this works.

Figure 3.7: Capacitor Discharge
Over and over, the capacitor is discharged to zero volts, and then more or less rapidly charges back up to the 1.3 volt threshold. By varying the resistance of the probe, you are affecting how fast the voltage level rises.



The program uses the `branch` instruction, which may be new to you. It instructs the BASIC Stamp to go to one of two possible destinations.

```
branch v,[loop2,loop1]  ’
                ^^^^^-----    go here if v=1 (capacitor voltage >1.3 volts)
           ^^^^------------    go here if v=0  (capacitor voltage <1.3 volts)
        ^------------------    branch on this variable
```

If the voltage level on the capacitor rises to 1.3 volts, then variable v will become equal to 1, the program will be directed back up to `loop1` to discharge the capacitor to zero volts, make a tone, and print a line return on screen. Otherwise, the program is directed to `loop2`, where it continues to read the input and print zeros on the screen. The variable `v` is a bit; either 0 or 1, so this `branch` instruction covers all the possibilities, `loop1` or `loop2`. Another way to write this would be,

```
if v=1 then loop1           ’ go back and discharge the capacitor
goto loop2                  ’ or just watch and wait
```

`Branch` is more concise, better programming when there is a clear set of choices. For more about the branch instruction, see the BASIC Stamp Manual Version 1.9, pg. 247.

Resistance Detector Using RCtime

Now type and run the following program:

```
' Earth Measurements program 3.2
' simple demo of the RCtime command.
rct  var word           ' a word variable
n    var byte           ' variable for the bar graph
low 10                  ' discharge the capacitor
loop1:                  ' loop here forever
  RCtime 10,0,rct       ' time for the volts to rise to 1.3
  low 10                ' discharge the capacitor to 0 volts
  debug ? rct           ' show the time
  n=(rct-1)/2048+1      ' calculate length of bar graph
  debug rep "*"\n,cr    ' display ascii art bar graph
goto loop1
```

Again experiment, with different wetness and pressure. What sorts of `rct` values do you observe?

Here are the parameters of the `RCtime` command:

```
RCtime 10,0,rct
           ^^^-----     variable to hold the result (In units of 2 microseconds)
         ^---------     command starts with in10=0, ends when in10=1
                        (alternatively, start with in10=1 and end with in10=0)
       ^^-----------    use pin 10 for this RCtime command
```

The `RCtime` command measures the time that it takes for the capacitor to charge up from zero to the 1.3 volt threshold. The program makes pin 10 low to start off, and that discharges capacitor to zero volts. The `RCtime` command then turns P10 into an input, and immediately starts looking for the voltage at the I/O pin to cross the 1.3 volt threshold, while at the same time it counts up the elapsed time. The `RCtime` command counts up in two microsecond intervals. If the voltage at the pin does cross the 1.3 volt threshold, then the `RCtime` command wraps up and puts the elapsed time into the variable `rct`, and the program continues with the instruction after the `RCtime`. Here that is a `low 10`, which discharges the capacitor back to zero. If the voltage at the pin does not cross the 1.3 volt threshold within a tenth of a second (0.13107 second, to be exact), the `RCtime` command gives up. It puts zero into the variable `rct` (to indicate overflow) and then the program continues with the next instruction after the `RCtime`.

`RCtime` counts in units of 2 microseconds, and the maximum value of the count is 65535 (The maximum value that will fit in a sixteen bit word.), so it follows that the maximum time is 131,070 microseconds (2* 65535)=0.13107 second. See the BASIC Stamp manual pg. 298 for more information about `RCtime`. To reiterate, if nothing happens within 0.13107 second, the `RCtime` puts zero in the variable `rct`, to indicate an overflow condition.

The RCtime command is useful for measuring many different things. Electrically, the circuit can be arranged that the time depends on resistance, capacitance, voltage or current. Many transducers output one of those electrical quantities. For example, the temperature sensor coming up is a transducer that transduces temperature into an electrical current. A simple formula will allow us to convert the value returned by rctime immediately into temperature. Another type of temperature sensor that is well suited for use with the rctime command is the thermistor. It has a resistance that varies with temperature. It is not so convenient, because it harder to calibrate.

Finally an explanation of the ASCII art bar graph in Program 3.2. This is a continuing education into the capabilities of the debug command. Before the advent of computer graphic displays and printers, these ASCII graphs were the only way to produce a graphical output!

```
n=(rct-1)/2048+1       ' calculate length of bar graph
debug rep "*"\n,cr     ' display ascii art bar graph
```

When rct has a value between 0 and 65535, the value computed for n will be between 1 and 32. Note that 65535/2048=31. That defines the maximum value, and lower values fall into place. We scale it to 32 maximum simply so that the graph will fit on the width of the STAMP2.EXE debug screen. Subtracting 1 from rct is a refinement. Recall that the RCtime command only waits around for 0.13107 second, and then returns rct=0 to show that the time was longer than that. If we just graph that, then the longest, overflow, times end up having the shortest length on the graph. By subtracting 1, rct=0 becomes (rct-1)=65535. (That is how unsigned binary 16 bit math works--zero minus one equals 65535). The graph makes more sense that way. The debug command then uses the "rep" modifier to print n stars on the screen, followed by a line return. See the BASIC Stamp Manual Version 1.9, page 256 for more information about the rep modifier of the debug command.

3

Temperature Sensor Probe Using the AD592 and RCtime

Now, remove the capacitor and conductivity sensor shown in Figure 3.4, and build the circuit shown in Figure 3.8. A schematic of the revised circuit is shown in Figure 3.9.

Figure 3.8: AD592 Temperature Sensor and RCTime
AD592 temperature sensor probe. The AD592 is mounted and insulated with heat-shrink tubing on the end of a 16" pair of wires. Please see Appendix B for construction details if you are making your own.

- AD592 probe red wire (+) to hole next to the pushbutton, node A.
- AD592 probe black wire (-) to the row one up from the red (+) wire, node B.
- other end of node A, wired to Vdd (+5 volts) node next to the piezo.
- other end of node B, wired through 100Ω resistor to P5.
- 0.22 uF monolithic capacitor (marked 224) from node B to 2nd row up, node C
- reroute Vss wire from switch (-) to node C, and a new wire from node C to Vss (0 volts).

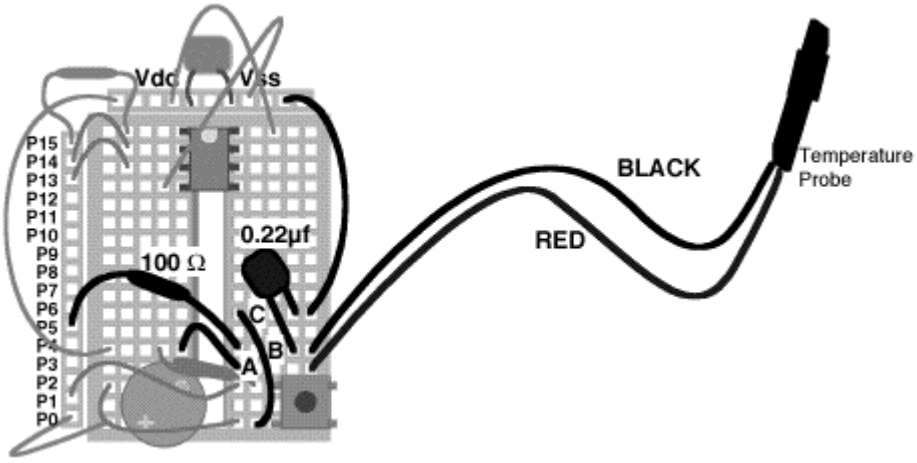
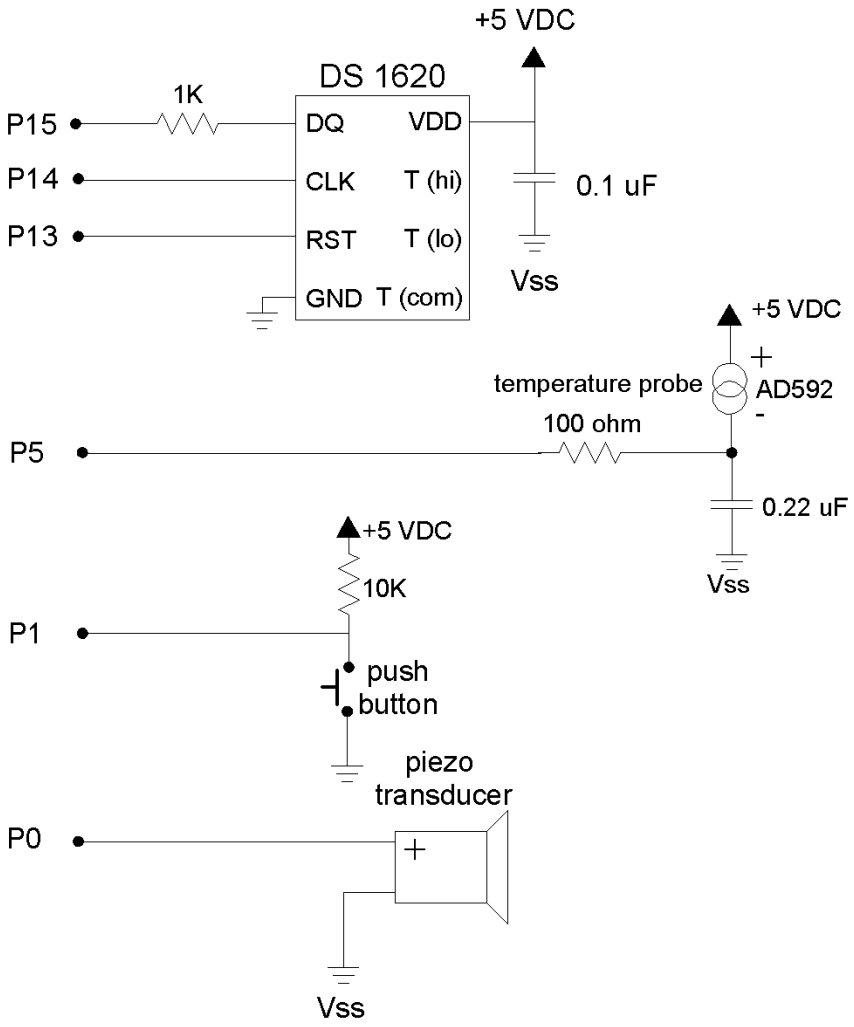


Figure 3.9: AD592 Temperature Sensor and RCTime Schematic



Then try the following program:

```
' Earth Measurements program 3.3
' Reading the AD592 temperature sensor using the RCtime command.
Kal   con 15300                              ' constant to be determined
rct   var word                               ' a word variable
TK    var word                               ' Kelvin temperature
TC    var word                               ' degrees Celsius
loop:
  low 5                                      ' discharge the capacitor
  RCtime 5,0,rct                             ' time for the volts to rise to 1.3
    TK = Kal/rct*10 + (Kal//rct*10/rct)      ' calculate Kelvin
  TC = TK-273                                ' and Celsius
  debug dec rct,tab,dec TK,tab,sdec TC,CR    ' show the results
goto loop
```

The display on the debug screen should show three columns, the raw time count (in units of two microseconds) from `RCtime`, and the calculated Kelvin and Celsius temperatures. Heat up the temperature probe in your hand or by some other means and verify that the `rct` reading on the debug screen goes down as the temperature goes up. The TK and TC readings should go up with temperature, but do not pay attention to the exact values yet. You still need to "calibrate" the sensor.

Note: This form of the `debug` command separates the decimal values of the variables with "tab" characters to put them in columns. The "`sdec`" modifier allows for the display of negative numbers.

A word about what sort of device the AD592 transducer is, electrically. It is a current source. The equation that governs its behavior is exceedingly simple:

Output = 1 microamp / Kelvin

That is, at 273 Kelvin (freezing, 0° C) , it produces 273 microamps. At 373 Kelvin (boiling, 100° C), it produces 373 microamps. At absolute zero it would produce zero microamps, although that is actually outside its operational limit of -40 degrees Celsius.

If you look at the AD592 in terms of the analogy with a water tank, it is like a flow regulator on the input pipe. The flow does not depend on the level in the tank, nor does it depend on the pressure (voltage) that supplies the current on the other side of the regulator. This is very different from a resistor or wet fingers, where the current depends on several factors. The name "`RCtime`" comes from R for resistance, C for capacitance, and the time it takes for a resistor to charge the capacitor. A current source is a very special kind of regulated resistor, one that makes the calculations relatively easy, lucky for us!

AD592 Calibration

The formula that relates the temperature to the time measured by RCtime is a reciprocal: in this case TK is Kelvin temperature.

rct = constant/TK or TK = constant/rct

See the box for the theory regarding the **rate of change of voltage on a capacitor**. The constant will be around 153000 when the capacitor is 0.22 uF. But it will not be exactly that value due to variations in the component values. That is why it needs to be calibrated.

Theory governing the rate of change of voltage on the capacitor:

The equation governing the rate of change of voltage on the capacitor is

dV/dt = I/C

where **I** is the current and **C** is the capacitance. If you know calculus, and assume that **I** and **C** are constant, you can easily solve for elapsed time in terms of the change in voltage and the capacitance and the current:

t = C * V / I

where **t** is in seconds, **C** is in Farads, **V** is in volts, and **I** is in amps. If we substitute TK in Kelvin for microamps, 0.22µf for C, 1.3 for the voltage, and 2*rct for the time in microseconds, and taking care for the units, we come up with the formula in the text:

rct = constant / TK

The constant is 153000, when those ideal values are plugged into the formula. In reality, the capacitor will not be exactly 0.22uF, the threshold will not be exactly 1.3 volts, and the AD592 will not have an output of exactly 1 microamp per Kelvin. Nevertheless, since there is only one free constant, we will need just one point of calibration.

In order to calibrate the sensor, we need to find the constant for this particular setup. To do this, the AD592 sensor must be put in a location where you know the temperature exactly. A good choice is an ice bath at 0° C, 273 K. With an ice bath reference, TK=273, the constant will be (rearranging the previous equation):

constant = 273 * rct

We have to put the probe into an ice bath, let it stabilize, read the value of rct, and multiply to find the constant.

Now try it yourself. Put your AD592 probe in an **ice bath**, and run the program so that you see the `RCtime` readings on the debug screen. Wait for the reading to equilibrate (become steady).

reading rct?_____

Take that number and multiply it times 273. This is your calibration constant.

constant =_____

Be aware that this constant is specific for this sensor, this BASIC Stamp, and this capacitor. Now round off the

constant to the nearest 10, and drop the final digit (a zero). This should be a five-digit number. This will be the value of Kal you need to substitute in the program.

Kal = _____

Put this value in your program 3.3 in place of the 15300 "default" value. When you rerun the program, you should see TK and TC show the temperature of the calibration bath--273 Kelvin, 0 Celsius.

What's a...

Ice bath, calibration standard:

The melting point of ice made with pure water is a physical constant. Zero degrees Celsius, 32 degrees Fahrenheit, 273 Kelvin (Or 273.14 if you want to push the precision). You can get the best results if the ice and water mixture is:

(1) made with crushed ice made from distilled water;
(2) is held in a vacuum thermos bottle with a narrow mouth;
(3) stirred gently while making the measurement; and
(4) at least 5 cm of wire is submersed above the sensor probe tip.

Lacking a thermos bottle, you can substitute a well-insulated foam container. Careful preparation is very important if you want to achieve good results in the calibration! Watch until the reading settles down to a steady value, to equilibrium.

Metrologists (not meteorologists!) are scientists who advance the science of accurate measurements. They have to think about all possible factors that could influence the measurements.

Now, to explain the peculiar formula needed to calculate TK. Unlike big computers, where the computer language has lots of fancy math available, you will need to stretch the BASIC Stamp's limited math brain. The reason we need the trick is that the constant, ~153000 (or whatever you found) is larger than the highest possible number that the BASIC Stamp can work with (2^{16}=65536).

Recall how you did long division in elementary school. This is the same thing, really, but the notation is a little different. Here are examples of the two essential elements in BASIC Stamp math:

BASIC Stamp Notation:	. . meaning
1432/524 = 2	Single slash means INTEGER DIVISION (524 goes twice into 1432) and there is a remainder.
1432//524=384	Double slash means remainder after INTEGER DIVISION: 1432-(2*524)=384. Remainder always less than divisor, 384<524.

Observe that 2*524+384=1432, that is, the quotient times the divisor, plus the remainder is equal to the original number. That is really the definition of division.

Now, think how you would solve the problem of 143220/524, using elementary school arithmetic:

Equation:	Steps in elementary school arithmetic:
27 524 ⟌ 143220 14148 174	First step in a long division. 524 goes 27 times into 14322, and the remainder is 174. The BS2 knows how to divide into numbers that have numerators less than 65536, so it has no trouble in figuring out that 14322/524=27 in one step.
273 524 ⟌ 143220 14148 1740 1572 168	Next step, the zero is brought down to the right of the 174 remainder. That effectively multiplies the 174 times 10. Then 1740 is divided by 524, and the result, 3 is put after the quotient, which becomes 273. This too effectively multiplies the 27 times 10, as it moves up by one significant figure. The fractional remainder, 168/524, is dropped.

Here is how the BASIC Stamp II denotes the same problem:

```
TK = 14322/524*10 + (14322//524*10/524)
    '^^^^^^^^^^^^-----------------    first step of division, times 10-> 270
                 '^^^^^^^^^^^^^^    the remainder times ten= 1740
                               ^^^division to get next quotient digit=3
    '^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^bottom line quotient is 273
                                    final remainder 168 is dropped
```

The result is that the BASIC Stamp has calculated 143220/524=273.

If you like math, great, you understand or can figure out how it works. If you are one of those who draws a blank when you see math, or you don't have the time to sit down and think it trough, well, just take the formula, use it, and turn the crank. A lot of math in computer science is like that. It comes in libraries you just use without thought when you write programs. You assume that the wizards in the tower have gotten it right. Nonetheless, understanding it can be helpful and rewarding, or a career, if you are good at it.

What is the smallest change in temperature that we can detect? Look at some typical `rct` values and `TK` values, if the constant is 143000.

Raw conversion:	real	Kelvin	Celsius
143000/484	=295.5	295	22
143000/485	=294.8	294	21
143000/486	=294.2	294	21
143000/487	=293.6	293	20
143000/488	=293.0	293	20
143000/488	=292.4	292	19

The real resolution is about 0.6 Kelvin. That is, each step in `rct` is at best a step of 0.6 Kelvin in temperature. We are rounding it off to 1 Kelvin. (Losing a little information.)

If we were to use a larger capacitor (say 0.33µF) in the circuit, the constant would be larger, and the **resolution** would be improved. On the other hand, with a smaller capacitor (like 0.1µF), the resolution would be worse.

In the next experiment, you will check on the calibration of the AD592 probe in comparison with the DS1620 from Experiments #1 and #2.

What's a...

Resolution:

Suppose you are measuring a quantity that can take on any value between zero and 100. If your instrument can only tell the difference between "greater than 50" and "less than 50", then it has one bit of resolution, that is, the measurement is a sort of "yes/no". On the other hand, if your instrument can tell the difference between 1 and 2 and 3 and so on up to 100, then the resolution is 1%, or about 7 bits (7 bits, because 2^7=128). Resolution is not the same thing as accuracy. If your instrument reads 50 when the true value is 52.3, then it is not accurate, or at least it needs to be calibrated. That is true whether it has one bit or 7 bits or more of resolution.

Talking Thermometer Revisited, Two Channels

Now let's combine this new sensor with the DS1620 talking thermometer. Load in program 2.10, saved from Experiment #2, and add the lines shown below with the ❒ symbol. In place of the `Kal` value shown here, use the one that you calculated above. See how the program is building up. You are simply adding the new variables and the new PBASIC routine for the AD592 to the existing program. Position the AD592 probe in contact with the DS1620 on the Board of Education while you type this in. The first thing you are going to test is to see if the AD592 probe and the DS1620 have the same reading at "room temperature", so you will want them to be at the same temperature.

```
' Earth Measurements program 3.4
' talking thermometer, two channels.
dit   con   70                ' milliseconds for Morse dit
dit2  con   2*dit             ' constants related to dit
dah   con   3*dit             ' ditto
mc    var   byte              ' temporary for Morse pattern
xm    var   byte              ' morse input variable
j     var   nib               ' index for digits to send
i     var   nib               ' index for dits and dahs
x     var   byte              ' define a general purpose variable, byte
degC  var   byte              ' define a variable to hold degrees Celsius
                              ' note: DS1520 preprogrammed for mode 2.
                              ' high 13:shiftout 15,14,[12,2]:low 13
Kal   con  15300              ' ❒ USE YOUR OWN calibration constant
TK    var  word               ' ❒ for Kelvin temperature from AD592
TC    var  word               ' ❒ Celsius of AD592
rct   var  word               ' ❒ for the RC timer.

outs=%0000000000000000        ' define the initial state of all pins
     'fedcba9876543210
dirs=%1111111111111101        ' as low outputs
                              ' except P1, an input for a pushbutton
freqout 0,20,3800             ' beep to signal that it is running
high 13                       ' select the DS1620
shiftout 15,14,lsbfirst,[238] ' send the "start conversion" command
low 13                        ' finish the command
clik:                         ' loop here while the button is up
  if in1=1 then clik          ' decide if the button is up(1) or down(0)
clik1:                        ' loop here while the button is down
  if in1=0 then clik1         ' decide if the button is down(0) or up(1)

DS1620:                       ' ❒ label for DS1620 temperature sensor code
  high 13                     ' select the DS1620
```

```
    shiftout 15,14,lsbfirst,[170]     ' send the "get data" command
    shiftin 15,14,lsbpre,[x]    ' get the data
  low 13                        ' end the command
  degC=x/2                      ' convert the data to degrees C
  debug ? degC                  ' show the temperature on the debug screen
  xm=degC                       ' morse routine expects data in variable, mx
  gosub morse                   ' to the subroutine
pause 100

AD592:                          ' ❒ label for AD592 temperature sensor code
  rctime 5,0,rct                ' ❒ get the AD592 count
  low 5                         ' ❒ pull input low, discharge cap
  TK = Kal/rct*10 + (Kal//rct*10/rct)
                                ' ❒ calculate Kelvin
  TC = TK-273                   ' ❒ and convert to degrees C
  debug ? TC                    ' ❒ show the result (if hooked to the PC)
  xm=TC                         ' ❒ morse routine expects data in variable, xm
  pause 500                     ' ❒ delay to separate the readings
  gosub morse                   ' ❒ to the subroutine

goto clik                       ' back to wait for button again

morse:                          ' enter here to send byte xm as morse code
  for j=1 to 0                  ' send 2 digits, tens then ones.
    mc = xm dig j               ' pick off the (j+1)th digit
    mc = %11110000011111 >> mc  ' set up pattern for morse code
    for i=4 to 0                ' 5 dits and dahs
      freqout 0,dit2*mc.bit0(i)+dit,1900    ' send pattern from bits of mc
      pause dit                 ' short silence
    next                        ' next i,dit or dah of five
    pause dah                   ' interdigit silence
  next                          ' next j,digit of two
return                          ' back to main
end
```

If the Morse code becomes obnoxious to you or your classmates, simply unplug the wire from P1 to turn off the buzzer, or put an apostrophe in front of the `gosub morse` and turn it into a comment.

Run this with the AD592 probe in direct contact with the DS1620 on the Board of Education (and no direct sunlight). If you have some, you can put some heat sink compound (thermally conductive grease) between the two to improve the contact. Be sure the readings are constant, and record the readings in degrees Celsius:

DS1620 :_____
AD592 :_____

They should be pretty close to one another. You just calibrated the AD592 in an ice bath, and the DS1620 data sheet specifies that its reading will be within ±0.5 degree.

Save this new version of the program on disk, following your teacher's instructions.

Automatic Calibration (Advanced Topic)

One feature of many modern instruments is automatic calibration. For example, since we know that the DS1620 has 0.5 degree accuracy, we might like to skip the preparation of an ice bath, which, after all, requires quite a few materials and effort to do it right. We can use the DS1620, at room temperature, as the calibration reference. You could attach a new AD592 temperature probe to the Board of Education, put it in contact with the DS1620, and let it sit for a few minutes, and then press a button to enter the calibration value. Viola! The BASIC stamp calculates the correct calibration value and stores it in EEPROM for you. Here is a program to do that. The program also instructs how to store and retrieve word-size data in the EEPROM.

In program 3.4, the calibration constant was entered as

```
Kal   con  15300      ' USE YOUR OWN calibration constant
```

but now you need to replace that with:

```
eKal  data word 15300 ' ❒ constant to be determined, automated
Kal   var  word       ' ❒ for calibration constant
```

The value `eKal` points to a value in the EEPROM, and that value will be transferred to and from the variable Kal, using the BASIC Stamp's `read` and `write` statements.

The calibration constant is a word value, but the EEPROM stores only bytes. The trick is to store the word in two successive bytes of EEPROM. Enter the following two lines in the program just before the `RCtime` instruction.

```
read eKal, Kal.byte0           ' ❒ get calibration constant low byte
read eKal+1, Kal.byte1         ' ❒ get calibration constant high byte
     ^^^^^^--------------      ' read from location eKal, then from e=Kal+1
             ^^^^^^^^^---      ' byte0, then byte1, of word variable Kal
```

The `Kal.byte0` is a modified form of the variable `Kal` that means, "byte 0 of word Kal". This is like the modifiers that you saw in the Morse code routine in Experiment #2.

Also modify the `clik1` routine so that if you hold the button down for a long time, it will branch to a special calibration routine. Recall the `longclik` routine from Experiment #2?

```
    x=0                              ' ❒ add this line to zero the counter
clik1:
    pause 100                        ' ❒ 0.1 second pacing
    x=x+1                            ' ❒ add this line to count up x
    if x>30 then calibrate           ' ❒ calibrate if button held > 3 seconds
  if in1=0 then clik1                ' loop until buttonUp
```

Finally, add the following subroutine at the end of the program:

```
calibrate:
  freqout 0,5,3400                   ' ❒ signal to show we got here
  debug "The probe should be in contact",CR
  debug " with the DS1620",CR
  TK=degC+273                        ' Kelvin temperature of DS1620
  Kal = TK/10*rct + (TK//10*rct+5/10) ' ❒ compute and round off Kal
  debug ? Kal                        ' ❒ show value of Kal
  write eKal,Kal.byte0               ' ❒ write the low byte of Kal
  write eKal+1,Kal.byte1             ' ❒ write the high byte of Kal
  freqout 0,5,1900                   ' ❒ show finished
goto clik                            ' ❒ back to the top
```

Try it out. When you first run the program, the temperatures returned from the AD592 will be incorrect, due to incorrect default value of `eKal`. Put the AD592 in contact with the DS1620. IMPORTANT: make good contact between the AD592 and the DS1620, using a wire to hold them together, and improve the contact with silicone heat sink grease if you have some. Be sure there are no nearby sources of heat.

Click the button and watch the reading until you see that it has settled down. When ready, press and hold the button until you hear the calibration click. When you release the button, the readings should suddenly become correct in comparison to the DS1620. The AD592 probe can now be extended out to measure other

temperatures. This kind of auto-calibration capability is especially important for instruments that read things like conductivity or pH (acidity), where the sensors need frequent recalibration.

Place your AD592 probe in the ice bath. It should read close to zero, within the ±1 degree resolution of the Board of Education measurement system. Remember, the calibration routine depends on having the AD592 at the same temperature as the DS1620! With this auto-calibration routine, do not press the calibration button when the probe is in the ice bath!

The calibration constant comes from the equation:

```
constant = (true Kelvin temperature) * rct
```

We are assuming that the DS1620 gives us the "true" temperature. Suppose the DS1620 is at 25 degrees Celsius, 298 Kelvin. Suppose the value of `rct` is 591. So,

```
constant = (298 * 591) = 176134,
```

. . .and the value that must go into the EEPROM is the top five digits of that, rounded off as before. The trick is to get this result on the BASIC Stamp without overflowing 16 bits. It takes two steps, rewriting the reference temperature as. . .

```
298  =  29 * 10 + 8
```

or in the notation of the BS2, for any Kelvin temperature:

```
TK  =  (TK/10)*10 + (TK//10)    ' integer division, plus remainder.
```

Multiplying both sides by `rct`, then dividing by ten (to get the top 5 digits of the product), we end up with the formula in the program:

```
Kal = TK/10*rct + (TK//10*rct+5/10)   ' ❒ compute and round off Kal
```

The 5 added before the final division is for rounding. Think it through.

The two `write` statements store the byte 0 of word `Kal` in the location, e`Kal`, and then byte1 of word `Kal` in the location right after that. There again are the `.byte0` and `.byte1` modifiers of the word variable, `Kal`. Subsequently, the `read` command retrieves the calibration value in exactly the same order. The calibration constant stays there unchanged in EEPROM until (1) you press the calibration button again, or (2) until you reload (RUN) the program again from the STAMP2.EXE on the PC.

Some Earth Measurements Temperature Experiments

Investigate the temperatures you find around and about your environment. Your instructor may have specific instructions. Look at hot and cold tap water, an aquarium, out in the open sun, under trees, underground. Note that it is possible to extend the length of the temperature probe, if you want, simply by adding more wire. Look for those microenvironments? Where are the sources of heat that lead to variation in temperatures in microenvironments?

Here are a few specific experiments you can try. These merely illustrate how a temperature probe can be used to measure more than just temperature.

1) Measure the air temperature in the shade.

Dry bulb temperature:_____

Then wrap wicking, gauze or cloth or a piece of paper towel around the temperature sensor and hold it in place with rubber bands or wire. Try to make the wrapping tight and compact, not too much of it! Make the cloth wet. Observe the temperature as you fan air across the wet sensor, or as you spin the probe rapidly around on its wire. It will cool down to its final value quicker if you have constructed a compact wet bulb. What is the new final temperature?

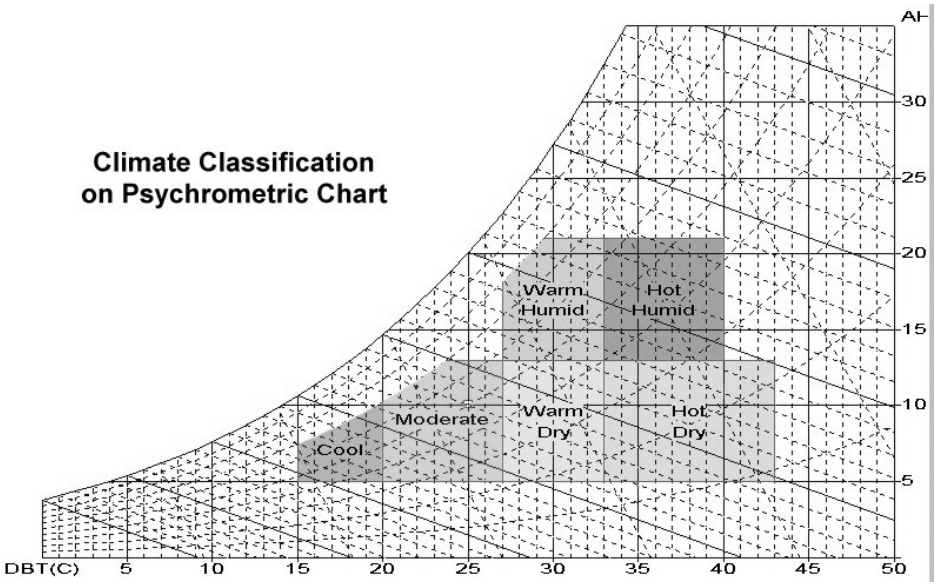
Wet bulb temperature:_____

You might expect 4 or 5 degrees Celsius of wet bulb depression in a room at 50% relative humidity. Everyone knows that a wet body cools off in the breeze. The cooling effect is greatest in dry air. This is called the wet bulb depression. It depends mainly on the relative humidity in the air, and on windspeed. At higher windspeeds it depends only on relative humidity. An instrument to measure humidity using a wet and a dry thermometer is called a "psychrometer" (drom Greek root, psychros, cold). By spinning the wet bulb through the air, it becomes a "sling psychrometer". Psychrometric charts are where you would go to look up the humidity as a function of the dry and wet bulb temperatures. Try this both inside and out of doors. For interest, an example of a psychrometric chart is shown in Figure 3.9.

Figure 3.9: Example of Psychrometric Chart

- "y" axis on the right side is relative humidity.
- "x" axis is dry-bulb temperature (C)

This chart was designed by Hong Kong University in order to classify building comfort zones.



2) Ice point depression in salt water.

You have an ice bath from your calibration of the temperature probe. When ordinary table salt is mixed with ice water, what happens to the temperature of the mixture? Think about the role of salt water in relation to ice cream makers, icy roads and icebergs. You might do some quantitative experiments with the amount of salt, or with different types of salt. Do the experiments in a well-insulated container to get the best results!

3) Solar radiation intensity from black and white thermometers.

Wrap a cylinder of aluminum foil around the temperature sensor, twisted at the end, and tie a thread to the foil so that you can pull it put it off the probe. Put it in the sun inside a clear plastic or glass jug (to cut down the wind). Record the reading when it settles down

reading with foil wrap:_____

Use the thread to pull off the foil. Compare the reading with and without the foil.

reading with black sensor :_____

Everyone knows that dark objects can get hot in the sun. A device that measures radiation by looking at the temperature difference between a black and a white surface is called a "black and white" pyranometer. (Pyr is a Greek root that means "fire", but you already knew that!).

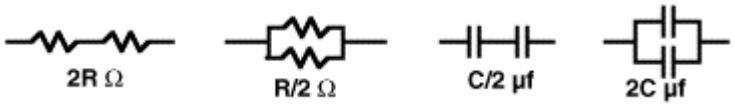
4) Windspeed from hot probe anemometer.

Allow the black temperature sensor to become hot in the sun. Then fan air across it or spin the probe around on the end of its wire. Everyone knows that warm bodies cool off in the breeze. This shows that temperature can be used to measure winds speed. "Hot wire anemometers" use a platinum wire both as the sensing element (its resistance changes with temperature) and as the heating element (electrical current passing through it makes it heat up).

change in reading:_____

Challenge!

1) Hook up a circuit as in Figure 3.3 (or 3.4), except use 100K resistor and a 0.22 uF capacitor. Measure the `rct` value, and insert it in the center cell of the table below. You have two 0.22 uF capacitors in your kit, and two 100K resistors. By making parallel and series connections of those parts, you can fill in the rest of the entries in the chart.



Rctime vs. R and C	50KΩ	100KΩ	200KΩ
0.11µF			
0.22µF			
0.44µF			

2) Modify program 3.3 so that it shows its result in degrees Fahrenheit and degrees Rankine, instead of degrees Celsius and Kelvin. (Rankine=Kelvin*1.8.). Do not just convert Kelvin to degrees Rankine though. Calculate a new constant for Rankine=constant/rct.

3) Sometimes the Morse code sound may be annoying. You can unplug the wire from P0 to shut it off. But the challenge here is to make a way to turn it off in software. Think of a way using the pushbutton to toggle the sound on and off.

4) Hook up a 0.1 µF capacitor and 100 ohm resistor to P10, as in Figure 3.4, with the conductivity probe. Also hook up an LED and resistor to P8, so that the BASIC Stamp can turn the led on and off. Then (a) make a program that turns on the led only when the probe is dipped in water; (b) Make a program that pauses for 10 seconds, then tests the input and blinks the led if the probe was dipped in water anytime during the 10 second pause. Then recharge the capacitor, turn the pin to an input, then go back to the pause; and (c) Modify the program so that it counts up how many times you dip the probe in water, one count possible for each pause. This program shows how the capacitor helps to monitor things like rain gages and traffic counters--switches that close unpredictably for short periods of time. The capacitor is used as a "memory", remembering the event until the BASIC Stamp gets around to looking at the input.

3

Vocabulary

micro-environment

analog vs digital

analog as analogy

transducer sensor probe

sourcing current

sinking current

capacitor voltage level

threshold

rate of charging

RCtime

branch

calibration bath

ice bath

273.14 Kelvin

debug rep "*"\n

integer division, remainder

accuracy, resolution

automatic calibration

wet bulb, dry bulb

psychrometer

Experiment #4: Light on Earth and Data Logging

The theme of the Light on Earth and Data Logging experiment is "light, and it's importance for everything under the sun". To demonstrate this, we'll build a light meter/data logger. The activities we'll perform in this experiment consist of: (1) Light sensor photodiode using `Rctime`, and observations of the orders of magnitude scales of intensity; (2) Combined temperature and light meter; (3) A pushbutton data logger for temperature and light; and (4) A couple of experiments using the meter/logger.

"Let there be light"

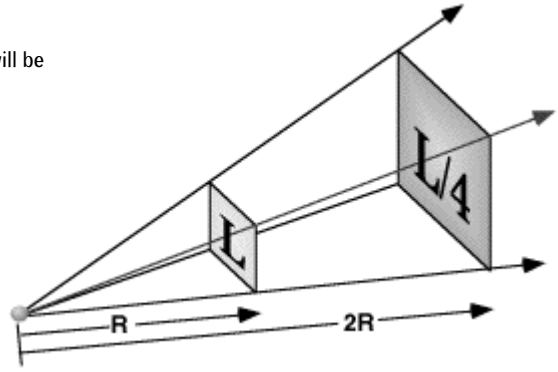
If temperature is the number one variable in earth measurements, then light would have to be number zero. The sun is the driving force of most of the weather and physical processes on earth. Where would we be without photosynthesis? People have been taking measure of the sun since the dawn of prehistory. At Stonehenge, at the Caracol of Chichen Itza, and around the world, the ancients took the measure of the solar cycle of the seasons in relation to agriculture and temporal and spiritual life.

Temperature is a relatively simple variable in comparison to light. Light comes in a spectrum of colors, both visible and invisible, and the spectrum extends out to fuzzy limits of wavelength. It has polarization and direction. Many aspects of light have special significance. Certain wavelengths are responsible for sunburn; other wavelengths are special for the ripening of fruit. There are subtle patterns of light. For example, bees can see patterns of deep blue on flowers that the human eye cannot perceive, and hummingbirds vision extends farther into the red than our own. Light is important to us in a tremendous range of intensities, from solar energy for electricity and heating, to bioluminescence of creatures in the deep oceans.

Like temperature, light is often used to measure other things. For example, instruments for detecting air quality and CO2 are often based on lasers or on the fact that gases absorb light at characteristic wavelengths. Astronomers use the spectra to deduce the chemical composition of the stars and interstellar gasses. On the other end of the scale of size, light is used to probe chemical processes in the DNA and mechanisms of the living cell. In a practical arena, light is used for motion detectors, for indicators, and of course for illumination, which is itself a whole specialty of engineering.

One fundamental law is that the light from a point source falls off with the square of the distance. That is, at double the distance from a light bulb (or from the sun), the light intensity will be 1/4 of its value at the first location. The same amount of energy is spread over 1/4 of the area. Using the light meter you build in this lesson, you will have a tool to investigate that law, as well as to explore light variation in your environment. This concept is shown in Figure 4.1.

Figure 4.1: Light Attenuation
At twice the distance from a bulb (or from the sun), the light intensity will be 1/4 of its value at the first location. We'll investigate this law with our microcontroller-based light meter.



Parts Required

For this experiment we'll be leaving parts on the Board of Education that we installed in Experiment #3. The following parts are required for Experiment #4:

(1) photodiode (Photonic Detectors' C113 or V113)
(4) 100 ohm resistor (brown black brown)
(2) 0.01 uF film capacitor (103)
(1) 0.22 uF film capacitor (224)
(2) 100 pF ceramic capacitor (101)
(1) red led
(1) green led
(1) 9 volt battery
(1) 50 watt R20 spotlight (for experiments)

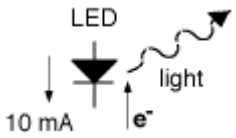
Build it!

Photodiode as a Light Transducer

In "What's a Microcontroller" you used a photoresistor (also called a photoconductive cell) to detect imaginary people approaching a supermarket door, where a servo motor opens the door automatically to let them in. A wildlife photographer or biologist might use a similar arrangement to detect an elusive animal, to trigger the shutter of a camera. In this lesson, we will use a different type of photodetector, a photodiode. A photodiode passes an electrical current (in amps) that is directly proportional to light intensity. This characteristic makes it especially well suited for quantitative measurements.

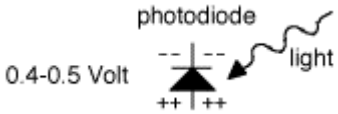
You are familiar with a light emitting diode, which turns electrical current into light. The led emits light when the current flows in the direction of the diode arrow. You may know that electrons (negative charges, e-) actually flow in the opposite direction. But in an accident of historical interpretation, in electronics we usually think of current as if it were carried by positive charges. Again, the LED emits light when the (positive) current flows in the direction of the arrow.

Figure 4.2: LED
The LED emits light when the (positive) current flows in the direction of the arrow.



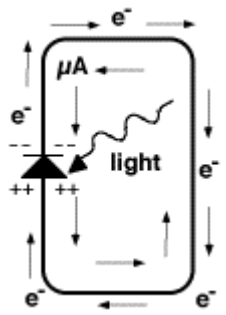
It also works in reverse. Light falling on a diode produces electricity. If you connect a voltmeter to a diode exposed to light, you will measure a fraction of a volt, with the polarity as indicated. Electrons accumulate at the cathode end.

Figure 4.3: Photodiode
A photodiode produces electricity with light.



When you connect the diode into a circuit loop with a wire, then current flows around the loop. The amount of current is proportional to the intensity of the light. This is fundamental. It is the light that generates the charges that make the current. The only way they can get back together is to flow around the external circuit. Observe that the electrons flow clockwise around the circuit. The conventional current (positive charges) flows in the opposite direction, against the direction of the diode arrow. This is called a photocurrent, and it is a reverse current. Compare this to the forward current that lights up an LED. This business of the arrow can be confusing, but in electricity, it is all relative. In this circuit, the voltage across the diode is zero--it is short-circuited.

Figure 4.4: Photocurrent
Photocurrent occurs when conventional current (the positive charges) flow in the opposite direction, against the direction of the diode arrow.



Sensitivity to light is a fundamental property of transistors and diodes. In most places, that would be an undesirable side effect. Transistors and integrated circuits are usually potted in a plastic, ceramic or metal case, and one reason to do that is to keep out light that would greatly affect their performance. Photodiodes are made especially to accentuate the sensitivity to light. Look at the C113 photodiode in your kit. It has a clear epoxy top, and underneath that you can see a small black square of silicon, with wires attached at the sides. Electrical charge builds up on the top and the bottom, where the wires pick it up. The difference between a solar panel and a photodiode is largely in the area of the diode. Solar panels have huge areas, square feet or square meters, so they can intercept a lot of light and produce lots of current and power, measured in amps and watts. The photodiode is made of especially pure material for measurement, not for energy production.

One thing that makes photodiodes very useful for measurement is that a simple equation governs their behavior as a transducer:

i = constant * (light intensity)

The sensor is linear. That means that if the light level increases, say, by a factor of 1000, then the current through the diode will also increase by that same factor. For the photodiode, this holds true over several orders of magnitude, over several powers of ten. It is that characteristic that makes it so useful for measurement.

The simple equation also holds true when the diode is hooked up in a reverse voltage circuit as in Figure 4.5. At the same light intensity, the amount of current is exactly the same here as it would be in the short circuit of Figure 4.4. The current through the photodiode charges the capacitor. The charge accumulates on the capacitor as shown, and the voltage across the capacitor gradually increases. The BASIC Stamp II program will measure the time it takes for the voltage at P6 to fall from 5 V down to 1.3 V as shown in Figure 4.5.

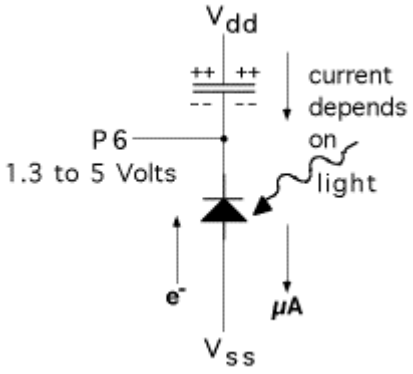
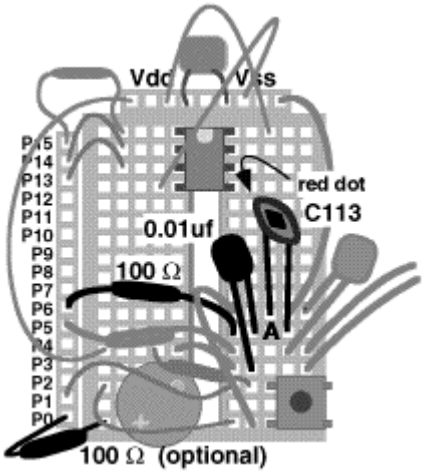


Figure 4.5: Photodiode Circuit
With this circuit the BASIC Stamp can measure the time it takes for the charge to fall from 5 V to 1.3 V. This is called a resistor/capacitor circuit.

So let's built it! The pictorial is shown in Figure 4.6, and schematic is in Figure 4.7.

Figure 4.6 Photodiode Light Transducer



- 0.01 uF film capacitor (marked 103) from Vdd node (same one as AD592 red wire) to **node** A.
- Photodiode anode (with red dot) to Vss node.
- Photodiode cathode to node A.
- 100 ohm resistor from node A to P6

- Optional: if you want to quiet down the sound from the annunciator, replace the wire from P0 to the annunciator with a 100 ohm resistor.

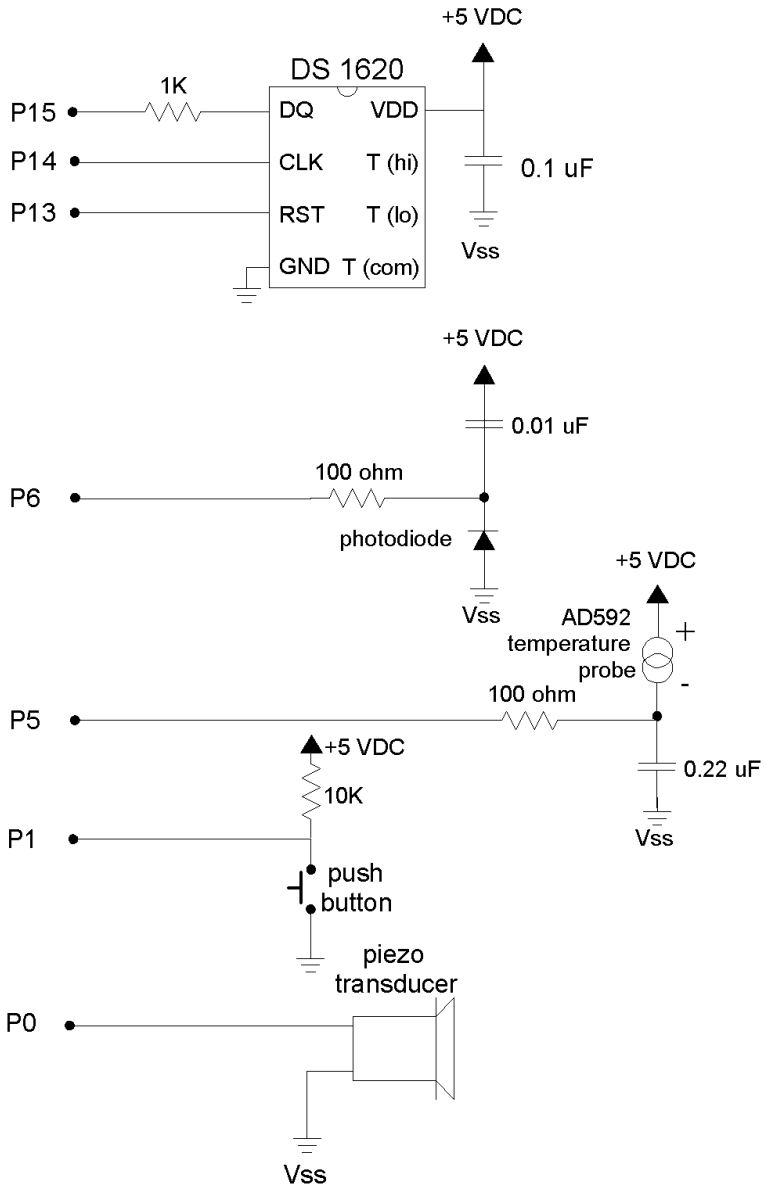


Figure 4.7: Photodiode Light Transducer Schematic

An optional resistor may be installed between the piezo transducer and BASIC Stamp PO to quiet down the speaker.

What's a...
Node:
A node consists of all the points in a circuit that are connected together. Each row of 5 holes on the Board of Education is a node, because all of the holes are connected electrically. One node in circuit 4.6 is the point where the diode, the capacitor and P6 meet.

Enter the following program:

```
' Earth Measurements program 4.1
' Sound out light levels from the photodiode
rct  var word              ' variable for RCtime
high 6                     ' discharge the capacitor
loop:
  RCtime 6,1,rct           ' time for volts to fall to 1.3
  high 6                   ' discharge the capacitor
  if rct=0 then loop       ' no sound if RCtime overflows
  freqout 0,1,3400         ' make a click
goto loop
```

After you run the program on the BASIC Stamp, expose the light sensor to a variety of light conditions, from dim light, to light bulbs, to sunlight if available. Read the next couple of paragraphs before you get too far though.

Light levels can vary tremendously in the natural environment. Our eyes have an amazing capability to accommodate to both dim and to bright light. Sensitivity is the amount of light that it takes to get a response. Sometimes we need a sensitivity adjustment, or a switch for "high" and "low" sensitivity. Cameras and eyes have an iris that opens or closes to adjust the amount of incoming light, to extend the range of sensitivity.

Observe that the program jumps back to the top without making a sound, if the value of `rct` is equal to zero. That is at the dim end of the range. You may have to cover the sensor with a box or something to make it dark enough to see this effect. The capacitor takes too long to charge and the `RCtime` command does not see a transition from 1 to 0 at P6 within its 0.13107-second limit.

- To make it respond better in dim light (more sensitive), replace the 0.01f capacitor with 100 pf (picofarads, marked 101).

At the other extreme, in bright light, the clicking becomes very high-pitched and bunched up, so you can't distinguish differences.

- To make it respond better to bright light (less sensitive), substitute a 0.22f capacitor (marked 224), or put a piece or two of tissue paper over the sensor, held it in place with a rubber band, in order to decrease the sensitivity.

Now, run the Board of Education from a 9-volt battery and carry it around your environment. (Be sure the battery is good-weak batteries can lead to puzzling results!) Carry along the 100pF and a 0.22 uF capacitor and change the sensitivity whenever you feel it is necessary.

- Try pointing the sensor both up and down, to detect the direct light and the reflected light and patterns of light and shade.

- Try scanning close across the objects on a desk, or close to the bold pattern on a curtain or clothing, or the flickering pattern on your computer screen or TV set.

- Try scanning in a relatively dark place (using the 100pF capacitor), as well as outside in sunlight, if available (using the 0.22 uF capacitor).

Have fun with this!

What's a...

Light intensity:

- A **Pyranometer** quantifies light intensity by the light energy hitting a surface per unit time. This is the right measurement if you are designing a solar panel system or a solar water heater, or if you are an architectural engineer thinking about the energy efficiency of a building. This kind of light intensity is measured in watts per square meter. The power input from the sun at the earth's surface, on a clear summer day, is a little over 1000 watts per square meter, or 75 watts per square foot. The solar energy input above the atmosphere is nearly 1400 watts per square meter, which is often given in other units as 2 Langley's per minute. (1 Langley=1 calorie per square centimeter). Sometimes we are interested only in the energy of certain wavelengths. For example, ultraviolet light at about 300 to 320 nanometer causes sunburn, not only in people, but in other life on earth too, like coral in the ocean. This UVB light can be separated out and measured. It amounts to less than 0.1% of the total, less than 1 watt per square meter. But it is a very significant 0.1%. More and more UVB is getting down to the earth as the ozone in the upper atmosphere is depleted, due it seems to human activity, the use of certain CFC chemicals.

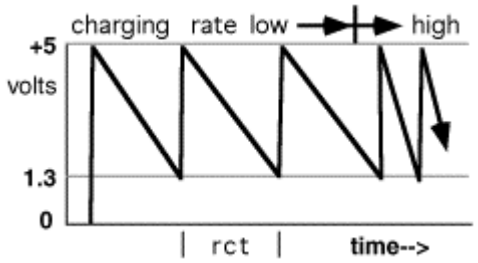
- A **Photometer** quantifies light intensity is as our eyes see it. This is the subject of illumination engineering. And physiology. How does an owl see? Our eyes are most sensitive to bright light in the yellow-green, and the sensitivity falls off in the red and the blue. The question of light intensity is still one of energy per unit area per unit time, but now it includes only energy at the wavelengths we can see. And it has special units, lux or footcandles. The light intensity looking at full sunlight is about 110,000 lux, but of course, looking at direct sunlight is not something that we do. It is too intense. In contrast, a 100 watt light bulb, viewed from 1 meter distance, is about 100 lux. That too is perceived as bright. Normal room lighting is measured in 10s of lux. There is a tremendous range of values that sensors, including our eyes, have to deal with. 7 or 8 orders of magnitude. But that is nothing compared to the range of light levels arriving from celestial objects, which is detectable over 20 orders of magnitude.

- A **PAR meter** quantifies light intensity is as it effects the growth of plants. This is of great interest to farmers, and aquaculturalists, and botanists. Photosynthesis occurs in special band of wavelengths, called the photosynthetic action spectrum. PAR stands for Photosynthetically Active Radiation. Measurement of PAR allows botanists to estimate how much growth would be possible for a plant, if light were the limiting factor. Light comes in packets of energy, called quanta, and each unit of photosynthesis take one quanta of light. The units of PAR are micromoles of quanta per square meter per second. Full sun illuminosity is about 2000 moles per square meter per second. A lot of plant biology has to with plant adaptations to light levels.

- A **Spectrophotometer** is the most versatile of the bunch. It tells you how much light energy is in each narrow band of wavelengths across the spectrum. In contrast, the photocurrent in your photodiode is due to the convolved effect of many different wavelengths. The spectrophotometer can be used to characterize and calibrate almost any of the other light measuring instruments, but, needless to say, it can be a much more complicated and expensive instrument. An economical version of this is a colorimeter, that you might find in a paint store for matching colors.

Let's talk more about the BASIC Stamp program. One side of the capacitor is connected to Vdd (+5) (instead of to Vss, where it was in the AD592 temperature sensor circuit). Here, the program starts off with `HIGH 6`, to discharge the capacitor. This might seem strange, to make the pin high to discharge the capacitor, but note that a capacitor is said to be "discharged" when the voltage from one side to the other is equal to zero. The top of the capacitor is connected to +5 volts, so `HIGH 6` makes both sides equal to +5 volts, so it is discharged. Figure 4.8 illustrates the time course of the voltage at P6.

Figure 4.8: Resistor/Capacitor Discharge
Current sinking through the photodiode gradually charges the capacitor down until the voltage at P6 reaches the 1.3 volt threshold. The RCtime command holds P6 as an input during that time. As soon as the RCtime detects the 1.3 volt level, the program moves to the HIGH 6 and quickly discharges the capacitor, and the voltage at P6 quickly returns to +5 volts.



Look carefully at the `RCtime` command in Program 4.1. The second parameter is now a 1. That parameter was 0 in the programs in Experiment #3 with the AD592 temperature transducer. The 1 instructs the `RCtime` routine to count time while P6 is equal to one, and to stop when P6 makes the transition to zero. Try changing the second parameter from 1 to 0. It doesn't work, does it? That is, is it sensitive to light? Do you hear a very high tone, a very low tone, or no tone at all? For debugging, it is good to think these "what if..." kinds of questions.

The BASIC Stamp provides both forms of the command, with the second parameter either 0 or 1, because there are situations where one or the other will be the best or the only choice. For example, the AD592 temperature sensor works fine in the circuit of Experiment #3, but it would fail to work in this circuit of Experiment #4. That has to do with the voltage requirements of the AD592. We can't go into all the advantages and disadvantages of one circuit over the other, but it is good to be aware of this flexibility when you get down to the fun of designing your own serious projects.

Observe that the pitch of the tone from the annunciator goes higher in brighter light. How come, when the time to discharge the capacitor, `rct`, goes lower in brighter light? Be sure you understand the reason, that with lower values of `rct`, the program loop cycles faster, which makes a higher tone.

Photodiode and the BASIC Stamp as a Digital Light Meter

The previous program was an analog meter, because the frequency output was an analog of the light input. Both go up and down in a similar manner. Analog meters are great conveying information directly to our senses. Let's look at the actual numbers on the debug screen. This is the digital connection. Modify the program as follows (lines remarked with a ❒ are the ones you need to add or change):

```
' Earth Measurements program 4.2
' Numerical light levels from the photodiode
rct  var word                    ' variable for RCtime
light var word                   ' ❒ variable light intensity
high 6                           ' discharge the capacitor
loop:
  RCtime 6,1,rct                 ' time for the volts to fall to 1.3
  high 6                         ' discharge the capacitor
  light=65535/rct                ' ❒ light=constant/rct
  debug dec rct,tab,dec light,tab, bin light,cr ' ❒ display
  pause 400                      ' ❒ slow things down
goto loop
```

The constant 65535 is arbitrary. The important thing is that the light is proportional to 1/rct. The exact value of the proportionality constant will be determined when we calibrate the sensor against a light source of known intensity, like the sun, or a standard light bulb.

Theory:

For those of you who are interested, the theory here is the same as for Experiment #3. The voltage across the capacitor here must change by 3.7 volts (see Figure 4.8), instead of 1.3 volts (see Figure 3.7). The formula is:

2*rct = C * 3,700,000 / i

(2*rct) is in microseconds, C is in microfarads, and i is in microamps. This shows the theoretical inverse relation between the photocurrent, i, and the rct variable that comes out of the `RCtime` command. The photocurrent is proportional to the light intensity falling on the photodiode. However, unlike the AD595 temperature probe, where the calibration constant is 1 uA/K, there is not an exact equality between standard units of light and photocurrent. But it doesn't matter. All the constants can be lumped into one that can be determined at the time of calibration:

rct =constant / (light level) or light level = constant / rct

When you run the program, observe how the numbers in the second column on the screen increase in bright light. And the numbers in the first column decrease, as the capacitor charges more rapidly from the larger photocurrent.

The final column is included, not so much so you can see the binary value of the light level (BASIC Stamp Manual Version 1.9, page 256), but because the length of the binary number, from 1 to 15 binary digits, is proportional to the logarithm of the light level. One digit is added for every doubling of the light intensity. Try taking your light meter from very dim to very bright to see what we mean. Logarithms are useful for dealing with phenomena that vary over huge ranges. As another example, the VU meter on a stereo sound system shows you a logarithmic graph of sound level. Our ears, like our eyes, can accommodate over a tremendous range of sound levels. In technical terms, the length of the binary number shows the integer part of the logarithm to base 2. The same is true for the decimal number in the second column, that it adds one digit for each factor of 10 increase in light level, but that is harder to perceive. You can try replacing `bin light` in the `debug` statement with `rep 42\ ncd light`. This prints a string of stars, instead of the actual binary number. The `ncd` operator (BASIC Stamp Manual Version 1.9, page 237) is the closest thing the BASIC Stamp has to a logarithm function.

Let's look now at the effect of the capacitor on the reading. With a 0.01 uF capacitor in the circuit, place the Board of Education in a place where the light is constant, and note the reading. Now find the second 0.01 uF capacitor in your kit, and install it on the Board of Education in parallel with the first one, as in Figure 4.9. Now (with the same light level as before) observe the `rct` and the light level readings. The `rct` reading should be about twice what it was before, and the light reading in the second column should be about 1/2 of what it was. This is because the value of the capacitor enters into the calculation of the calibration constant, as shown in the theory box. If the current from the photodiode is constant, then doubling the capacitor value also doubles the time it takes for the capacitor to charge down to the threshold.

Figure 4.9: Parallel Capacitors
By adding the second capacitor in the circuit in a parallel fashion, the rct reading will double from what we've measured before. The value of the capacitor enters into the calculation of the calibration constant.

The capacitors we are using are of a type called "polyester film" capacitors. They are desirable because of their stability. Temperature changes do not affect their capacitance. We need a stable capacitor like that, so that the `RCtime` value will only depend on the current from the photosensor.

Now remove both of the capacitors, so that there is no capacitor at all in the photodiode circuit. (Be sure you are getting the photodiode capacitor, not the temperature capacitor). It still works! The actual numbers don't mean anything, but you should find that it acts just as if there were a capacitor in the circuit. It will be sensitive to very low levels of light. Take it into the shadows to see. As a matter of fact, there is a capacitor in the circuit. The input gate on the PIC 16C57 microcontroller on the BASIC Stamp has a built-in capacitance of about 50pf (picofarads). In addition to that, the wiring of the white block breadboard contributes capacitance. Remember, capacitance exists whenever electrical conductors come close to one another, by intent or not. This is called stray capacitance, because it was not really intended to be there as part of your circuit. The combination of the PIC input capacitance and the capacitance of the white block wiring add up to the equivalent of about 250 pF of stray capacitance. Put in a 100 pF capacitor where you just removed the 0.01 uF one. The reading will not change too much. The capacitance has only gone from about 250 to 350 pf, not from zero to 100 as you might have expected. Put a second 100pf capacitor in parallel with the first. The readings will not change by a factor of 2, because the capacitance has gone from about 350 up to 450, not from 100 to 200. Often when things do not turn out the way you expect, it is due to stray circuit elements for which you are not accounting.

Now remove the two 100 pF capacitors, and restore the single 0.01 uF capacitor in your Board of Education light meter. Hold the light meter in a bright light source. If you have it, use a 50-watt R20 spotlight (the kind used in track lighting). If you don't have this type of bulb an alternative would be a 100 watt bulb. The light intensity of such a light source facing the center of the beam at one-meter distance is about 425 lux (40 candlepower). Write down the light reading you see in the second column on the computer screen.

light (raw reading, second column) = _____

This may be easier said than done. You may find that the reading on the computer screen fluctuates up and down quite a bit, making it hard to decide what the "reading" really is. These fluctuations come from a couple of different sources. Not the least of them is that the light level is really fluctuating, very fast, faster than your eye can perceive it. The intensity of the lamp depends on the power line voltage, so we should really emphasize again that its intensity is approximately 425 lux at one meter. The AC line voltage that drives the lamp goes from zero to 170 volts, 120 times per second. As this happens, the intensity flickers. The filament in the lamp stays glowing hot, due to its thermal mass, but the output does fluctuate on a time scale of 1/120th of a second, about 10 milliseconds. Along comes the photodiode and the BASIC Stamp, to sample the light in less than one millisecond. Sometimes it samples the highs, and sometimes it samples the lows.

There are ways to average all this out, but for now, just take it as a lesson. Make a rough estimate of your average value. Look at the readings first to find a minimum value, and then look for the maximum value. Make your estimate halfway in between.

minimum? _____
maximum? _____
mean? _____

Now, see if you can find a scale factor so that the program displays the numerical value in standard units of 425 lux, instead of the raw value in arbitrary units, when the sensor is in position in front of the lamp.

(raw reading) * (scale factor) = 425 lux

For example, if the raw reading happens to be 168, then on a calculator you would multiply times 2.53. That number is found by using

(scale factor) = 425/168 = 2.53

Subsequently you can move the light meter into an unknown area and find the actual light level there in units of lux, because light meter is calibrated.

(new light level in lux) = (raw reading) * 2.53

The trouble is, the BASIC Stamp (like most microcontrollers) uses integer math. It does not have fractions. Well, not quite true. The BASIC Stamp II has a peculiar math operator called `*/` that is called fractional multiply. The catch is that the fraction has to be one of these specific values: 0, 1/256, 2/256 and so on, up to 256/256 (unity) and so on: 257/256 (one + 1/256th), up to 65535/256 (255 + 255/256ths). All these fractions have a denominator of 256. The factor that goes on the right side of the `*/` is the numerator of the fraction, and the denominator of 256 is implied. Here are some examples:

```
Y = X */ 256  ' is the same as Y = X, because 256/256=1
Y = X */ 128  ' is the same as Y = X* 1/2, because 128/256 = 1/2
Y = X */ 384  ' is the same as Y = X* 3/2, because 384/256=3/2
Y = X */ 647  ' is the same as Y = X * 647/256 ...
```

It so happens that 647/256 is the close to 2.53, which is the scale factor we need. Try it:

647/256 = _____ close to 2.53?

168 * (647/256) = _____ close to 425?

Here is how to find the value to put on the right side of the */. You have the raw light reading, say it is 168. You have the known value of light intensity, say 425 lux. Then the factor to put on the right side of the */ is 425*256/168=647. Using your value from page 88 in place of the 168:

425 * 256 / (your raw reading) = _____

The equation in the program now becomes:

```
light = 65535/rct*/647       ' computes light from left to right
```

(but you use your own constant in place of the 647). When you run the program with this change, it should display 425 on the debug screen in the second column, when the sensor is placed one meter in front of the calibration source. Now, as you take the light meter around the room, the reading will be displayed in standard units of lux. This is a "ballpark" calibration. But it demonstrates the idea, and how the */ operator can help with the math (BASIC Stamp Manual Version 1.9, page 242). Calibration of analog sensors often involves multiplying a raw reading times a fraction, so understanding how to use the */ operator can be a great help.

Temperature and Light Meter

Do you still have your temperature sensor probe hooked up? We hope so. If not, hook it back up according to Figure 3.8, and enter the following program. Please enter your own values for the calibration constant for the light sensor, the one you just found, 647 in our example. And also enter your own constant, `Kal`, for the AD592 temperature sensor, from Experiment #3 on page 61. That number goes in the place of the 15068 below.

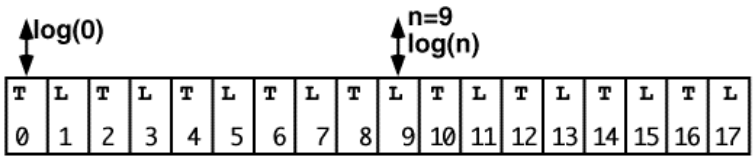
```
' Earth Measurements program 4.3
' light intensity and temperature
kal     con 15068               ' ❒ calibration constant for AD592 temperature.
lical   con 647                 ' ❒ calibration constant for photodiode
                                      ' Use Your Own Calibration Constants!!

rct     var word                ' variable for RCtime
light   var word                ' variable light intensity

TC var word                     ' ❒ for degrees Celsius from AD592
low 5                           ' ❒ discharge the temperature capacitor
high 6                          ' discharge the photodiode capacitor
loop:
  RCtime 5,0,rct                ' ❒ read the temperature probe
  low 5                         ' ❒ discharge the temperature capacitor
  TC=kal/rct*10+(kal//rct*10/rct)-273         ' ❒ calculate Celsius
  RCtime 6,1,rct                              ' read the photodiode
  high 6                                      ' discharge the photodiode capacitor
  light=65535/rct*/lical                      ' ❒ calculate lux
  debug dec TC," C",tab,dec light," lux",cr   ' ❒ display
  pause 400                                   ' slow down the loop
goto loop
```

Now you have both temperature (first column) and light readings (second column) on screen, with units. That's progress! Compare the two `RCtime` instructions that go with the temperature and the light level. Be sure you understand why the two instructions are different, in relation to how the circuits are set up. Get this working before you proceed to the next section.

Light and Temperature Logger, Using RAM Memory

Now let's make the program store up a bunch of readings. We have already discussed in earlier lessons why data logging is important. It's time to get down and do some of it. The ultimate goal is to log data in the EEPROM memory of the BASIC Stamp. But for now to keep it simple we will log data in the RAM memory of the chip. Recall from Experiment #1 that there are only 26 bytes of RAM available for our general purpose use in the BASIC Stamp II. We will set aside 18 bytes of that for the data log shown in Figure 4.10.

Figure 4.10: Allocation of Memory for Log File



The instruction to reserve 18 bytes in RAM will be,

```
log    var  byte(18)
```

These bytes are like 18 bins in a row, numbered from 0 to 17, and we are going to store temperature values in the even numbered bins and light intensity values in odd numbered bins. In the program, will refer to these bins by using an index inside parentheses:

```
log(0)=TC           ' stores a temperature reading in the first bin
light=log(9)        ' retrieves a light reading from the 10th bin.
light=log(n)        ' bin number is held in a variable, n.
                    ' when n=9, retrieves the light value from the 10th bin
```

On signal from the pushbutton, the program will acquire a temperature and a light level reading, and it will store the numbers in the next two available bins. When all the bins are full, the program will make a special protest beep to annunciate the fact, "memory full". When you make a special long press on the pushbutton, all of the values in the 18 bytes will be displayed on the debug screen. To erase the data and start over, all you have to do is press the reset button. That "re-zeros" all the bins and the pointer.

Modify program 4.3 as follows (changed or new lines are shown with the ❒):

```
' Earth Measurements program 4.4
' temperature and light intensity logging in RAM
kal     con 15068                ' calibration constant for AD592 temperature.
lical   con 647                  ' calibration constant for photodiode
                                      ' Use Your Own Calibration Constants!!
log     var byte(18)             ' ❒18 bytes reserved for the log file.
rct     var word                 ' variable for RCtime
light   var word                 ' variable light intensity
TC      var word                 ' for degrees Celsius from AD592
n       var byte                 ' ❒counter for the pushbutton
ptr     var byte                 ' ❒pointer to next entry in data log file

outs=%0000000001000000           ' ❒ now put in the outs and dirs statements.
     'fedcba9876543210
dirs=%1111111111111101           ' ❒ all are low outputs
                                 ' ❒ except P6 is high output to discharge C
                                 ' ❒ and P1 is input for pushbutton

debug cls,"ready to log data!",cr' ❒
freqout 0,200,2550               ' ❒
freqout 0,400,3400               ' ❒
mainloop:                        ' ❒
  if in1=1 then mainloop         ' ❒ loop here until the button is pressed
  n=0                            ' ❒ this will count how long the button is down
clik1:                           ' ❒
  pause 100                      ' ❒ time the button in 0.1 sec increments
  if n>12 then playback          ' ❒ jump to the playback routine after 1.2 second
  n=n+1                          ' ❒ increment time
  if in1=0 then clik1            ' ❒ loop until button goes up, or time runs out
getdata:                         ' ❒
  if ptr>17 then honk            ' ❒ protest if the memory is full
  freqout 0,10,1900              ' ❒ sound to show we got here

  RCtime 5,0,rct                 ' read the temperature probe
  low 5                          ' discharge the temperature capacitor
  TC=kal/rct*10+(kal//rct*10/rct)-273  ' ❒ calculate Celsius
  log(ptr)=TC                    ' ❒ store temperature
  ptr=ptr+1                      ' ❒ point to next bin

  RCtime 6,1,rct                 ' read the photodiode
  high 6                         ' discharge the photodiode capacitor
  light=65535/rct*/lical         ' calculate lux
  log(ptr)=light/2 max 255       ' ❒ store light intensity/2
```

```
  ptr=ptr+1                              ' ❒ point to next bin

  debug dec TC," C",tab,dec light," lux",cr  ' display!
goto mainloop

playback:                                ' ❒ Show all 9 readings on debug screen
  freqout 0,50,2550                      ' ❒ feedback
  freqout 0,100,3400                     ' ❒
  debug cls,"logged data!",cr            ' ❒ message on screen
  debug "degC",tab,"Lux",cr              ' ❒ print units of measurement
  for n=0 to 16 step 2                   ' ❒ will show 9 records
  TC=log(n)                              ' ❒ get temperature
  light=log(n+1)*2                       ' ❒ get light
  debug dec TC,tab,dec light,cr          ' ❒ display
  next                                   ' ❒ next record of 9
pb1:                                     ' ❒ finished showing
  if in1=0 then pb1                      ' ❒ wait for the pushbutton to go up
  debug cr,"press RESET to erase data",cr  ' ❒ print message
  goto mainloop                          ' ❒ back to the top

honk:                                    ' ❒ come here if memory is full
  debug cr,"memory full"                 ' ❒ message
  freqout 0,50,3400                      ' ❒ noise
  freqout 0,200,2000,2100                ' ❒
goto mainloop                            ' ❒ back to the top
```

When you run the program, you should be able to press the button 9 times to collect 9 records of temperature and light level. After that, the program should make the "memory full" beep. At any time, if you hold down the button for >1.2 seconds, then the program should display all 9 of the **records** on the screen. You can leave the debug window open on the screen, and go off and do an experiment to collect 9 records, and then come back to the computer to play them back. When you are ready to start over, press the reset button on the Board of Education.

Records and fields:
Each line, or row, of data is a record. Each reading across is a field. The first field here is temperature in units of degrees Celsius, the second field is light in units of lux. The fields line up in columns. The file here consists of up to 9 records. This terminology is commonly heard in relation to spreadsheets and databases.

In the Windows version of the STAMP2W.EXE software, you can open the debug window whenever you want. In the DOS version, you have to reload the program to reopen a debug window. If you are using the DOS version of the software, just leave the debug window open when you go off to do an experiment.

We'll comment on the program itself, then move on to some experiments you might want to try.

This program starts off with the outs and the dirs statements. They replace the HIGH 5 and LOW 6 that were there in Program 4.3. Note that the sixth position in the outs statement is 1, and the 5 position is zero. That makes P6 high and P5 low, as is required to discharge the two capacitors. The P1 position in the dirs statement is a zero, which makes P1 an input. The other pins on the BASIC Stamp are all set to be low outputs, as a matter of good programming practice.

The program starts off with familiar single click and long click routines for the pushbutton. The label here is not "clik" and "clik1", but you should recognize it at the same code. When the button is down, there is a race between the timer and the button. If the timer passes the 1.2 second mark, then the program jumps to the playback routine.

But if the button is released before the 1.2 seconds, then the program goes right into the getdata routine. There, it reads the temperature probe and the light probe just as in Program 4.3. Then it puts the temperature value in the bin that is pointed to by the variable, ptr. Then ptr is increased by one. Then it puts the value of light/2 into this next bin. Then ptr is increased by one again to point to the next empty bin.

At the top of the getdata routine, the program tests the value of the pointer, and jumps to the memory full message if the pointer is greater than 17. The program does not allow itself to collect more data than will fit in the allotted space. You might try taking out this line, just to observe what sort of error will occur!

Why log the value of light divided by 2, instead of simply light as it is? The bins only hold byte-size quantities, less than or equal to 255. That is fine for Celsius, which will be in the range of 0-100. But the light level might get higher than that. It was calibrated at 425 lux. By dividing by 2, light values of up to 511 lux can be stored. The downside is a loss of resolution, but it is not significant in light of our "ballpark" calibration. When we read the light values from the log, we will multiply them by two to get back the original value. Note that light/2 is followed by max 255. The 255 is a signal that your data is out of range.

The playback routine uses a for-next loop to step through all 9 records.

```
for n=0 to 16 step 2                    ' ❒ will show 9 records
```

The "step 2" makes the value of the index, n, take on the even values, 0,2,4,6,...,16. That is a total of 9 steps. (See the BASIC Stamp II manual, page 261.) The value of temperature is read first from log(0), and light from log(1), and these values are displayed on screen. Note that light is multiplied by 2 to reconstruct the original value. Then the for-next loop bumps up the value of the index to 2, and fetches and displays log(2) and log(3), and so on up through log(16) and log(17).

Why does the program use `n` as the pointer, instead of `ptr`? First, understand that it doesn't matter what variable goes in parentheses after the `log()`. The only thing that matters is the numerical value of what is in the parentheses. By using "n" as the pointer, the value of "`ptr`" is not disturbed. Let's say you have already acquired 5 readings, and then you do the long click to read out what you already have for those 5. You can then continue where you left off and collect readings 6 to 10, and then do the long click to read them all out. It is just a small refinement.

At the end of the playback routine, the program reminds you that you have to press reset in order to start over with a clean slate.

This program uses every single RAM variable available in the BASIC Stamp. Two bytes each are used for the word variables, `rct`, `TC` and `light`, and one byte each for the indexes `n` and `ptr`, and 18 bytes for the log file. That adds up to 26. Add one more variable to the program. When you try to run it, you will get an error message, "out of variable space."

Experiments with the Data Logger

1) Verification or 1/r^2 dependence for the light source.

Prepare a string with knots at one meter, 1.5 meter, 2 meters, 2.5 meters and so on up to 4 meters. Connect the string to the side of the fixture holding the 50 watt, R20 spotlight. Reset your Board of Education to clear the data log. Collect data at each distance from the light source, taking care to stay in the center of the beam. Upload your data to the PC debug screen. Graph the readings as a function of position. Verify that the intensity of light falls off as $1/r^2$. This is a "ballpark" experiment. Think of some factors that might make the results less than perfect. Don't forget about the fluctuations we discussed earlier!

2) Investigation of the light distribution from the spotlight.

Prepare a protractor, using the above string, and some ingenuity, so that you can hold the light sensor at 10 different angles around the center of the beam from the spotlight. Collect the readings and graph them.

3) Rate of heating and cooling, timed logging.

Hold the temperature probe a few inches from the lamp, and press the button at regular 15 second intervals. Then take the sensor away take 4 more readings at the same rate. Upload the data and graph the temperature as a function of time. Wouldn't it be nice to have the data logger press the button for you, at regular intervals? Easily done. Change the `mainloop` routine as follows:

```
mainloop:                       ' changes to program EM4.4
n=0.....................        ' ❒ initialize the time counter
ml1:                            ' ❒ loop here until button or time
pause 1000                      ' ❒ one second pacing
if n=15 and ptr<17 then getdata   ' ❒ get data at 15 second intervals
n=n+1                           ' ❒ count time
if in1=1 then ml1               ' ❒ can press button to get data too.
n=0
clik1:                          ' ... and so on as in program EM4.4
```

You can put in whatever number of seconds you need. Press reset to start your experiment. The routine still accepts clicks for data logging. The number n can be up to 65535 seconds--more than 18 hours between readings--if you care to start a long term experiment!

4) Alternative scale for the light sensor, for brighter light, easy method.

If you want to measure brighter light, outdoors in sunlight for example, here is an easy "ballpark" way to go about it. Put the 0.22 uF capacitor in place of the 0.01 uF capacitor in the photodiode circuit. Put a factor of 22 in the three lines that calculate the light intensity:

```
light=65535/rct*/lical*22           ' calculate lux
```

and

```
log(ptr)=light/44 max 255           ' ❒ store light intensity/44
```

and

```
light=log(n+1)*44                   ' ❒ get light
```

This works because the new capacitance is 22 times the old capacitance. The `rct` value that used to be produced at 100 lux is now produced at 2200 lux. Recall the effect of doubling the capacitance earlier in this lesson. The old range of measurement was 0 to 512 lux. That is now 0 to 11264 lux.

5) Alternative scale for the light sensor, calibration in full sun.

This will give a reading in PAR, in units of micromoles quanta per square meter per second. If you read the box about light intensity, this is the measurement used for plant growth. This is again a ballpark calibration! Our photodiode does not have the filters that would limit it to the wavelengths best for plant growth. Put the 0.22 uF capacitor in place of the 0.01 uF capacitor. Temporarily make the constant `lical` equal to 256, and take out the divide and multiply factors in the light calculations:

```
lical   con 256             ' ❒ calibration constant for photodiode
log(ptr)=light max 255      ' ❒ store raw light intensity
light=log(n+1)              ' ❒ get light
```

Recall that in using the `*/` operator, `*/256` is like doing nothing. It is the fraction 256/256, unity. Place the photodiode sensor directly facing the outdoor sun. Of course, you may have to to go outside with the Board of Education running on a battery on a sunny day to do this! You can log data by pressing the button, or by using the timed data mode. Upload the data to the PC. (Note: if the light readings are 255, that means it is out of range, too bright. Put a piece of two of tissue paper over the sensor, held with a rubber band, and try again.) Take the raw value of light shown on screen in the second column. Suppose it is 188, and multiply it times 256, and divide by 200. This is the new value for `lical`. For example, 2000*256/188=2720.

```
lical   con 272                    ' calibration constant for photodiode
```

and

```
log(ptr)=light/10 max 255          ' ❒ store PAR/10
```

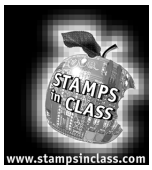
and

```
light=log(n+1)*10                  ' ❒ get PAR
```

Also, change the units of measurement in the debug statement from lux, to PAR. The reading now when the logger in full sun should be about 2000.

6) Use your logger to explore the temperatures and light levels in the outdoor environment!

You can set the time for automatic logging for 2 hours (7200 seconds), and come back in 18 hours to see what happened to time and temperature while you were away!



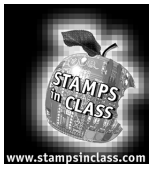
Challenge!

Do you understand energy per unit area? A certain laser puts out a total energy of one milliwatt into a beam with an cross sectional area of one square millimeter. How does the intensity of that light compare with the intensity of sunlight, which is about 1000 watts per square meter?

Assignment: Pluto You are planning to visit the planet Pluto, and you want to know how bright the light will be there. Guess--would it be enough to read this page comfortably? (a) Estimate Pluto's daytime illumination in lux. (the earth is 149,500,000 kilometers from the sun, while Pluto averages 5,920,000,000 kilometers from the sun. On Earth, with the sensor pointed directly at the sun, we measure about 110,000 lux. Approximately what value in lux will you measure when you point the light meter at the sun from Pluto? (b) Using your calibrated BS2 light meter, find a place in your environment where the ambient light level falling on this page would be comparable to what you will experience outdoors on the sunny side of Pluto.

Reaction Time Tester Install a light emitting diode and 470 ohm resistor on your Board of Education so that a `HIGH 9` instruction can turn it on, and `LOW 9` can turn it off. Write a program that does the following to test your reaction time. When you press and hold down the pushbutton, the program waits a random amount of time from 1 to 15 seconds, and then turns on the LED. Then you then have to release the pushbutton as fast as you can. The program should use the `RCtime` instruction to measure the time it takes to release the button. Then it displays your reaction time in milliseconds on the debug screen, turns off the LED, and goes back to the top to await another round. The program should test to see if you released the button before the LED goes on, and call you a "cheater" if you do.

Colorimeter In your kit you have a red and a green light emitting diode. Not only can these diodes emit light, they can also act as photodiodes to receive light. That is, the reverse current in the LED is proportional to light level hitting it. They respond best at the same color they emit. So a red LED responds most to red light, and green to green. Hook up the red and the green LEDs as shown in Figure 4.5 except use BASIC Stamp I/O pins P8 and P9, and use 100pf capacitors. The current produced by the LEDs is very small. You may reverse the position of the diodes and the capacitors to get more sensitivity if needed. Write a program that reads the output of both sensors in succession and displays the result on the debug screen. With the sensors in bright white light, the two readings will be different, because two diodes will have different natural sensitivities. Adjust the amount of light reaching the diodes, or adjust the scaling in the program, so that both readings are the same in white light. Then try putting red and green filters in front of the diodes. Have the diodes look at different colors of paper or through different filters, or at the light from a prism.



Experiment #5: The Liquid Environment

The theme of the Liquid Environment experiment is "level and conductivity of water as examples of sensors of the liquid environment." This is a more difficult kind of sensor, but we'll continue with the data logging experiments. The activities associated with this experiment consist of: (1) Conductivity using ON-OFF input or `Rctime`, and its shortcomings; (2a) Add a 555 oscillator as an input to the Board of Education and experiment; (2b) Use the 555 oscillator for measurement of conductivity in water using stainless steel probes; and (3) Add a conductivity measurement to the data logger from Experiment #3.

5

Introduction

The view of Earth from the moon in 1969 cleared up for once and for all that we live on a water planet. Scientists, farmers, emergency response agencies, the general public, everybody needs to know some "how, when, or why" about water. When is it going to rain? How deep it is. How cold, how hot, how clear, how clean. What minerals, what organic materials it contains. How fast it moves. How much is underground? How long has it been there? How much water is in the ice caps, the oceans, the rivers, in living tissue? How do raindrops form? What makes an El Niño event? What happens inside a cloud when it snows? Is there danger of landslides, droughts, floods, or famines? Can a cactus survive here, a frog, a mouse? Can I drink it? Are our wetlands disappearing? Should we care?

So, water will be the third variable we get to in Earth Measurements. What can you measure about water? I am sure that you can think of hundreds of things right off the bat. We will concentrate on a couple. The first is to detect its presence, or its level. This is the sort of measurement that is needed in order to monitor or control the level of water in a stream, or a fish farm, or a water treatment plant, or when it is a question of when to irrigate a field or a potted plant. The second type of measurement will be the electrical conductivity of water. This is a measurement that is used to detect the presence of salt and minerals in water, and it is also used to assess the quality of drinking water or to study the mixing of fresh and salt water in a tideland or an estuary. There are many kinds of water measurements that require different sensors, like how acidic it might be, or how clear it might be. It is a big field, with lots of research going into the development of sensors that can detect the quality of water.

Measurements in the liquid medium are more problematic than measurements of temperature or light. The sensor probes that detect temperature or light do not actually have to contact the medium electrically. In contrast, wetness sensors often do have to come in direct contact and are subject to corrosion and all sorts of electrical interactions with metals, ions and currents in the liquid medium itself.

Following that observation, we want to make an IMPORTANT precautionary note. Water and electricity don't mix, usually, without planning. Do not, we repeat, do not by any mistake by spilling water on your Board of Education. And always be careful about your own safety when working around electricity and water.

Parts Required

The following parts are required for this experiment:

(1) LMC555 cmos timer
(4) jumper wires
(2) 0.1 uF capacitor
(1) 100 ohm resistor
(2) 100K ohm resistor
(1) conductivity probe (couple of 2" screws 0.4" apart threaded through the green PCB cup spanner included in the kit with nylon nuts holding them on the PCB)
(1) cup
(1) water and salt

Build it!

Wetness Alarm

In Experiment #3 you experimented with the conductivity probe in your kit, as a way of introducing the `RCtime` command. Now construct the circuit shown in Figures 5.1 and 5.2.

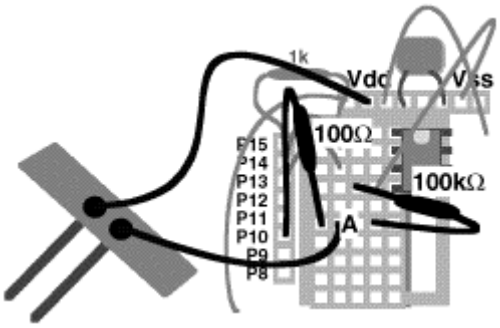


Figure 5.1 Conductivity Probe Pictorial

- 100K resistor from Vss (pin 4 of the DS1620) to node A.
- 100 ohm resistor from node A to P10.
- conductivity probe from node A to Vdd.

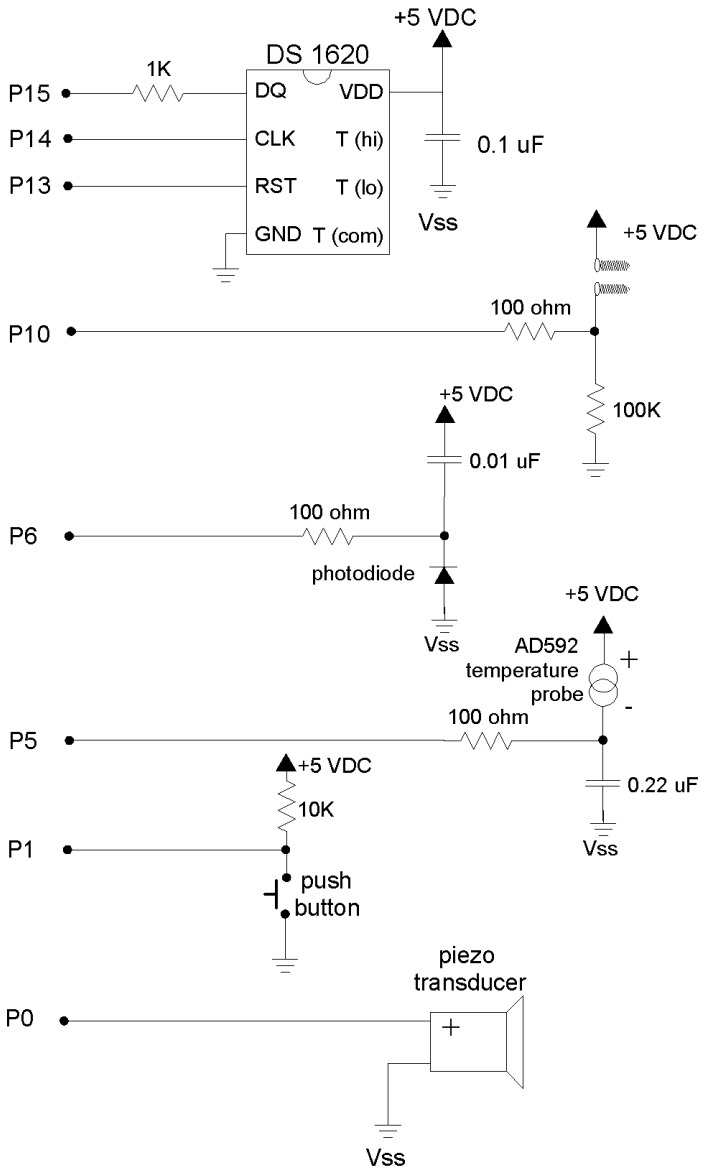


Figure 5.2: Conductivity Probe Schematic

Now enter the following program:

```
' Earth Measurements program 5.1
' wetness alarm
loop:
  debug bin in10
  if in10=0 then loop
  freqout 0,6,2550
  goto loop
```


Have a cup of water handy. This is your basic water detector and alarm. When you run the program, you will not hear anything until you dip the probe in the water. The program here is like the pushbutton routines you studied in Experiment #2, where pushing down the button made the "cricket" sound. Here, the conductivity probe in water takes on the role of the pushbutton. Recall the discussion of the conductivity probe as a variable resistor in Experiment #3. Explain what is going on by adding remarks to program 5.1.

What happens if you replace the `in10=0`, with `in10=1`? Think of a situation where that flavor of alarm might be useful.

Why is a 100K resistor chosen for the circuit? The resistor sets the sensitivity. With higher resistance values, we would run the risk of the circuit saying "wet!" even if a little condensation forms on the wiring. With lower resistance values, that type of error becomes less likely, but on the other hand, the sensor might fail to say "wet" when it should, if the water happens to be especially pure and non-conductive. It comes down to trial and error.

Try a 1K resistor in place of the 100K. (You can also borrow the 1K temporarily from pin 1 of the DS1620 temperature sensor.) You will find that you have to get the probe much wetter than before, to get the alarm. This kind of wetness alarm is used to train toddlers not to wet their bed. A pad absorbs urine and sets off the alarm. A similar circuit is used to set off the alarm in industrial plants if there is a spill.

Imagine expanding this circuit to control water level. If the water is spilled, we could turn on a pump clean it up. Then when sensor tells us the level is down, we turn the pump off. That is how a sump pump works, to keep water out of someone's basement, or a bilge pump to keep water out of a boat. But we are getting ahead of the game. That is the topic for Experiment #6. At this point the game is to look into quantitative analog measurements, not just "yes/no", but "how much water", and "water of what quality?"

Measurement of Conductance Using RCtime

Replace the 100k resistor in the above circuit with a 0.1 µF capacitor. This is now precisely the circuit of Figure 3.4. And enter the program, which is Program 3.2 revisited:

```
' Earth Measurements program 5.2
' RCtime measures conductivity.
rct  var word                          ' a word variable for RCtime
n    var byte                          ' variable for the bar graph
low 10
loop:
  RCtime 10,0,rct' time for the volts to rise to 1.3
  low 10                               ' discharge the capacitor to 0 volts
  rct=rct-1                            ' calculate length of bar graph
  debug dec rct,tab, rep "*"\ncd rct,cr ' display ascii art bar graph
  pause 1000                           ' slow it down to 1 per second.
goto loop
```

Now you have a digital output that reflects the resistance of the water between the probes. Observe how the numbers change as you change the depth of the probe in the water.

Probe Location	rct Reading
Probe not in water	
Probe tip only touching water	
Probe 1cm in water	
Probe 2cm in water	
Probe 3cm in water	

Do you see a trend? Try repeating the measurements a few times, and write down the numbers. Are the readings repeatable? That is, do you get the same result each time? Allow the probe to sit at depth for a minute or two in the water.

Here are a couple of program notes.

First, why include the formula, `rct=rct-1`? The reason is to make the graph look better. As you pull the probe out of the water, the number `rct` gets larger and larger, but suddenly when the probe leaves the water, `rct` suddenly goes back to zero. That is due to a peculiarity of the `RCtime` command, which returns the value "`0`" as a kind of error message when it runs over its maximum value. By putting in the formula `rct=rct-1`, we are in effect making "aces high".

When you subtract 1 from zero in integer arithmetic, you get 65535. When you use microcontrollers, or really, when you program any computer, you often have to compensate for the little peculiarities of the commands available to you.

How about the graph? It uses a debug modifier:

```
rep"*"\ncd rct
```


You know `rct` is the variable. `rep` is short for "repeat". It repeats printing the character "*" on the debug screen, and the number of times it repeats is given by the number after the "\|. For example:

```
debug rep "*"\12
```

would print 12 stars in a row on the screen. You could also program that as:

```
debug "************"
```

But doing it with "`rep`" is more concise. And the number after the "`\`" can be a variable, which can be very useful. Here the variable after the "`\`" is actually an expression that results in a number. The expression is "`ncd rct`". The `ncd` is a math operator unique to the BASIC Stamp. The result is just the length of the number `rct` in binary form. For example, if `rct=35` in decimal, its binary form is `rct=%100011`. The length of that binary number is 6 binary digits. Run that through the `ncd` operator, and `ncd rct` returns the value, 6, and six stars are printed on the debug screen. Maybe you will never have to use that command! But there it is, to add to your bag of programming tricks. The number of stars increases by one for each doubling of the value of `rct`.

Make one more measurement series for the above table. Reverse the connection of the conductivity probe to the Board of Education. That is, take out the probe wire that connects to node A, and connect it to Vdd instead, and take the probe wire that was in Vdd and connect it to node A instead. Make the series of measurements again in the above table. You will probably find that the numbers are a little different.

The difference is due to what is going on in the liquid medium, as electricity passes through it from one probe to the other.

The effect you are seeing is called "polarization". Chemical reactions are actually changing the electrode. This is not a big deal for the simple on-off kind of sensor, but it is a disaster for quantitative measurements. Polarization occurs because the electrical current is constantly moving in one direction through the probe. It is direct current, DC. The simple cure is to drive the sensor with current first one direction and then other. That is alternating current, AC. Many of these chemical reactions are reversible, up to a point, so the alternating current leads to much more stable readings. The BASIC Stamp can't supply the necessary AC

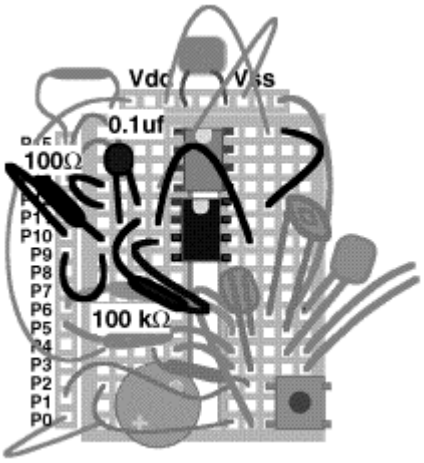
signal. An external chip can help here. It is one you are already familiar with from "What's a Microcontroller" Experiment #5, the 555 timer chip.

Measurement of Conductance Using the 555 Timer IC

Remove the parts from the above experiment, and install the 555 timer IC on the Board of Education. Be attentive since the wiring is getting tight! A pictorial for this circuit is shown in Figure 5.3, and a schematic is shown in Figure 5.4.

Figure 5.3 Conductance Using the 555 Timer IC Pictorial

- Plug the 555 timer integrated circuit into the Board of Education in the position just below the DS1620. Match the notch at the end with pin 1.
- 555 pin 8 to Vdd (pin 8 of DS1620)
- 555 pin 1 to Vss (pin 4 of DS1620)
- 555 pin 2 to 555 pin 6
- 555 pin 4 to P9
- 555 pin 3 thru 100 ohm resistor to P10
- 100K resistor between 555 pin 2 & 3
- 0.1µf capacitor between 555 pin 1 & 2



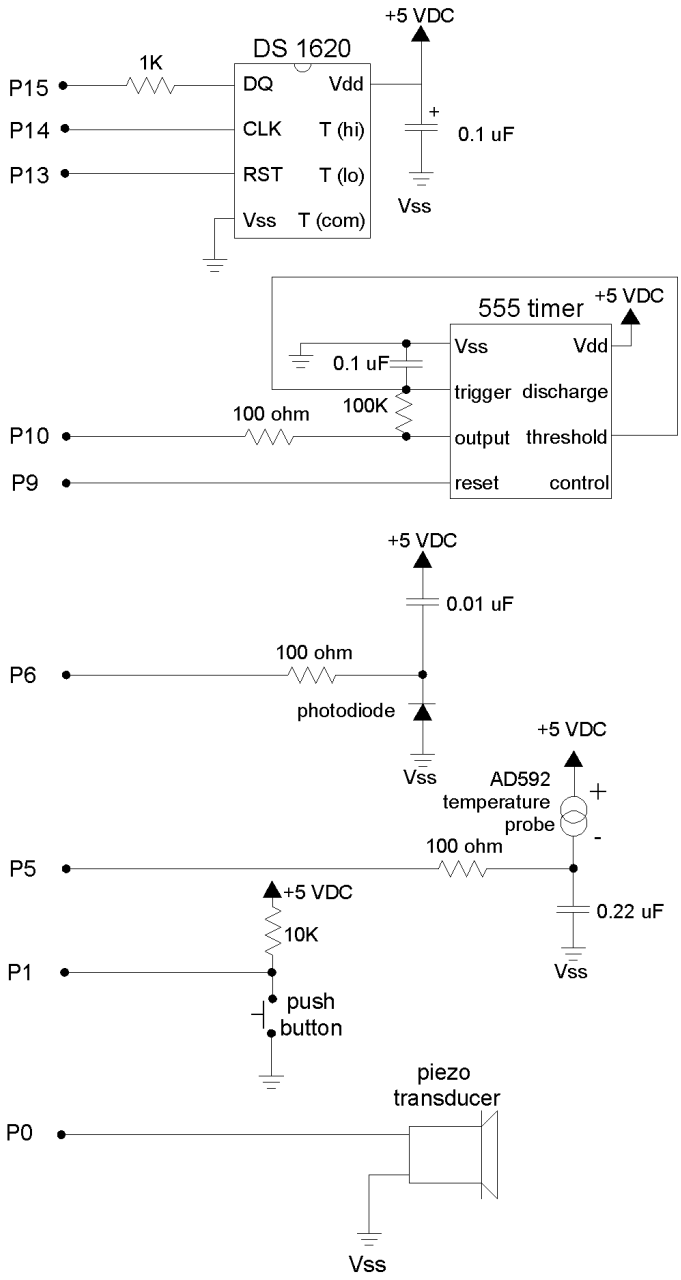


Figure 5.4 Conductance Using the 555 Timer Schematic

Space on the breadboard is becoming very limited. Follow the pictorial to make the project fit on the breadboard.

This circuit is similar to the one you built in "What's a Microcontroller?" Experiment #5. It is an astable multivibrator. That is terminology held over from the early days of electronics. All it means is that the circuit output (pin 3 of the 555) alternates from high to low repeatedly on its own. The resistor from pin 3 to pin 2, along with the capacitor from pin 2 to pin 1, determine the frequency of oscillation. P10 on the BASIC Stamp will be configured as an input so that it can monitor the frequency produced by the 555. I/O pin 9 will be configured as an output that can turn the 555 ON or OFF. When P9 is high, the 555 is ON. (If you look back at "What's a Microcontroller?", you will notice that the circuit here is different. There are several ways to hook up the 555 chip, in fact, there are entire books devoted to nothing but the 555.)

Enter and run the following program:

```
' Earth Measurements program 5.3
' Test of the 555 oscillator
cnt  var word       ' a word variable for count
high 9              ' turn on the 555 oscillation
loop:
  count 10,1000,cnt ' count for one second
  debug dec cnt,cr  ' values
goto loop
```

Here are the parameters of the BASIC Stamp II count command:

```
  count 10,1000,cnt        ' count
                ^^^-------RAM variable for the result of counting
             ^^^-----------count this long in milliseconds, duration
          ^^---------------pin to use for counting, a stamp input.
```

The reading you see on screen when the duration is 1000, should be about 75. Write down your reading.

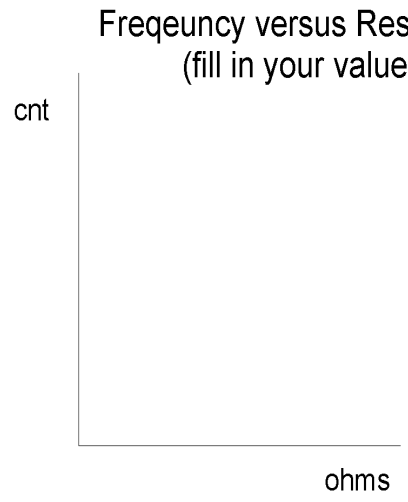
reading, cnt=?_____ when count duration is 1000, 100K resistor, 0.1uF capacitor.

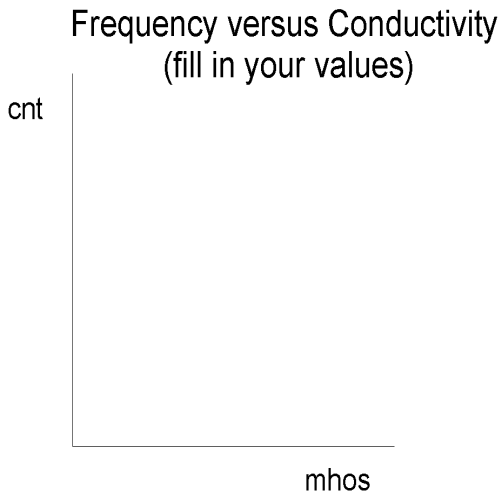
Now place a second 100K resistor in parallel with the first, side by side in the Board of Education. The parallel combination of two 100K resistors is a resistance of 50K (the series combination gives 200K). The frequency should be about double. Enter the value in the table below. Now put the two 100K resistors in series from pin 2 to pin 3 of the 555. The frequency should now by 1/2 of the original value. Also put this in the table below. Calculate the value of 1/R, which is called the conductance, and has units of siemens (an older, more colorful term for conductance is the mho, or ohm spelled backwards: 1 siemen=1 mho).

Make a quick graph of the frequency versus resistance, and a graph of frequency vs conductivity in the space provided below.

R, resistance, ohms	G, conductance, mho (=1/R)	cnt, from BASIC Stamp COUNT command
50K		
100K		
200K		

You'll need to calculate G=1/R, and measure `cnt` from Program 5.2



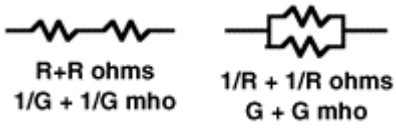


Observe which graph is more linear.

Why do we have to have to talk about both resistance and conductance, if one is just the inverse of the other? Get used to it! There is a separate term for the inverse of everything in electronics. In electronics it is more common to talk about resistance. However, in materials science, chemistry, and environmental instrumentation, it is more common to hear the term conductance. Maybe that is because in the liquid medium there are many, many different paths between any two points. The different paths are like many resistors in parallel, and changes in the liquid medium tend to change those parallel elements. So it is easier to talk about conductance, where conductances add in parallel. Figure 5.5 shows how resistors are placed in series and parallel to measure resistance (R) and conductance (G).

Figure 5.5: Resistance and Conductance Formulas

Resistance R, and conductance G, of series and parallel resistors (conductance). The formula for parallel resistors is easier in terms of conductance.



Put the single 100K resistor back in the circuit. In your Program 5.3, try changing the duration parameter from 1000 milliseconds, to 500 milliseconds, or to 2000 milliseconds. Observe that the reading changes by a factor of close to 2 each way.

Let's put this another way. Just above you wrote down a value of `cnt`, the value that came out when a 100K resistor and a 0.1µF capacitor were in the circuit, and the duration parameter was equal to 1000 in the `count` command. What duration would you have to put in the `count` command in order to make the reading come out at 100 instead of at 75 (or your reading _____)? Well, you just have to make the duration proportionately longer. A longer duration gives you a higher count, right? So calculate:

duration = 1000 * (100/75) = 1333, but you use your own reading:

duration = 1000 * (100/_____) =_____
^^^^ put your reading here

That is your duration calibration constant. You will need later.

Put that new duration value for the count command in your program in place of the 1000. Now when you run the program with 10^{-5} siemens (100K) in the circuit, it should display 100 on the screen instead of the original value.

The point of this is that the duration parameter in the count command can be used to scale the result, so that it appears directly in siemens. We want you to think quantitatively!

Now let's change the debug command so that it shows the units. And while we're at it we might as well calculate the resistance in ohms and display that too.

```
' Earth Measurements program 5.3b
' Calibration of the 555 oscillator
cnt  var word       ' a word variable for count
R    var word       ' a word variable for resistance
high 9              ' turn on the 555 oscillation
loop:
  count 10,1333,cnt ' count for about one second
          '^^^^------------- You use your own constant here!!!
  R = 50000/cnt*2   ' calculate resistance R=1/G
  debug dec cnt,"E-7",tab,dec R,"00",cr ' values
goto loop
```

Theory behind using a 555 timer to measure conductivity?

There are many references that talk about how the 555 timer circuit works, and how to apply it, even whole books on nothing but the 555. The important point for measuring conductivity is that the current through the resistor in this circuit goes back and forth, first one direction and then the other direction, equal and opposite. As we discussed above, that is what we are looking for in a probe to put in the liquid medium. There is a balance of current flow in each direction, to forestall corrosion, plating, and polarization. The theory for the 555 is very similar to the theory for RCtime, but we don't want to get into it here. The equation for the output frequency is approximately: f = 3/4*R*C. With R=100000 Ohm and C=0.1µf, that comes out to 75 Hertz.

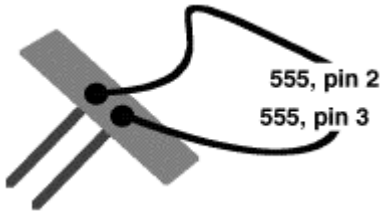
The display should now show $100E^{-7}$ (for $100*10^{-7}$ siemens), and in the second column it should show 100000 (for ohms). Verify the calibration by putting the extra 100K in parallel to get 50K back. The display should show $200E^{-7}$ siemens and 50000 ohms. Notice that the resistance reading appends two zeros to the value of R, to make it come out in ohms.

Conductance in Water

Now it is time to dive in. You will need a cup full of water, and keep a spoon and a couple of pinches of table salt handy. Replace the 100K resistor with the conductivity sensor. Leave the duration calibration factor at the value you calculated for your setup. Figure 5.6 shows how to connect the probe to the 555 timer on your Board of Education.

Figure 5.6: Conductivity Sensor

Replace the 100K resistor with the conductivity sensor.



With the conductivity sensor in the circuit, run Program 5.3b again. You should get a reading in siemens and in ohms when you put your fingers across the probe. Try wetting your fingers to see what happens to the reading. How do you explain the result in terms of conductances?

Without touching the probe, you should be able to touch the probe to the leads of the 100K resistor, to confirm that the meter still reads correctly. It should read $100E^{-7}$ siemens, 100K.

Put the sensor at 1 cm depth at the center of the cup in tap water and record the reading. Then do it again at 3, 2 and 1 cm depth. Read the conductance from the PC screen. You will have to figure out how to determine the depth in the water. Maybe you can put marks on the side of the cup, or on the sensor itself.

Distilled or Tap Water, Conductance versus Depth

Water level	Conductance
1 cm	
2 cm	
3 cm	
4 cm	

Keeping the sensor at one constant depth, move it over until it is near the side of the cup. What happens to the reading, and can you explain why in terms of conductances in parallel?

Move the sensor back to the center of the cup. Note the reading, and then bring a metal object such as the back of a metal spoon up near the sensor probe. How does that affect the reading, and why is it different from bringing the probe near the side of the cup?

Dissolve a pinch of table salt in the water in the cup. First just drop in the salt crystals, look at the reading, and then stir and look at the reading again. Write down some readings as you set the probe to different depths:

Pinch of Salt Dissolved in Water, Conductance versus Depth

Water level	Conductance
1 cm	
2 cm	
3 cm	
4 cm	

Conductivity is often used to determine the salinity of water (how much salt it contains per unit volume), or more generally, how much mineral content is present. If you used tap water, you might try the experiment again using distilled water. By holding the depth constant, you could use this probe to measure salinity, which is closely related, through a rather complicated formula, to conductivity.

Note that in each case the conductance is proportional to depth, either in the tap water, or in the salt water. You can use this device to measure the depth of water.

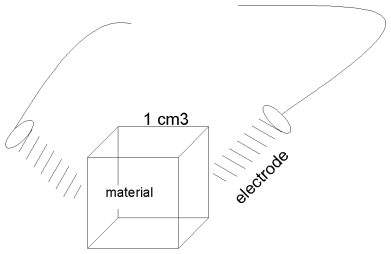
However, the two measurements are confounded. To use the device to measure depth, you have to be sure that the amount of salt in the water is going to be constant, or you have to acquire a separate measurement of conductivity in order to compensate. On the other hand, in order to measure conductance, you have to be careful to keep the sensor at a constant depth.

Design of professional instruments is largely concerned with overcoming or compensating for the effects of confounding variables. For conductivity measurements, great care is taken to confine the solution to a fixed volume, and to use stable electrode materials, and to monitor the temperature at the point of measurement. Professional water depth meters are seldom based on the conductance principle because of these difficulties.

The conductivity of water in the natural environment spans many orders of magnitude. Conductivity is measured in units of Siemens per centimeter. Ocean water may have a conductivity of 50000 Siemens per cm, whereas pure distilled water may have a conductivity of microsiemens per cm.

Figure 5.7: Conductivity Measurement

Conductivity is measured, in theory, with a block of material, 1 centimeter on a side. Electrodes are fastened to two opposite faces of the block, and the conductance is measured. Since the sample block is one cm long, the conductivity is in units of siemens per cm. Conductivity measurements are reported as if they were made in this setup, but the actual measuring setup is a lot more complicated!



What's...

The difference between conductance and conductivity?

Conductivity is a property of materials. Materials that conduct electricity well, like metals, have a high conductivity, while insulators have low conductivity. If you take a thin wire one meter long, it will have a certain resistance from end to end, and its conductance will be simply one over the resistance. A thicker wire made of the same material and of the same length will have a lower resistance and a higher conductance. The conductivity of the wire is the same in both cases. It is a property of the material

To calibrate a conductivity instrument, you would need to have a standard salt solution, that lets you make the leap from conductance (measured by your Board of Education across its particular sensor) to conductivity (a property of the water being measured, independent of peculiarities of the measuring instrument). You have to find a constant, called the "cell constant" or the "instrument constant" that accounts for the actual geometry of the probe you are using. It is a constant of proportionality. We leave the calibration to the exercises that follow this experiment.

Data Logging Continued: Drying of Soil

In the natural world, the phenomenon of evaporation is very important. Water evaporates from the soil and water is also lost in the transpiration of plants. The rate of evaporation and other mechanisms of water loss depends on the solar intensity, the wind speed, the temperature, the humidity and the ground cover. You can use your Board of Education data logger to study evaporation. On a practical basis, you can use your logger to tell you when to water your houseplants or your garden!

Modify Program 4.4 as shown below. This adds conductivity to the mix. Now you will be able to store 6 readings in memory and read them out later. The total number of bytes available for logging is 18. With 3 per record, that means we are limited to 6 records total.

```
' Earth Measurements program 5.4
' temperature, light & conductivity logging in RAM
kal     con 15068           ' calibration constant for AD592 temperature.
lical   con 647             ' calibration constant for photodiode
condcal con 1333            ' ❒ calibration constant for conductance.
                                  ' Use Your Own Calibration Constants!!
                                  ' for temperature, light, and conductance.
interval con 10             ' ❒ get data once per ten seconds
                            ' you can choose whatever interval you need.

log    var byte(18)         ' 18 bytes reserved for the log file.
rct    var word             ' variable for RCtime
light  var word             ' variable light intensity
cnt    var rct              ' ❒ variable for conductivity  ALIAS of rct
TC     var word             ' for degrees Celsius from AD592
n      var byte             ' counter for the pushbutton
ptr    var byte             ' pointer to next entry in data log file

outs=%0000000001000000     ' ❒ now put in the outs and dirs statements.
     'fedcba9876543210
dirs=%1111101111111101     ' ❒ all are low outputs, except:
                           ' P6 is high output to discharge C for photodiode
                           ' ❒ P5 is low output to discharge C for AD592
                           ' P1 is input for pushbutton
                           ' ❒ P10 is input for conductivity (555)
                           ' ❒ P9 is control of 555 conductivity ON-OFF
                           ' ❒ P0 is output for piezo buzzer

debug cls,"ready to log data!",cr
freqout 0,200,2550
freqout 0,400,3400
```

```
debug "degC",tab,"lux",tab,"siemens",cr       ' ❒ display units

mainloop:                   ' ❒
  n=0                       ' ❒ initialize the time counter
ml1:                        ' ❒ loop here until button or time
  pause 1000                ' ❒ one second pacing
  if n=interval and ptr<17 then getdata   ' ❒ get data at intervals
  n=n+1                     ' ❒ count time
  if in1=1 then ml1         ' ❒ can press button to get data too.
  freqout 0,5,3400          ' ❒ tick for button down
  n=0                       ' ❒ zero the counter for button down timer
clik1:                      '
  pause 100                 ' time the button in 0.1 sec increments
  if n>12 then playback     ' jump to the playback routine after 1.2 second
  n=n+1                     ' increment time
  if in1=0 then clik1       ' loop until the button goes up, or time runs out
getdata:                    '
  if ptr>17 then honk       ' protest if the memory is full
  freqout 0,10,1900         ' sound to show we got here

  RCtime 5,0,rct            ' read the temperature probe
  low 5                     ' discharge the temperature capacitor
  TC=kal/rct*10+(kal//rct*10/rct)-273  ' calculate Celsius
  log(ptr)=TC               ' store temperature
  ptr=ptr+1                 ' point to next bin

  RCtime 6,1,rct            ' read the photodiode
  high 6                    ' discharge the photodiode capacitor
  light=65535/rct*/lical    ' calculate lux
  log(ptr)=light/2 max 255  ' store light intensity/2
  ptr=ptr+1                 ' point to next bin

  high 9                    ' ❒ turn on the 555
  pause 100                 ' ❒ delay for it to get up to speed
  count 10,condcal,cnt      ' ❒ count the frequency
                            ' ❒ <-- use your scale factor!!!
  low 9                     ' ❒ turn off the 555
  log(ptr)=cnt              ' ❒ store the conductivity
  ptr=ptr+1                 ' ❒ point to next bin

  debug dec TC,tab,dec light,tab dec cnt,"E-7",cr ' display data!
goto mainloop

playback:                             ' ❒ Show all 6 records on debug screen
  freqout 0,50,2550                   ' feedback
  freqout 0,100,3400.....................'
  debug cls,"logged data!",cr         ' message on screen
```

```
  debug "degC",tab,"Lux",tab,"mho",cr  ' ❒ print units of measurement
  for n=0 to 15 step 3                 ' ❒ will show 6 records
  TC=log(n)                            ' get temperature
  light=log(n+1)*2                     ' get light
  cnt=log(n+2).........................' ❒ get conductivity
  debug dec TC,tab,dec light,tab,dec cnt,cr         ' ❒ display
  next.................................' next record
pb1:                                   ' finished showing
  if in1=0 then pb1                    ' wait for the pushbutton to go up
  debug cr,"press RESET to erase data",cr  'print message
  goto mainloop                        ' back to the top

honk:                                  ' come here if memory is full
  debug cr,"memory full"               ' message
  freqout 0,50,3400                    ' noise
  freqout 0,200,2000,2100              '
goto mainloop                          ' back to the top
```

Notes about this program:

Recall from Experiment #4 that we ran out of variables for that program. We used 18 bytes for the data log file, and the rest of the available variables for the program. In this program we reuse the variable `rct` for the `count` function for conductivity. We call it `cnt`, and define it in a variable alias statement at the top of the program. That means the `cnt` and `rct` are really the same physical variable. Changing one of the variables changes them both, simply because they are physically the same.

There is nothing unusual about this program. It is a straightforward expansion of the one from Experiment #4. It seems to be getting longer, but each piece has its special part to play.

In this program the `outs` and `dirs` statements are modified to take account of the new pins. P10 is an input for the `count` operations. P9 is an output to turn the 555 ON and OFF.

The new quantity `interval` is a constant for the number of seconds between readings (0-65535).

The new program has the code necessary for the conductance probe.

We have to modify the `playback` routine to play back the conductivity data.

Get this program running, and tested at 10 second intervals. Then put the temperature probe and the wetness probe in a cup or flower pot full of vermiculite or other potting medium. Put it in the sun, with the light sensor set for outdoor light. Leave it for 6 hours. Look at your data. Is the time interval appropriate?

If you are doing this in a classroom, different groups can do the experiment with variations. For example, some in the sun, some in the shade, some with ventilation, some not. Use a real potted plant. Experiment – that's the way you learn how to use microcontrollers in Earth Science!

Alternatively, for a quicker experiment, drape a wet paper towel over the conductivity sensor probe. Follow and log the conductivity and temperature of the paper towel as it dries out.

Additional Experiments to Try

1) Condensation Sensor

Press a piece of plastic or glass up against the screws on the condensation sensor. When the glass is dry, it is an insulator, and the conductivity is low. But if you breathe heavily on the glass, it will deposit condensation that will conduct electricity. Depending on the temperature and humidity, you may have to chill the surface before condensation will form. This kind of sensor is useful in agriculture, where condensation forming on leaves of plants can lead to infection by fungus and scab diseases.

2) Humidity Sensor Experiment

Find a length of heavy thread or light-weight cotton twine. Soak it in salt water (a pinch of salt will do) and then wrap it around the stainless steel screws of the conductivity probe. Dry it off with a hair dryer and observe the conductivity while you do this. Then let it come back to atmospheric moisture. If you breath on it, the conductivity will increase. The NaCl has a transition point at about 75% humidity where it picks up lots of water. Below 75% humidity, NaCl tends to give up water to the atmosphere. Above 75% humidity, NaCl tends to absorb moisture from the atmosphere. The conductivity follows along. Different salts respond at different humidity levels.

3) Surface explorer:

a) Create a shallow pool of water in a large shallow plastic, glass or paper dish. Carefully put some pieces of rock salt in the dish. Use the probe to explore the diffusion of salt into your "aquifer". Incursion of salt water into fresh water marshes and aquifers is a big problem in areas where the fresh water is drawn off for uses in industry and agriculture. b) Scribble in an area on a piece of paper with heavy pencil or artist's charcoal. Drag the conductivity probe across the surface, to explore the thickness and resistivity of the pattern.

4) Temperature Dependence of Conductivity

The conductivity of salt solutions in water depends on the type of salt, and also on temperature. Dissolve some table salt in water, and log both temperature and conductance as you heat the water. Remember that the sensor has to be immersed at a constant depth. Graph the result of your experiment. Try the same experiment with a different salt (say KOH), or with acids (Vinegar). You will find that each solution has its own characteristic temperature dependence. Commercial conductivity sensors always measure both temperature and conductivity. From temperature and conductivity, they can then calculate the concentration of the salt. Can you figure out how to do that calculation? Chemistry reference books, like the Handbook of Chemistry and Physics, contain this kind of information. Look and see. You have to know in advance what type of salt or salt mixture is in solution.

5) Quantitative Calibration of the Conductivity Probe, Using a Standard Solution.

In order to calibrate this sensor to measure conductivity (property of the material) instead of conductance (an electrical quantity), you would have to prepare a standard solution that has a known conductivity. Such standard solutions can be purchased, or you can make them yourself in the classroom, by adding a known amount of potassium chloride (KCl) to a known amount of water. Tables of conductivity are found in chemistry or in water quality handbooks, or in references such as the Handbook of Chemistry and Physics. Once you have the standard solution, you measure its conductance with your Board of Education instrument. That gives you a constant of proportionality between your conductance reading and the conductivity of the solution. This constant is called the instrument constant. It has to do with the geometry of the electrodes and the cup. You also measure the temperature of the solution. With this information in hand you can proceed to measure the conductivity (and the concentration) of unknown water samples.

6) Ground Loop Error

Connect a long piece of hookup wire to Vss (zero volts) on the Board of Education. Drop the free bare end of the wire into the cup where the conductance probe is operating and showing its readings on screen. You will see that the reading changes. This is due to the extra ground path provided by the wire. This kind of situation is common in large instrumentation systems, say, in a fish farm or in an industrial plant. But it is sometimes hard to track down where the interaction is coming from. There can be unplanned connections between points in the system. To avoid this problem, engineers often design "isolated" sensors, which means that signals are passed across an optical link or other barrier of that sort, so that there will be no direct electrical connection. This is also extremely important in situations where safety and shock hazard are an issue, such as in medical instrumentation.

7) Environmental Explorer

Measure the conductivity of some samples of water. Try distilled water, tap water, ocean water. How much salt would you have to add to one liter of distilled water to make it as salty as the ocean?

Challenge!

1) Write a program that counts the number of times you press the pushbutton in 5 seconds. The BASIC Stamp should sound a "start" tone, and then count the number of button presses, play a "finish" tone, display the result on screen, pause for 3 seconds, and then do it again.

2) Install a red and a green led on the Board of Education, so that P7 and P8 can turn them on or off. Modify the program that measures temperature, light and conductivity as follows. It should turn on the green LED if the measurements are all within normal operating range (you decide what that range is). If they go outside the normal range, the green LED should turn off, and the red LED should turn on. If they then return to normal, the green LED should come back on, but the red light should stay on to show that there has been a "problem". If the readings get way out of range, turn on the "alarm siren".

3) With the circuit of figure 5.4, the frequency of oscillation is proportional to 1/RC, where R is the resistance, and C is the capacitance. In your kit, you have 2 of the 100K resistors and 2 of the 0.1µF capacitors. You can put resistors in series to make 200K, and in parallel to make 50K. You can put two of the capacitors in series to make 0.05µF, and in parallel to make 0.2µF. Write a program to show you the frequency in hertz for each value of resistance and capacitance in the following table.

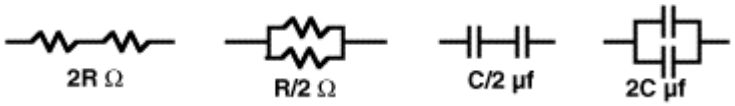


table entries are frequencies from 555 timer	0.05 uF	0.1 uF	0.2 uF
50K-20E^{-6} mho			
100K-10E^{-6} mho			
200K-5E^{-6} mho			

This will test your understanding of how the white block is connected underneath!

Does it seem to be true that frequency=constant/RC? Graph frequency vs conductivity. Graph frequency vs resistance. Which is linear?

Experiment #6: Measurement and Control

The theme of the Measurement and Control experiment is that microcontroller can do both measurement and control, closing the feedback loop. The activities associated with this experiment consist of: (1) Feedback to control of the level of water in a cup using a battery powered pump, and conductivity as the level detector. (Do not get water on your Board of Education!--Electronics in the real world!); and (2) Simultaneous measurement and control of 4 variables. The materials used in this experiment will require you to improvise with items you can find around the classroom.

Introduction

Measurement and data logging are often combined with control. Not satisfied to simply sit there and watch, the Board of Education reaches out and does something that affects the conditions in the outside world. It might open a door in response to an approaching pedestrian, as you did in "What's a Microcontroller" Experiment #4. Or it might function as a thermostat, to control a heater or a fan when the temperature gets too hot or too cold. In industry, on the farm, in public works, in scientific research, all manner of processes need to be controlled and regulated based on measurements, to achieve a desired result. Some instruments may themselves require internal measurement and control. Imagine what goes on in a machine like the automated Mars rover, where robot arms and chemistry laboratories and instruments of all kinds have to function as an integrated measurement and control system far from human interaction. Many modern instruments, like DNA analyzers or automated water quality analyzers, are marvels of measurement and control.

In this experiment, you will turn on a pump to regulate the level of water level in a cup, or to keep up the moisture in the soil around a potted plant. You might think of these as smaller versions of a fish farm, or a water treatment plant, or a full-scale irrigation system for a vineyard.

Feedback is important here. It is possible to have control without feedback. If you pour one cup of water on your potted plant every day, regardless of the condition of the plant, there is no feedback in that. You are in danger of over-watering the plant, and wasting water and fertilizer besides. It might not matter with one small plant in well-drained soil, but imagine a dry-land farm, or a large greenhouse operation. If the condition of the soil or the plant is first measured, and from that decision is made of whether or not to irrigate, that is feedback at work. The result can be a happier plant as well as more efficient use of resources. This is especially important in places and times where water is scarce. Feedback can take on many forms, and involve a combination of measurements in the control decisions.

The final project in this Earth Measurements series will be a data logger that combines the two temperature sensors (the DS1620 and the AD592), the light sensor, the conductivity sensor, and the pump control in one program. The data will be stored in the BASIC Stamp's EEPROM memory. This data logger can be used for a variety of experiments you may think of to undertake as class projects or on your own initiative. Thank you for sticking with it! Do good for the earth!

Parts Required

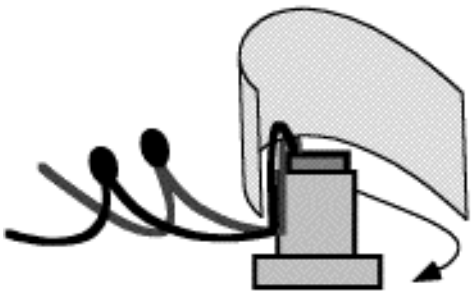
The following parts are required for this experiment:

(1) ZTX1049A NPN "superbeta" transistor (marked ZTX 104 9A)
(1) 100 ohm resistor
(1) 10 ohm, 1 watt resistor (brown-black-black) heavier construction (this resistor is manufactured out of special temperature resistant materials)
(1) pump (Edmund Scientific) with 1/4" tubing
(2) 12" red and black wires to extend from the pump's smaller wires
(1) cup and tray, cup prepared with a 1/4" hole punched or drilled near the bottom hookup wire (cup and tray not included in the Earth Measurements Parts Kit)

Build it!

Figure 6.1 shows the preparation of the pump. The lead wires from the pump are fragile. The pump *cannot tolerate water inside the motor*. We suggest that you take a piece of duct tape or wide plastic tape and wrap it around the plastic pump housing to secure the wires. Pinch the tape together at the top to protect the pump from splashed water. Extra heavy wires are provided to extend the fine wires.

Figure 6.1: Pump Preparation
Wrap a piece of tape around the top of the pump to protect the wires and prevent water from getting into the pump. Join and tape the pump's wires with the long red and black wires that are included in the kit.

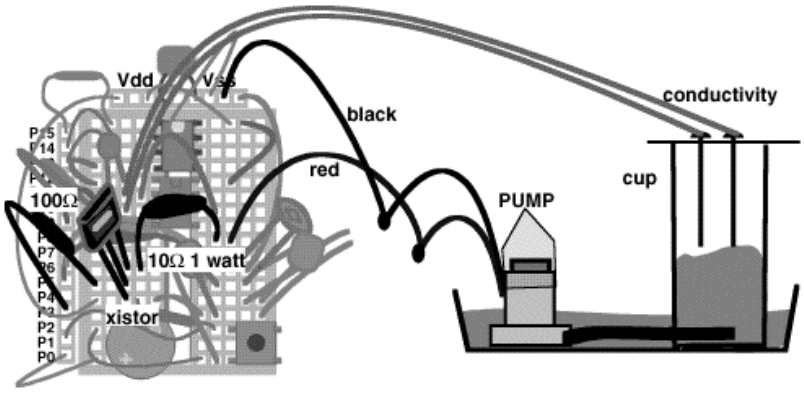


On-Off Control of Pump

Add the transistor circuit to the Board of Education, and connect the pump as follows so that the PBASIC program will be able to turn it on and off. The diagram in Figure 6.2 shows the conductivity sensor in place on top of the cup, but for the time being put the sensor to one side. We will return to it in a moment.

Figure 6.2: Pump Control with Transistor

- transistor collector to Vdd node
- transistor base through a 100 ohm resistor to P3
- transistor emitter to the 10 ohm , 1 watt resistor
- 10 ohm 1 watt resistor the red pump wire
- black pump wire to Vss



The three pins on the transistor are the collector, the base and the emitter, as shown above. The orientation is keyed to the printing on one face of the transistor. Put the pump in shallow water, enough to cover the impeller unit 2 cm deep. A complete schematic for this project is shown in Figure 6.3.

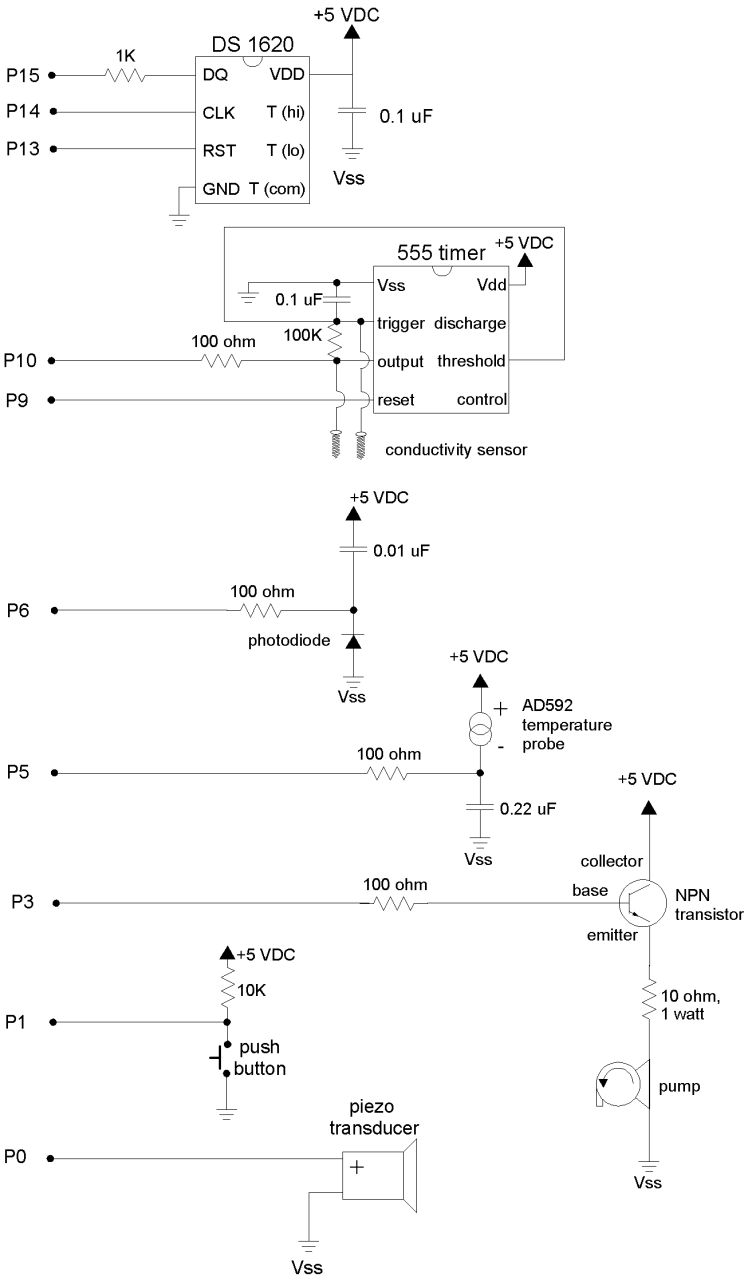


Figure 6.3: Pump Control with Transistor Schematic

Enter the following program:

```
' Earth Measurements program 6.1
' pump tester
loop:
  high 3
  pause 5000
  low 3
  pause 2000
 goto loop
```


The pump should turn on for 5 seconds and off for 2 seconds, and repeat. By the way, do not use a 9-volt battery for this-the pump draws too much power. Use the 300 mA 9 volt power supply that came with the Board of Education.

Troubleshooting. If it works, great! If not, here are a couple of suggestions.

- Kick the pump. Not literally. But sometimes the **impeller** becomes stuck if it has dryed out in a way that leaves mineral deposits inside. Look at the bottom of the pump, and you will see a hole and the vanes of the impeller inside. You can loosen them with the tip of a paper clip. Also, look to be sure that the tube that emerges from the side of the pump housing is not pushed in too far. If it is pushed into far it can prevent the impeller from turning.

- Use a wire to jump directly from the collector to the emitter of the transistor on the Board of Education. That bypasses the program control of the pump. If the pump still does not operate, then disconnect the pump from the Board of Education and touch the pump wires to a 1.5 volt flashlight battery. If that doesn't do it, the pump itself may be defective, or the wires that connect to the pump may be broken.

- If the pump does work, but the circuit does not respond to the program, recheck the wiring. Be sure that the transistor is oriented correctly in the circuit and that the base is connected to P3. The collector should be in the +5 volt node next to the piezo buzzer. And one end of the 10 ohm resistor should be in the same row as the emitter of the transistor.

- The wiring is getting tight. You have to check the connections carefully to be sure there is not a short circuit with one of the other parts on the Board of Education.

What's a...

Impeller:
Inside the plastic housing at the bottom of the motor is a rapidly rotating disk with radial vanes. The vanes pull in water at the center through a hole you can see in the bottom of the housing, and throw the water out along the edge and force it to flow through the exit tube. The water is pulled through by the action of centrifugal force. The spinning disk with vanes is called an impeller.

Figure 6.4 shows an impeller, the rotating disk within the pump.

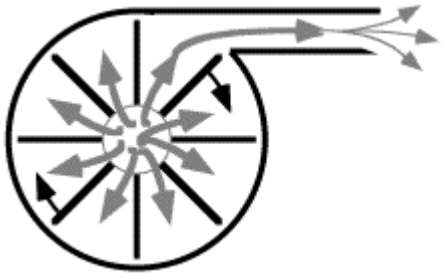


Figure 6.4: Pump Impeller

The motor in the pump draws about 300 milliamps of current at 3 volts. That is lots more than the pins of the BASIC Stamp are capable of supplying. It is necessary to use a transistor to amplify the current available from the BASIC Stamp. There are several ways to use transistors. The one here is called an "emitter follower". The voltage at the emitter of the transistor, "follows" the voltage at the base. When P3 is `low`, at zero volts, then the emitter of the transistor is also at zero volts and the motor is `off`. But when P3 goes `high`, to +5 volts, then the emitter follows along (4.4 volts when P3 is at +5 volts) and the pump is turned on. The 300 milliamps needed by the pump comes through the transistor from the power supply, not from pin P3. Only a tiny current of about 0.3 milliamps is necessary to turn on the much larger 300 milliamp current. The transistor is acting as an amplifier for current. The 10 ohm, 1 watt resistor limits the voltage applied to the pump. The pump is only rated to operate at up to 3 volts. Figure 6.5 shows the transistor action

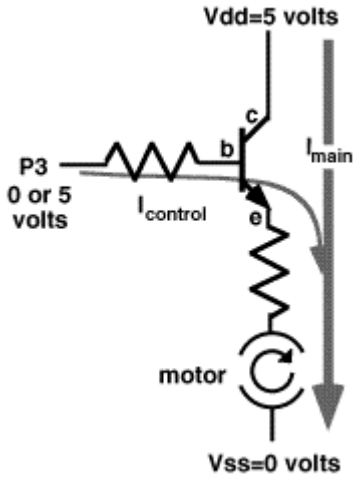


Figure 6.5: Transistor Action
e = emitter
b = base
c = collector

Now modify the program as follows so that the pump is ON only when you hold the pushbutton down.

```
' Earth Measurements program 6.1
' pump tester
output 3
loop:
  out3=~in1
goto loop
```

Maybe you were expecting a longer program? Look at the statement, `out3=~in1`. What that says is, "make the output the opposite of the state of the pushbutton input." The symbol "~" means "not". If `in1` is zero, pushbutton down, then the state of the pump will become 1, ON. If `in1` is equal to 1, pushbutton up, then the state of the pump will become 0, OFF.

Try it. Mark a level on the side of the cup, and hold down the button until the water level reaches your mark. Then release the button. Try to press and release the button so that you hold the water level close to the mark. You are now part of a feedback loop. And soon you are going to be replaced by automation. The conductivity sensor is going to do the looking and turning the pump on and off. I think this is one job you would just as soon have automated.

A further word about the program. Another way to write it would be to use IF-THEN statements, something like this:

```
' Earth Measurements program 6.1b
' pump tester
loop:
  if in1=0 then pumpOn
  if in1=1 then pumpOff
goto loop
pumpOn:
  high 3
goto loop
pumpOff:
  low 3
goto loop
```

That is a fine way to write the program. It shows very well what is going on. If the pushbutton is down, the pump turns ON. If the pushbutton is up, the pump turns OFF. The program has to take one or the other course of action, because `in1` has to be either 0 or 1. Try it.

The program could alternatively be written using the branch instruction:

```
' Earth Measurements program 6.1c
' pump tester
loop:
  branch in1,[pumpOn, pumpOff]
pumpOn:
  high 3
goto loop
pumpOff:
  low 3
goto loop
```

See the BASIC Stamp Manual Version 1.9 (p. 247) for information about the `branch` command. The program branches to `pumpOn` if the pushbutton is down (in1=0) or to `pumpOff` if the pushbutton is up (in1=1). That too is a clear way to write the program. It will branch to one or the other, because in1 can only take on the values 0 or 1.

Try the different ways to write the code. It is always good to know that there are different ways of accomplishing a task. The goal may be to make the code as compact as possible, or to make it run as fast as possible, or to make the program as easy as possible to follow in documentation.

Pump Control with Feedback

Arrange the tray and the cup so that the outlet tube from the pump enters the cup through a hole near the bottom of the cup as shown in Figure 6.2. The hole needs to be 1/4 inch in diameter to hold the tube firmly without leakage. Put the conductivity sensor in the container, so that the tip of the sensor is above the resting water level. The objective now is to operate the pump until the water comes up to the level of the probe, and then to hold the level there, automatically, using the conductivity probe as the level sensor. In the above exercise you were using your eye as the level sensor and your finger on the pushbutton as part of the feedback loop.


```
' Earth Measurements program 6.2
' pump controller
cnt     var word
loop:
  high 9                        ' turn ON the 555
  count 10,133,cnt              ' count pulses
          '^^----------------- ' USE YOUR CONSTANT divided by 10
  low 9                         ' turn OFF the 555
  debug dec cnt," umho",cr      ' display micromho
  if cnt > 36 then Poff         ' level too high, turn pump off
  if cnt < 30 then Pon          ' level too low, turn pump on
  goto loop                     ' do it again
Poff:                           ' turn pump off
  low 3
  goto loop
Pon:                            ' turn pump on
  high 3
  goto loop
```

Use your own calibration constant (divided by 10 here) that you determined in Experiment #5. The calibration constant makes the reading come out in units of micro-mho (or micro-siemens, µS, in cgs units). What does that mean? If you take the reading from the debug screen into 106, that will be the resistance in ohms. It is not so important here to have real units for control of the level, but on general principles we like to remind you that there are accepted units of measurement for conductance.

The water should rise up to the level of the sensor, and then the pump should turn off. The water level drops until it no longer hits the sensor, and then the pump turns on. Remember, the count is higher as the probe is deeper in the water.

Observe the action. How often does the pump come on? What is the ratio of the ON time to the OFF time? This isn't a very efficient system, because the water leaks back out through the pump when the pump is off.

The critical level chosen for the water, the point where the pump changes from OFF to ON and vice versa, is called the "set point". There are two set points in this program, one for ON and one for OFF. The pump turns ON when the level is less than 30, and it turns OFF when the level is greater than 36. Think about what happens in the program code when the level is between 30 and 36. Neither of the IF statements is true, so the program goes around the loop without taking any action to change the state of the pump motor. If the motor was OFF, it stays OFF. If the motor was ON, it stays ON. This gives 8 units of hysteresis. This is a desirable feature in some kinds of feedback systems. For example, there are some kinds of motors and equipment that suffer if they are cycled on and off too often. It is better to let the liquid reach the upper set point, and then let the motor rest until the liquid drop below the lower set point before turning the motor back on again. That saves wear and tear on mechanical parts. Other times it is a requirement of the system, say, to let a plant dry out between watering, or to create a surge in the water level in a fountain. This kind of control, where the two set points are offset like that, is called hysteresis. This program has 7 units of hysteresis, from 30 to 36 inclusive.

The vertical axis is the water level, or the conductance, since higher water level --> higher conductance. The horizontal axis is the state of the pump, ON or OFF. On the left, the water is low, and the pump is on. On the right, the water is high, and the pump is off. In operation, the system spends most of its time going around the square of hysteresis. ON up to the upper set point, then OFF, and falling down to the lower set point.

Figure 6.6: Hysteresis

Experiment with the value of the high and the low set points. Increase the upper set point to make the water reach a high level on the conductivity sensor, without overflowing the cup. Observe the value of `cnt` on the debug screen. Observe how often the pump cycles on and off, when the set points are close together, versus when they are farther apart.

What will happen to the actual level if the conductivity of the water changes? Try adding a pinch of salt to the water and see. (This is not a professional level sensor!)

Here is an alternative way to program it that avoids the use of the `if. . then` statements. This is a matter of programming style. Try it!

```
' Earth Measurements program 6.2b
' pump controller
cnt     var  word               ' variable for count
low 3
loop:
  high 9                        ' turn on the 555
  count 10,200,cnt              ' count pulses
  low 9                         ' turn off the 555
  debug dec cnt,cr
  out3=~(cnt/36 max 1)
  goto loop                     ' do it again
```


When the value of the `count` is less than 36, the set point, then the value of cnt/36 will be zero. Remember, this in integer math, and the result of cnt/36 is always an integer. When `cnt` is greater than or equal to 36, then the value of cnt/36 will be 1 or greater. The additional operation, `max 1`, limits the value to 1 at most. There is a "not" operator, "~" in front of the whole expression in parentheses. The result is that when `cnt` is less than the set point, `out3` is `HIGH`, and the pump is ON. But when `cnt` is greater than or equal to the set point, then `out3` is `LOW` and the pump is OFF. Note that this program starts off with a command, `low 3`, that turns P3 into a `LOW` output. Otherwise P3 would be an input.

This program 6.2b does not have any hysteresis. The pump is ON for all values of the count less than 36, and OFF for all values greater than or equal to 36. Here is a statement that adds a little bit of hysteresis:

```
out3=~(cnt/(30+(out3*6)) max 1)
```

The pump stays ON until the water level reaches 36 or above, but once the pump is OFF, it does not turn back ON until the level falls back below 30. The hysteresis is added here by mixing the state of the output into the right-hand side of the formula. The BASIC Stamp can do that. `Out3` is a variable like any other variable, and your program can either read or set its value. Think this through. It is tricky and not as transparent as doing it with `if` and `goto`. But this way of coding it is more compact. `If` and `goto` clutter up a program in their own way. It is an advanced technique for your bag of programming tricks!

Memory in the BASIC Stamp, Revisited

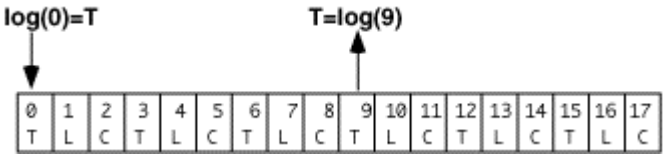
The data logger we developed in Experiments #4 and 5 had only 18 bytes of RAM memory. We could only have 9 records with 2 fields in Experiment #4, or 6 records with 3 fields in Experiment #5. Not only that, the data all disappeared if we turned off the power or pressed the reset button; is gone for good! This would be a serious shortcoming in a data logger for use in environmental science. A scientist or an engineer may want to collect

much more data than that, and he or she most likely won't want the data to disappear prematurely. It needs to be retrieved and stored safely in an archive file before it is erased from the logger.

So we are going to switch over and store the data in the EEPROM memory instead of in the RAM. Recall from Experiment #2 that there are 2048 bytes of EEPROM on the BASIC Stamp. We will reserve 250 bytes of that for our data log. That is lots more than we had available in RAM. We can reserve more than that, too, if need be for future experiments. Best of all, our data in EEPROM will survive resets and power outages.

The way we go about storing data is different in RAM that it will be in EEPROM. Here is the way it looks in RAM:

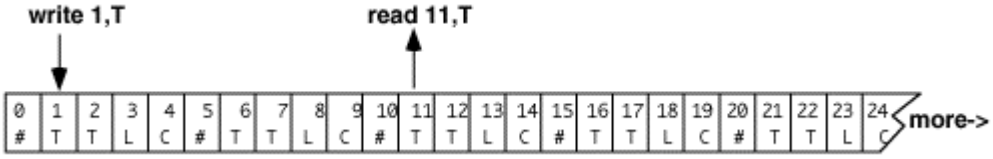
Figure 6.7: RAM log



There is an array of 18 byte-size variables, from log(0) to log(17). We put data into bins with statement of the form, log(i)=TC, and we retrieve data from the bins with statements of the form, TC=log(i). The value in parentheses is a variable, a pointer, as was done in Programs 4.4 and 5.4.

Here is the difference with EEPROM. Recall Experiment #2:

Figure 6.8: EEPROM log



To write a temperature value into location 1 in the EEPROM, we use the form

```
write 1,TC
```

and to retrieve data from location 11 in EEPROM, we use the form:

```
read 11,TC
```

`TC` is a byte-size quantity. As in RAM, we can use a pointer variable to reference the byte to write or to read. The following program demonstrates a couple of ways to reserve space in the EEPROM.

Try it:

```
' Earth Measurements program 6.3
' illustration of eeprom allocation
pad     data (32)               ' reserve 32 bytes, undefined
log     data 1(60)              ' reserve 60 bytes, preset all = 1
hey     data 72,101,121,33,32,66,83,50  ' reserve 8 bytes specified
ptr    var byte                 ' byte for pointer
x      var byte                 ' byte for eeprom data
for ptr=0 to 7                  ' point to 8 locations in a row
  read ptr+hey,x                ' read data from location
  debug dec x, " "              ' show it, with a space
next
debug cr
```


When you run this, (or press reset on the Board of Education), you should see the 8 numbers from the "hey" data appear on the debug screen.

This program sets aside space in EEPROM for data, 100 bytes of it to be exact. There are 32 bytes of undefined data (not preset to any particular value) starting at address zero, then 60 bytes of defined data (all preset to one) starting at address 32, and 8 bytes of numerical data at locations 92 through 99. Each of these 100 EEPROM locations contains an 8 bit data pattern. The 8 bit pattern could represent a number such as a temperature or a light level, or it could represent a letter to print on screen, or it could be a pattern of Morse code, or any other thing you could imagine to fit into a pattern of digital bits.

The `for . . next` loop reads and prints out the 8 numbers starting at EEPROM address "hey" . We could have written it like this:

```
for ptr=92 to 99        ' <-- explicit values for the pointer
  read ptr,x            ' <-- read from those locations
' and so on
```

But it is best to let the BASIC Stamp software keep track of the details of which number goes with which name. That makes future changes easier.

Now make a simple change to the `debug` statement, as follows:

```
for ptr=0 to 7
  read ptr+hey,x
  debug x               ' ❒ <-- change this, leave out dec and ," "
' and so on
```

Now when you run the program, the BASIC Stamp reads the same 8 bytes from the EEPROM memory. But the debug statement sends them as is as single bytes to the PC screen. The screen interprets them as printable characters. For example "72" sent as a single byte is the ASCII code for the letter "H" (ASCII-American Standard Code for Information Interchange). With the dec modifier, the debug statement takes the single byte with numerical value 72, and sends it to the screen as two ASCII codes, first the one for "7" and then the one for "2". If you substitute one of the other numerical modifiers, you can see the numbers in binary (72=%1001000) or in hex (72=$48):

```
debug x                 ' show as ascii text
debug dec x," "         ' show decimal, with a space
debug bin x," "         ' show binary, with a space
debug hex x," "         ' show hex, with a space
```

Try it! The point is that the binary pattern that is stored in the EEPROM is the same in each case. It is only the interpretation by the debug command and by the PC screen that is different. Please bear with us if you already know this. This is a confusing point for many students. We are going to use the EEPROM to store numerical data, but we usually use debug with the decimal modifier.

We will store each point of data as one byte in the EEPROM. Each byte can represent a number from 0 to 255 decimal. Our logger will not store larger values. It is possible to do so, but it would take two EEPROM locations per point.

The declarations:

```
pad     data (32)       ' reserve 32 bytes, undefined
log     data 1(60)      ' reserve 64 bytes, preset all =1
```

are two other ways to reserve space for data in the EEPROM. The second form initializes the 60 bytes to have a value of 1, while the first form simply sets aside the bytes without specifying a value to store there.

Now modify the central part of Program 6.3 once more as follows, to print out the decimal value of all 100 locations in EEPROM:

```
for ptr=0 to 99         ' ❒  read in all 100 bytes of data
  read ptr+pad,x        ' ❒  starting at pad (pad=0)
  debug dec x," "       ' ❒  show data as decimal value
next
```

Now you should see 32 bytes of garbage, followed by 64 zeros, followed by the 8 bytes that have special meaning as ASCII text. Why do we say "garbage"? It is because the 32 bytes you see first are simply stuff that was left over in your BASIC Stamp from earlier programs and experiments. The program reserves space, but it does not send any new data to the BASIC Stamp to put in those locations.

The BASIC Stamp II editor has a very useful feature that lets you look directly at the allocation of the memory. It is an invaluable tool for program development. Close the debug window if it is active on your screen. If you are using the DOS editor, STAMP2.EXE, press ALT-M. If you are using the Windows version, STAMP2W.EXE, press CTRL-M (or choose memory map from the menu or from the tool bar). What you see will depend on which version of the editor you are using, but in both cases there are three views. In the Windows version, all three views appear on one screen, and they are labeled "RAM Map", "Condensed EEPROM Map", and "Detailed EEPROM Map". In the DOS version, the three views appear on three different screens, and you cycle through the views by pressing the space bar.

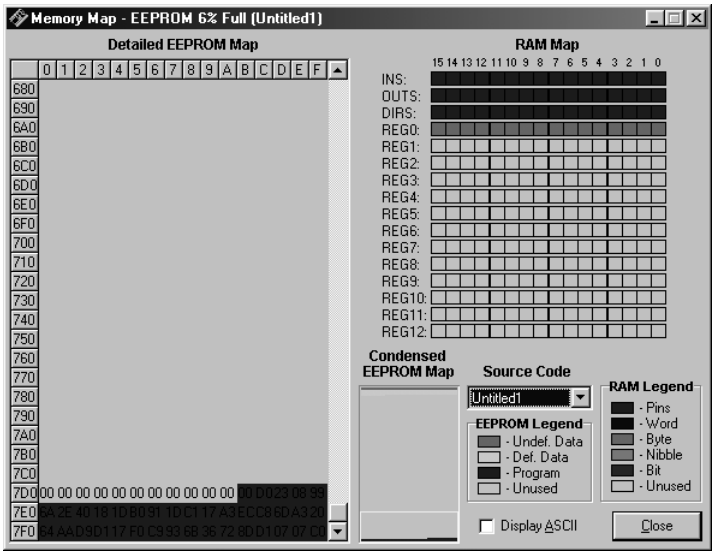


Figure 6.9: Memory Map.
The left portion of the map is the EEPROM, your BASIC Stamp source code and extra EEPROM space. The right side of the map is variable storage, RAM. The lower right hand corner of the memory map defines the type of data by color that you are storing in EEPROM and RAM.

Look at the "RAM Map". It is the first screen you see in the DOS version, or, in the Windows version, it is the view at the upper right side. Recall that RAM is Random Access Memory that is located inside the PIC microcontroller chip, and stores the program variables. There 32 bytes, (16 words, 256 bits) altogether. The first 6 bytes (3 words) are dedicated to the i/o pins of the BASIC Stamp. These variables bear the pre-assigned names, ins, outs and dirs. On the "RAM map" these pin variables appear in red. That leaves 13 words, 26 bytes, for the variables in our program. Program 6.3 has only two variables, both of them defined as bytes. The "RAM map" shows them in light blue color right below the pin variables. The rest of the RAM memory is not allocated in this program, and is shown in white (gray in Windows). Be aware that the "RAM Map" does not show you the actual values of the variables--that only happens when you run the program.

Now look at the "Condensed EEPROM Map". It is in the lower middle of the Windows version screen, but in the DOS version, you have to press the space bar once to bring up this view. Recall that the EEPROM memory is located in a separate chip, apart from the PIC microprocessor. There are 2048 bytes of EEPROM. At the top of the schema is the data in shades of blue, and at the bottom is the program code in red. Between the program and the data is empty space that will fill up as we write longer programs and reserve more space for data. You might ask what will happen if the two areas collide in the middle? That's easy, you will get an "out of memory" error message. Notice that the data area has two different shadings. The first 32 bytes in blue are the "undefined data", or "empty data" declared with the statement:

```
pad     data (32)
```

When you load (run) your program into the BASIC Stamp, the loading process does not touch those bytes, and that was the "garbage" you saw when you last ran Program 6.3. In contrast, the following statements create what is called "defined data".

```
log       data 1(60)
hey       data  72,101,121,33,32,66,83,50
```

When you run your program into the BASIC Stamp, specific bytes are loaded into the EEPROM along with the program itself.

Now look at the "Detailed EEPROM Map". If you are using the DOS version, press the space bar again to get to this view. This view shows the contents of the EEPROM byte by byte. You first see 32 dashes (DOS version) or 32 dark blue zeros (Windows version), followed by 60 zeros, followed by the 8 specific bytes. The display shows HEX numbers (from 00 to FF). In the DOS version, you can see the ASCII text (when meaningful!) to the right of the map, and in the Windows version, you can press ALT-A to view the data as ASCII text.

Note that the BASIC Stamp editor does not show you the "garbage". You only get to see that when you really run the program on the BASIC Stamp. You see, the BASIC Stamp gives you a lot of options on how to use the EEPROM resources.

Now use the scroll bar, or the arrow keys (ALT-arrow in the Windows version) to move down to the bottom of the "Detailed EEPROM Map". When you are looking at the bottom of the memory map, you will see the actual bytes of the program itself as it is stored in the EEPROM. Program 6.3 occupies about 34 bytes of memory. The program code is stored in a very compressed form, so don't look for an easy correspondence between the bytes in the EEPROM and the text of the program.

Press ESCAPE on the keyboard (DOS) or close the window (ALT-C Windows), to get back to the editor screen. The purpose of this digression is to help you think about the organization of the memory on the BASIC Stamp, and also to illustrate a very useful feature of the BASIC Stamp programming software.

Data Logger

Now it's time to get down to business. There are several issues that need to be addressed to make a working data logger. Rather than deal with them piecemeal, here altogether now are the design objectives.

- The logger will also function as a pump controller, to keep the water level up in the cup. So the logger has to do both measurement and control.

- Clicking the pushbutton once will log 5 bytes in EEPROM as follows below.

ordinal 1,2,3,. . . 50	temperature from DS1620	temperature from AD592	light from photodiode	conductance from probe	→ more

- There will also be a capability for timed data logging, so the logger can take readings at a programmed interval from seconds to hours. For example, with hourly logging, and 50 records total, the unit could hold two days worth of data. The interval is set at the time of programming the BASIC Stamp.

- Since we are going to set aside 250 bytes for the data log, and since each record has five fields, there will be room for 50 records in the file. The log file can be made smaller or larger as needed for different projects.

- The program can find its place in the data file even after a RESET or a power outage. This is done by scanning the data file, where the next free data location will be tagged by a zero in the ordinal number field.

- Press and hold down the pushbutton for 1.2 seconds to get into the routine to play back all the recorded data. This is like the RAM data logger in Experiments #4 and 5. After playback of the data, the logger can resume taking additional data where it left off.

- To erase the data and start over, press and hold down the pushbutton during RESET.

- Annunciate all the user interaction on the piezo transducer. Show data on the debug screen. Morse Code is optional.

The starting point for this program is Program 3.4, which you should have saved on disk. This program also uses code from Program 5.4. If you have the Windows version of the BASIC Stamp software, then you can save yourself a little typing by cutting and pasting. The program is getting long, but we want to emphasize that it is built out of lots of pieces that you already know. This program just brings them together in one place. The objective here is to give you a program you can use for further experiments in Earth Measurements.

Enter this program and get it running. One strategy is to simply type in all the changes, and then deal with any typos and errors later. That isn't a bad strategy where you have reason to believe that the program is basically okay (and we hope that it is!). Another strategy is to type small segments and test as you go along. That is usually the best way if you are uncertain about the pieces. First verify your Program 3.4, then add the additional variables and constants and data declarations, then the light and conductivity sensors, then the pump control routines, then the routine put data in memory, and then the timed data logging, and finally the routine to read the data out from memory.

```
' Earth Measurements program 6.4
' data logger for 2 temperatures, light and conductance
' with simultaneous control of water level

' morse code constants and variables
dit    con   50                  ' milliseconds for Morse dit
dit2   con   2*dit               ' constants related to dit
dah    con   3*dit               ' ditto
mc     var   byte                ' temporary for Morse pattern
j      var   nib                 ' index for digits to send
i      var   nib                 ' index for dits and dahs

' general purpose variables
xm     var   byte                '  morse & eeprom input variable
x      var   byte                ' general purpose variable
n      var  word                 ' variable for time counter

                                 ' note: DS1620 preprogrammed for mode 2.
                                 ' high 13:shiftout 15,14,lsbfirst,[12,2]:low 13

' sensor calibration constants. USE YOUR OWN calibration constants
Kal    con  16428                ' for the AD592 in Kelvin with 0.22uF
lical    con  647                ' for the photodiode in lux with .01uf
cntcal   con  1333/10            ' for conductance in umho with 0.1uf.
```

```
' sensor variables
degC   var  word                ' for Celsius temperature from DS1620
TK     var  word                ' for Kelvin temperature from AD592
TC     var  word                ' Celsius from AD592
rct    var  word                ' for the RC timer.
light  var  word                ' light level from the photodiode
cnt    var  word                ' for the conductance probe
umho   var  byte                ' conductivity
mhomax var  byte                ' maximum value of conductivity

' logging constants & variables
interval con  600               ' interval for logging in tenths seconds
nflds  con  5                   ' number of fields per record
nrecs  con  50                  ' number of records in file
logsiz con  nflds*nrecs         ' size of the log file in bytes
pad    data (16)                ' padding to save wear and tear on memory
log    data 0(logsiz)           ' bytes reserved in eeprom for data log file
ptr    var  byte                ' pointer to data in the log file
                                ' routine also uses variable xm

outs=%0000000001000000          ' now specify the initial outs and dirs.
     'fedcba9876543210
dirs=%1111101111111101          ' P0 is output for piezo
                                ' P1 is input for pushbutton
                                ' P3 low for pump
                                ' P5 is low output to discharge C for AD592
                                ' P6 is high output to discharge C for photodiode


                                ' P9 is control of 555 conductivity ON-OFF
                                ' P10 is input for conductivity (555)
                                ' P13-15 output for DS1620 SPI
                                ' all unused pins are low outputs

top:                            ' program starts here out of reset or erase

  ptr=-5                        ' pointer = -5 in prep for findptr routine
findptr:                        ' find next free location in eeprom
  ptr=ptr+5                     ' point to a record
  read ptr+log,x                ' read byte
  if x>0 AND ptr<logsiz then findptr'  if x not zero, this is not a free record
                                ' also test for full log, ptr=logsiz
                                ' pass here when free record is found
                                ' ptr points to the next free record

If in1=0 then eraselog          ' erase data log if pushbutton down at RESET
debug ? ptr                     '  show the pointer & the base address
```

```
debug "RESET+button=erase",cr     '  user message
freqout 0,20,1900                 '  beep to signal that it is running

mainloop:                         ' main program
  n=0                             ' initialize the time counter
clik:                             ' loop here until button or time
  if n=interval then getdata      ' get data at intervals
  gosub pump                      ' update the pump status
  n=n+1                           ' count time
  if in1=1 then clik              ' can press button to get data too.
  freqout 0,5,2550                ' tick for button down
  n=0                             ' zero the counter for button down timer
clik1:                            ' gosub pump
                                  ' time the button in 0.1 sec increments
  if n>12 then playback           ' jump to the playback routine after 1.2 second
  n=n+1                           ' increment time
  if in1=0 then clik1             ' loop until the button goes up, or time runs out
                                  '  pass here if button clicked

getdata:                          ' come here to scan and log data, timed or button
freqout 0,20,3400                 ' got here!
low 3                             ' turn off pump, unconditional, while scanning
xm=ptr/5+1                        ' first put the ordinal record number in memory
gosub writedata                   ' write it in eeprom!
debug dec xm," "                  ' print it on screen

DS1620:                           ' DS1620 temperature sensor code
  high 13                         '  select the DS1620
    shiftout 15,14,lsbfirst,[238]'  send the "start conversions" command
  low 13                          '  finish the command
  pause 450                       '  delay for conversion
  high 13                         '  select the DS1620
    shiftout 15,14,lsbfirst,[170]'  send the "get data" command
    shiftin 15,14,lsbpre,[x]      '  get the data
  low 13                          '  end the command
  degC=x/2                        ' convert the data to degrees C
  xm=degC                         ' morse routine expects data in variable, xm
  gosub writedata                 '  and write the degC data
  debug dec xm,tab                '  show it on debug screen
  'gosub morse                    '  and send it as morse code (optional)

AD592:                            ' AD592 temperature sensor code
  rctime 5,0,rct                  ' get the AD592 count
  low 5                           ' pull input low, discharge cap
  TK = Kal/rct*10 + (Kal//rct*10/rct)
                                  ' calculate Kelvin
  TC = TK-273                     ' and convert to degrees C
```

```
  xm=TC                          ' morse routine expects data in variable, xm
  gosub writedata          '  write the data to eeprom
  debug dec xm,tab         '  show it on debug screen
  ' gosub morse            '  and send as morse code (optional)

Photodiode:
  RCtime 6,1,rct           '  read the photodiode
  high 6                   '  discharge the photodiode capacitor
  light=65535/rct*/lical   '  calculate lux
  xm=light/2 max 255       '  ready to store it in eeprom
  gosub writedata          '  store it in eeprom!
  debug dec light,tab      '  show it on debug screen

Conductance:
  xm=mhomax                ' store the maximum value from pump routine
  gosub writedata          ' data to eeprom
  debug dec xm,cr          ' show max conductance in umho
  mhomax=0                 ' reinitialize the accumulator

goto mainloop              ' back to await time or button
end                        ' end of main program

'------- more goto routines --------

eraselog:
  freqout 0,400,2550,1900  ' sound we got here
  for x=0 to ptr step 5    ' step through all used record numbers
  write x+log,0            ' make them zero
  next
  debug cls,"data erased!",cr    ' message on cleared screen
el1:                       ' hold here until button released
  if in1=0 then el1
goto top

playback:
  low 3                    ' pump off unconditional
  freqout 0,50,2550        ' sound we got here!
  freqout 0,100,3400
  debug cls,"logged data!",cr    ' message and units
  debug "#",32,"degC",tab,"degC",tab,"lux",tab,"umho",cr
  ptr=0                    ' point to start of data
pb0:
  read ptr+log,x           ' read record number
  if x=0 then pb1          ' if it is zero, this is an empty record
  debug dec x," "          ' show record number
  read ptr+1+log,degC      ' read temperature (DS1620)
  read ptr+2+log,TC        ' read temperature (AD592)
```

```
  read ptr+3+log,light     ' read light
  read ptr+4+log,umho      ' read conductance
  light=light*2            ' restore light units
  debug dec degC,tab,dec TC,tab,dec light,tab,dec umho,cr
  ptr=ptr+5                ' point to next record
  goto pb0                 ' loop up
pb1:                       ' wait for button up
 if in1=0 then pb1
  debug rep "-"\31,cr      ' horizontal line
goto mainloop              ' go back to collect more data

' ------ subroutines --------

morse:                     ' enter here to send byte xm as morse code
  for j=1 to 0             ' send 2 digits, tens then ones.
    mc = xm dig j          ' pick off the (j+1)th digit
    mc = %11110000011111 >> mc          ' set up pattern for morse code
    for i=4 to 0           ' 5 dits and dahs
      freqout 0,dit2*mc.bit0(i)+dit,1900      ' send pattern from bits of mc
      pause dit            ' short silence
    next                   ' next i,dit or dah of five
    pause dah              ' interdigit silence
  next                     ' next j,digit of two
return                     ' back to main

Pump:
  high 9                   '  turn on the 555
  count 10,100,cnt         '  count the frequency
  low 9                    '  turn off the 555
  umho=cnt*cntcal/100 max 255    ' calculate umho
  mhomax=umho min mhomax   ' keep the maximum value of umho
  if umho>99 then pumpoff  ' threshold to turn pump off
  if umho<50 then pumpon   ' threshold to turn pump on
return                     ' get here if umho is betwee thresholds
pumpon:
  high 3                   ' turn it on
return
pumpoff:
  low 3                    ' turn it off
return
  '  out3=~(umho/(out3*49+50) max 1)    ' control the pump,  alternate

writedata:
  if ptr>=logsiz then writeout          ' check for end of log
  write ptr+log,xm         ' write this field
  ptr=ptr+1                ' point to next field
writeout:
```

```
return
```

Once you get the program running, you can try a short logging experiment to be sure that it works. It should work pretty much like the RAM data logger you made in Experiments #4 and 5. Check the design objectives above for the details about how it is supposed to work. The logging interval is initially set to be about 1 minute (interval con 600) in tenths of a second. If you are working in a classroom setting, your teacher may have other suggestions for the interval and for the size and location of the log file.

Program Notes and Troubleshooting

- Weird behavior? If the program resets frequently, and seems like it never quite gets started, try to run it without the pump plugged in.

- Hot Board of Education? The pump draws quite a bit of electrical power. The 10 ohm resistor that is in series with the motor on the Board of Education will get warm, as will the voltage regulator that powers the whole Board of Education. Expect the temperature of the DS1620 temperature sensor to rise when the BOE operates the pump for a long time. Touch them carefully and see.

- Dead DS1620? If DS1620 stops responding (you see only zeros in the second column on the debug screen), increase the delay in the DS1620 routine from 450 to a larger value. A delay is required after sending to the DS1620 the [238] code that starts the analog to digital conversion. If you compare carefully you will see that in program 3.4 that "start conversions" code was only sent once, early in the program. Unfortunately, the DS1620 is quite sensitive to noise generated by the pump. As a quick fix, we turn off the motor, and then issue the "start conversions" command. In a real engineering project, this behavior would be troublesome, and effort would be expended to isolate and resolve the problem.

- Calibration recall? Remember if you change the timing capacitors, you also have to recheck the calibration. Be sure you have the 0.22 µF capacitor for the AD592 temperature sensor, and the 0.01µF for the photodiode, and 0.1 for the conductance sensor. You may, for example, want to use the light sensor in outdoor sunlight, so you will have to change over to a 0.22µf capacitor and also change the calibration constant.

The constants are:

`Kal`	from Experiment #3	value= _____
`lical`	from Experiment #4	value= _____ indoor
`lical`	from Experiment #4	value= _____ outdoor
`cntcal`	from Experiment #5	value= _____

- Time critical multitasking? Note that the pump subroutine at the end of the listing is called in a couple of places. In particular, it is called repeatedly in the initial pushbutton up and pushbutton down loops. That is because the operation of the pump is a time-critical task. This is what many kinds of complicated programs have to do. They have to perform multiple tasks at practically the same time. Here it is to both monitor the button and keep up the water level. The programmer has to be sure that both tasks are serviced in a timely manner. The conductance reading in the pump routine takes 1/10 of a second, and paces the whole process. The BASIC stamp is a relatively slow computer, and it cannot do true "multitasking". The BASIC Stamp here is getting around to the pump and the pushbutton about 10 times per second. You can see the level in the cup drop when the program goes off to take readings from the sensors. Maybe even a one or two-second delay is acceptable here. But in other systems, it could matter a whole lot and you would want a faster microcontroller.

- Pump power can ruin the sensor readings? Note the `low 3` command near the top of the `getdata` routine. The pump is turned off unconditionally while reading the sensors. Otherwise, the noise and heavy power supply drain of the pump would affect the readings. Try it to see what we mean. Comment out the `low 3` command, and then run the program again. During an interval when the pump happens to be on, press the pushbutton and note the readings on the debug screen. Then press the pushbutton again during an interval when the pump is off, and compare those readings with it on.

- Why mhomax? The conductance reading requires some explanation. The conductance is being used to control the water level. So the pump routine reads the conductance value often as part of the pump subroutine. That routine keeps track of the maximum value of conductance that it has detected.

  ```
  mhomax=umho min mhomax
  ```

  What this says is, "let the new value of `mhomax` be equal to whichever is greater, the conductance (`umho`) or the current value of `mhomax` " (min, because `mhomax` is the minimum value in the match). For example, if the current value of umho is 67, and the current value of `mhomax` is 65, the new value of `mhomax` will be 67. It is `mhomax` that goes into the data file in the conductance routine. `mhomax` is then reset to zero so that it can accumulate a new and different maximum value during the next interval.

- 5 and next free? The `writedata` subroutine is called from several places in the `getdata` routine. First it is called to store the ordinal record number, and then once for each sensor. At the end of the `getdata` routine, the pointer is left pointing to the next free byte, where the next record number will go the next time the interval passes or the next time the pushbutton is pressed.

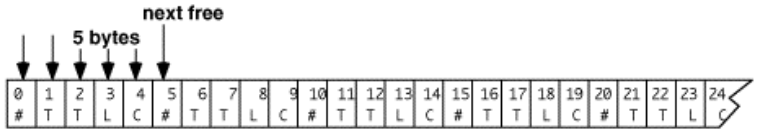


Figure 5.10: Writing Data

In a professional data loggers, the ordinal number field might be used to store the time and date (Y2K compliant, of course!).

- Find and seek? The `findptr` routine scans through all of the possible records in the data file, looking only at the locations reserved for record numbers: (ptr=0, 5, 10, 15, ... ,245). If it finds a zero in one of those locations, then that is the next free location where the next record number will go. This scan takes place each time you press RESET on the Board of Education, or when the power is turned on. By putting tags in the data file, it can reconstruct where it was. Note that right after that is the instruction that tests to see if the pushbutton is being held down, just after RESET. If so the BASIC Stamp branches to the routine that erases the data file. What it really does is to put zeros in all of the ordinal number locations. That way the `findptr` routine starts over at the beginning. The `erasedata` routine does not really erase all the data. Just that one field.

- Playback? The routine that plays back the data is the reverse of the routine that puts it into the memory. `rval` passes or the next time the pushbutton is pressed.

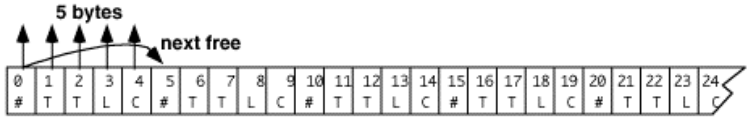


Figure 5.11: Writing Data

The routine reads out 5 locations, and then leapfrogs the pointer up to the next group of five. At the end, the pointer is left pointing to the next free location, one containing a zero in the ordinal number field.

- Push button? The code that detects when you press and release the pushbutton should be very familiar to you by now. Here every time around the loop, while waiting for something to happen, the progam checks the pump and turns if off or on as needed. The routine to read the conductivity takes about 0.1 second, so that paces the button loops too.

- Time out? The interval for logging is not very precise. You may want to calibrate it by changing the ballpark constants.

  10 seconds
  1 minute
  10 minutes
  18000 30 minutes
  36000 1 hour

  To calibrate the timing, you need a stop watch. Set the interval for something like 1 minute, and time it (using the beeps for logging) to see how it turns out with interval = 6000. If the actual time turns out to be 1% too long, then to compensate, count for 1% less time, or interval = 5940.

- Ruin the EEPROM? Note that 16 bytes are set aside at the top of the memory. The purpose of this is to avoid overstressing those top bytes. Recall that EEPROM has a finite life in terms of the number of times it can be written. It take a million or so rewrites. When using EEPROM for data logging, you have to remember this. It takes a long time to get to a million times, but don't get carried away with rapid fire experiments with sub second timing!

- What is in memory? While in the PC editor, press ALT-M (DOS) or CTRL-M (Windows). Recall the discussion of the "RAM Map" and the "EEPROM Map". This helps to visualize how the variables and EEPROM space are being used. What fraction of the EEPROM are the program and the data occupying?

Other Investigations

1) Study the flow rate from the pump, and the height of water it is capable of supporting.

2) Make a solar heater by putting black copper tubing under glass or in a bottle, out in the sun. Run the pump to bring water into a reservoir. Monitor the water temperature and the sunlight. Use the water temperature and the sunlight to decide when to turn on and off the pump automatically.

3) Try to PWM the pump, that is, turn it rapidly on and off as follows:

```
x con 5
loop:
high 3
pause x
low 3
pause 10-x
goto loop
```

The constant, 5 causes the pump to be on half the time and off half the time. So the pump effectively pumps half as fast. The on and off alternate so fast that you can't perceive a stop and start in the pumping action. Try varying the constant to see what happens at other ratios of ON to OFF time. You can also try the BASIC Stamp II built-in PWM command.

The data logger can be used in classroom experiments. Modify the program, so it will do what you want!

Challenge!

1) Write a simple program that turns on the pump when the pushbutton is pressed once, and then it stays on until the pushbutton is pressed again. (Push on, push off).

2) You are in charge of a public fountain that is supposed to operate only in the daytime, and only when the sun is out and the temperature is greater than 70 degrees Fahrenheit. Write a program to control the fountain.

3) Make a program that can draw a graph of conductance on the debug screen. Let the program fill the cup to overflowing, and then let the level drop below the sensor tips, before starting up the pump again. All the while graphing the conductivity reading on the debug screen.

4) A fish farm must keep the level of water up in a tank and replenished, and keep the water stirred, and monitor for conditions that could be deadly. Write a program that Keeps the water going up and down in the cup, but it will sound an alarm if the temperature of the water exceeds 80 degrees C, or if the water stops flowing for any reason, or if the conductivity of the water changes drastically.

5) Sometimes in real-world settings it is desirable to know how long or what percentage of time a motor is running versus off. This helps with maintenance and with planning for energy efficiency. Modify program 6.2 so that it displays the percentage of time that the pump stays on. You could also add this to your data logging program, as an indication of how much water was used.

Parts Listing

All components (next page) used in the Earth Measurements experiments are readily available from common electronic suppliers. Customers who would like to purchase a complete kit may also do so through Parallax. Parallax adds a small packaging and handling fee to the parts, partially offset by our volume purchases made to the suppliers. Customers may realize small savings of 10% on low volumes (~10 units) of the "Earth Measurements" Parts Kit by building their own component kits, but in higher volumes the savings from the Parallax kit is more substantial.

Each experiment requires the Board of Education - Full Kit (#28102):

Parallax also manufactures the Board of Education Kit (#28150), consisting of the Board of Education and pluggable wires only. Use the Board of Education Kit if you already have a BS2-IC module and power supply. Individual pieces may also be ordered using the Parallax stock codes shown below.

Board of Education – Full Kit (#28102)

Parallax Code#	Description	Quantity
28150	Board of Education	1
800-00016	Pluggable wires	10
BS2-IC	BASIC Stamp II module	1
750-00008	300 mA 9 VDC power supply	1
800-00003	Serial cable	1

Board of Education Kit (#28150)

Parallax Code#	Description	Quantity
28102	Board of Education and pluggable wires	1
BS2-IC	Pluggable wires	10

This printed documentation is very useful for additional background information:

BASIC Stamp Documentation

Parallax Code#	Description	Internet Availability?
27919	BASIC Stamp Manual Version 1.9	http://www.stampsinclass.com
28125	Earth Measurements Text	http://www.stampsinclass.com
27951	"Programming and Customizing the BASIC Stamp Computer"	Table of Contents only from http://www.stampsinclass.com

The Earth Measurements experiments require the Earth Measurements Parts Kit (#28126)

The contents of the Earth Measurements Parts Kit is listed below. These parts are required for building the entire project. In case you need specific replacement parts from Parallax the stock code is listed for each individual component. If you would rather purchase these components elsewhere and need assistance identifying an appropriate source for these parts, please feel free to contact us at stampsinclass@parallaxinc.com.

Earth Measurements Parts Kit (#28126)

Parallax Code#	Description	Quantity
150-01011	100 ohm ¼ watt 5% resistor	4
150-01020	1K ¼ watt 5% resistor	1
150-01030	10K ¼ watt 5% resistor	1
150-01040	100K ¼ watt 5% resistor	2
150-01000	10 ohm 1W resistor metal oxide	1
150-04710	470 ohm 1/4 watt 5% resistor	2
200-01010	100 pF mono radial capacitor	2
200-01040	0.1 uF mono radial capacitor	3
200-01031	0.01 uF 50V capacitor	1
200-02240	0.22 uF 50V capacitor	3
350-00012	Photodiode, blue enhanced (Photonic Detectors)	1
350-00001	LED, green	1
350-00006	LED, red	1
500-00004	2 A high gain transistor (Zetex1049A)	1
400-00001	Pushbutton	1
604-00002	DS1620 Digital Thermometer (Dallas Semiconductor)	1
604-00010	555 timer, 8-pin DIP	1
700-00020	2", 4-40 stainless steel machine screws	2
700-00036	4-40 nylon nut	2
700-00019	cable ties	2
800-00016	3" jumper wires (bag of 10)	2
800-00021	16" red jumper wire	1
800-00022	16" black jumper wire	1
900-00001	Piezospeaker (sound transducer)	1
28130	Temperature probe (Analog Devices 592 soldered to two foot-long wires, heat-shrinked and glued for protection)	1
700-00018	Pump (Edmund Scientific X50-345)	1

No stock code	Cup spanner made of plastic or printed circuit board, with two 1/8" holes for screws set 0.4" apart	1
By customer	Cup and tray	1

Sources

The Parallax distributor network serves approximately 40 countries world-wide. A portion of these distributors are also Parallax-authorized "Stamps in Class" distributors – qualified educational suppliers. Stamps in Class distributors normally stock the Board of Education (#28102 and #28150) and sometimes the Earth Measurements Parts Kit (#28126). Several electronic component companies are also listed for customers who wish to assemble their own Earth Measurements Parts Kit.

Country	Company	Notes
United States	**Parallax, Inc.** 3805 Atherton Road, Suite 102 Rocklin, CA 95765 USA (916) 624-8333, fax (916) 624-8003 http://www.stampsinclass.com http://www.parallaxinc.com	Parallax and Stamps in Class source. Manufacturer of the BASIC Stamp.
United States	**Digi-Key Corporation** 701 Brooks Avenue South Thief River Falls, MN 66701 (800) 344-4539, fax (218) 681-3380 http://www.digi-key.com	Source for electronic components. Parallax distributor. May stock Board of Education. Excellent source for components.
United States	**Mouser Electronics** 345 South Main Mansfield, TX 76203 (800) 346-6873, fax (817) 483-6899 http://www.mouser.com	Source for electronic components. Parallax distributor. May stock Board of Education in 1999. Excellent source for components.
Australia	**Microzed Computers** PO Box 634 Armidale 2350 Australia Phone +612-67-722-777, fax +61-67-728-987 http://www.microzed.com.au	Parallax distributor. Stamps in Class distributor. Excellent technical support.

Australia	**RTN** 35 Woolart Street Strathmore 3041 Australia phone / fax +613 9338-3306 http://people.enternet.com.au/~nollet	Parallax and Stamps in Class distributor.
Canada	**Aerosystems** 3538 Ashby St-Laurent, QUE H4R 2C1 Canada (514) 336-9426, fax (514) 336-4383	Parallax distributor and Stamps in Class distributor.
Canada	**HVW Technologies** 300-8120 Beddington Blvd NW, #473 Calgary, AB T3K 2A8 Canada (403) 730-8603, fax (403) 730-8903 http://www.hvwtech.com	Parallax distributor and Stamps in Class distributor.
Germany	**Elektronikladen** W. Mellies Str. 88 32758 Detmold Germany 49-5232-8171, fax 49-5232-86197 http://www.elektronikladen.de	Parallax distributor and Stamps in Class distributor.
New Zealand	**Trade Tech** Auckland Head Office, P.O. Box 31-041 Milford, Auckland 9 New Zealand +64-9-4782323, fax 64-9-4784811 http://www.tradetech.com	Parallax distributor and Stamps in Class distributor.
United Kingdom	**Milford Instruments** Milford House 120 High St., S. Milford Leeds YKS LS25 5AQ United Kingdom +44-1-977-683-665 fax +44-1-977-681-465 http://www.milinst.demon.co.uk	Parallax distributor and Stamps in Class distributor.

Boooks and Internet Resources

If you are new to BASIC Stamps, electronics, or programming there are several internet and printed sources you may wish to investigate.

Books and Publications

Programming & Customizing the Basic Stamp Computer by Scott Edwards. ISBN 0-07-913684-2. Available from Parallax (#27905) and Amazon (http://www.amazon.com).

Parallax BASIC Stamp Manual Version 1.9 from Parallax (#27919) and distributors.

Nuts and Volts Magazine Stamp Applications. Published each month in Nuts and Volts magazine (http://www.nutsvolts.com), with past issues available for free download from their web site.

Getting Started in Electronics by Forrest M. Mimms. Available at Radio Shack stores.

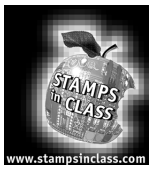
Internet

Parallax web site http://www.parallaxinc.com and the Parallax Stamps in Class web site http://www.stampsinclass.com include free downloadable BASIC Stamp resources.

Tracy Allen, Ph.D. web site at http://www.emesystems.com. Dr. Allen wrote the Earth Measurements series and uses the BS2SX-IC in his commercially available dataloggers.

Al Williams Consulting hosts the BASIC Stamp Project of the Month at http://www.al-williams.com.

List of Stamp Applications from http://www.hth.com provides over 150 projects using the BASIC Stamp.



Building the AD592 Temperature Probe

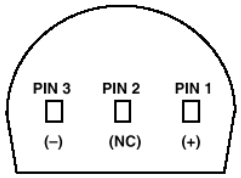
The Earth Measurements experiments use the AD 592 temperature transducer. The part needs to be protected before being inserted into liquid. Parallax builds a custom temperature probe (#28130), but you can do this yourself from these plans. An abbreviated datasheet for the AD 592 is included in Appendix D of this text.

You'll need the following materials:

(1) AD592 Temperature Transducer in plastic TO-92 case (Newark Electronics)
(2) 16" wires, one black and one red, stripped on both ends (or eq.)
(2) 1" solder sleeves (Powell Electronics CWT-1502 or eq.)
(1) 1½" adhesive lined heat shrink tubing with ¼" inside diameter (Digi-Key #EPS3316NK-ND or eq.)
(1) heat gun

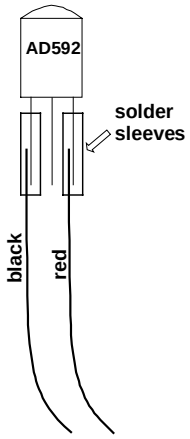
To build the probe:

1. Identify the AD 592's (-), NC, and (+) pins from this picture as viewed from the bottom.



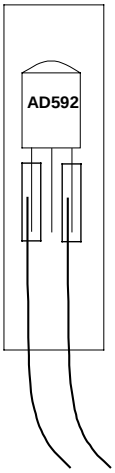
2. Slip the solder sleeve over the black wire and pin 3 (-). Slip another solder sleeve over the red wire and pin 1 (+). Heat up the connections until the wires are joined.

   If you have no solder sleeves you can use heat shrink tubing.



3. Slip the heat shrink tubing over the entire package. Fasten the package with a heat gun, and while it's still hot clamp the top portion to ensure that it stays shut.

Clamp here →



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%

Appendix C: Resistor Color Code

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 ohms (or 270k) & a tolerance of 5%.

Data Sheets

Appendix D consists of abbreviated data sheets for the key components used in these experiments. The full data sheets are available from the manufacturer's web sites shown below. This text includes the first two pages only of the data sheets.

Datasheet Locator

Component	Manufacturer's internet address	Pages on their web site
Analog Devices 592	http://www.analogdevices.com/	8
Dallas Semiconductor 1620	http://www.dalsemi.com/	12
Edmund Scientific Pump	http://www.edmundscientific.com/	1

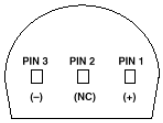
ANALOG DEVICES

Low Cost, Precision IC Temperature Transducer

AD592*

FEATURES
High Precalibrated Accuracy: 0.5°C max @ +25°C
Excellent Linearity: 0.15°C max (0°C to +70°C)
Wide Operating Temperature Range: –25°C to +105°C
Single Supply Operation: +4 V to +30 V
Excellent Repeatability and Stability
High Level Output: 1 μA/K
Two Terminal Monolithic IC: Temperature In/ Current Out
Minimal Self-Heating Errors

CONNECTION DIAGRAM



★ PIN 2 CAN BE EITHER ATTACHED OR UNCONNECTED
BOTTOM VIEW

PRODUCT DESCRIPTION

The AD592 is a two terminal monolithic integrated circuit temperature transducer that provides an output current proportional to absolute temperature. For a wide range of supply voltages the transducer acts as a high impedance temperature dependent current source of 1 μA/K. Improved design and laser wafer trimming of the IC's thin film resistors allows the AD592 to achieve absolute accuracy levels and nonlinearity errors previously unattainable at a comparable price.

The AD592 can be employed in applications between –25°C and +105°C where conventional temperature sensors (i.e., thermistor, RTD, thermocouple, diode) are currently being used. The inherent low cost of a monolithic integrated circuit in a plastic package, combined with a low total parts count in any given application, make the AD592 the most cost effective temperature transducer currently available. Expensive linearization circuitry, precision voltage references, bridge components, resistance measuring circuitry and cold junction compensation are not required with the AD592.

Typical application areas include: appliance temperature sensing, automotive temperature measurement and control, HVAC (heating/ventilating/air conditioning) system monitoring, industrial temperature control, thermocouple cold junction compensation, board-level electronics temperature diagnostics, temperature readout options in instrumentation, and temperature correction circuitry for precision electronics. Particularly useful in remote sensing applications, the AD592 is immune to voltage drops and voltage noise over long lines due to its high impedance current output. AD592s can easily be multiplexed; the signal current can be switched by a CMOS multiplexer or the supply voltage can be enabled with a tri-state logic gate.

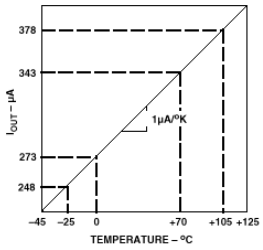
The AD592 is available in three performance grades: the AD592AN, AD592BN and AD592CN. All devices are packaged in a plastic TO-92 case rated from –45°C to +125°C. Performance is specified from –25°C to +105°C. AD592 chips are also available, contact the factory for details.

***Protected by Patent No. 4,123,698.**

REV. A

PRODUCT HIGHLIGHTS

1. With a single supply (4 V to 30 V) the AD592 offers 0.5°C temperature measurement accuracy.
2. A wide operating temperature range (–25°C to +105°C) and highly linear output make the AD592 an ideal substitute for older, more limited sensor technologies (i.e., thermistors, RTDs, diodes, thermocouples).
3. The AD592 is electrically rugged; supply irregularities and variations or reverse voltages up to 20 V will not damage the device.
4. Because the AD592 is a temperature dependent current source, it is immune to voltage noise pickup and IR drops in the signal leads when used remotely.
5. The high output impedance of the AD592 provides greater than 0.5°C/V rejection of supply voltage drift and ripple.
6. Laser wafer trimming and temperature testing insures that AD592 units are easily interchangeable.
7. Initial system accuracy will not degrade significantly over time. The AD592 has proven long term performance and repeatability advantages inherent in integrated circuit design and construction.



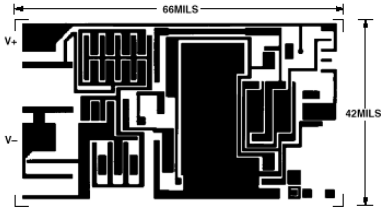
One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 617/329-4700 **Fax: 617/326-8703**

AD592—SPECIFICATIONS (typical @ T_A = +25°C, V_S = +5 V, unless otherwise noted)

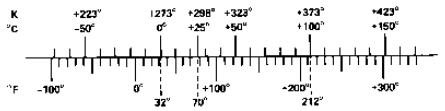
Model	AD592AN Min	AD592AN Typ	AD592AN Max	AD592BN Min	AD592BN Typ	AD592BN Max	AD592CN Min	AD592CN Typ	AD592CN Max	Units
ACCURACY										
Calibration Error @ +25°C[1]		1.5	**2.5**		0.7	**1.0**		0.3	**0.5**	°C
T_A = 0°C to +70°C										
Error over Temperature		1.8	3.0		0.8	1.5		0.4	0.8	°C
Nonlinearity[2]		0.15	0.35		0.1	0.25		0.05	0.15	°C
T_A = –25°C to +105°C										
Error over Temperature[3]		2.0	**3.5**		0.9	**2.0**		0.5	**1.0**	°C
Nonlinearity[2]		0.25	0.5		0.2	0.4		0.1	0.35	°C
OUTPUT CHARACTERISTICS										
Nominal Current Output @ +25°C (298.2K)		298.2			298.2			298.2		µA
Temperature Coefficient		1			1			1		µA/°C
Repeatability[4]			0.1			0.1			0.1	°C
Long Term Stability[5]			0.1			0.1			0.1	°C/month
ABSOLUTE MAXIMUM RATINGS										
Operating Temperature	–25		+105	–25		+105	–25		+105	°C
Package Temperature[6]	–45		+125	–45		+125	–45		+125	°C
Forward Voltage (+ to –)			**44**			**44**			**44**	V
Reverse Voltage (– to +)			**20**			**20**			**20**	V
Lead Temperature (Soldering 10 sec)			300			300			300	°C
POWER SUPPLY										
Operating Voltage Range	4		30	4		30	4		30	V
Power Supply Rejection										
+4 V < V_S < +5 V			**0.5**			**0.5**			**0.5**	°C/V
+5 V < V_S < +15 V			**0.2**			**0.2**			**0.2**	°C/V
+15 V < V_S < +30 V			**0.1**			**0.1**			**0.1**	°C/V

NOTES
[1]An external calibration trim can be used to zero the error @ +25°C.
[2]Defined as the maximum deviation from a mathematically best fit line.
[3]Parameter tested on all production units at +105°C only. C grade at –25°C also.
[4]Maximum deviation between +25°C readings after a temperature cycle between –45°C and +125°C. Errors of this type are noncumulative.
[5]Operation @ +125°C, error over time is noncumulative.
[6]Although performance is not specified beyond the operating temperature range, temperature excursions within the package temperature range will not damage the device. Specifications subject to change without notice.
Specifications shown in boldface are tested on all production units at final electrical test. Results from those tests are used to calculate outgoing quality levels. All min and max specifications are guaranteed, although only those shown in boldface are tested on all production units.

METALIZATION DIAGRAM



TEMPERATURE SCALE CONVERSION EQUATIONS



$°C = \frac{5}{9}(°F - 32)$ $K = °C + 273.15$

$°F = \frac{9}{5}°C + 32$ $°R = °F + 459.7$

ORDERING GUIDE

Model	Max Cal Error @ +25°C	Max Error –25°C to +105°C	Max Nonlinearity –25°C to +105°C	Package Option
AD592CN	0.5°C	1.0°C	0.35°C	TO-92
AD592BN	1.0°C	2.0°C	0.4°C	TO-92
AD592AN	2.5°C	3.5°C	0.5°C	TO-92

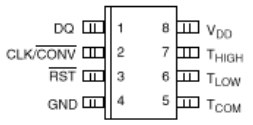
DALLAS SEMICONDUCTOR

DS1620
Digital Thermometer and Thermostat

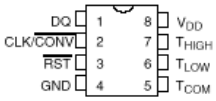
FEATURES

- Requires no external components
- Supply voltage range covers from 2.7V to 5.5V
- Measures temperatures from –55°C to +125°C in 0.5°C increments. Fahrenheit equivalent is –67°F to +257°F in 0.9°F increments
- Temperature is read as a 9–bit value
- Converts temperature to digital word in 1 second (max)
- Thermostatic settings are user–definable and non–volatile
- Data is read from/written via a 3–wire serial interface (CLK, DQ, $\overline{RST}$)
- Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system
- 8–pin DIP or SOIC (208 mil) packages

PIN ASSIGNMENT



DS1620S 8–PIN SOIC (208 MIL)
See Mech Drawings Section



DS1620 8–PIN PDIP (300 MIL)
See Mech Drawings Section

PIN DESCRIPTION

DQ	–	3–Wire Input/Output
CLK/$\overline{CONV}$	–	3–Wire Clock Input and Stand–alone Convert Input
$\overline{RST}$	–	3–Wire Reset Input
GND	–	Ground
T_{HIGH}	–	High Temperature Trigger
T_{LOW}	–	Low Temperature Trigger
T_{COM}	–	High/Low Combination Trigger
V_{DD}	–	Power Supply Voltage (3V – 5V)

DESCRIPTION

The DS1620 Digital Thermometer and Thermostat provides 9–bit temperature readings which indicate the temperature of the device. With three thermal alarm outputs, the DS1620 can also act as a thermostat. T_{HIGH} is driven high if the DS1620's temperature is greater than or equal to a user–defined temperature TH. T_{LOW} is driven high if the DS1620's temperature is less than or equal to a user–defined temperature TL. T_{COM} is driven high when the temperature exceeds TH and stays high until the temperature falls below that of TL.

User–defined temperature settings are stored in non–volatile memory, so parts can be programmed prior to insertion in a system, as well as used in stand–alone applications without a CPU. Temperature settings and temperature readings are all communicated to/from the DS1620 over a simple 3–wire interface.

OPERATION–MEASURING TEMPERATURE

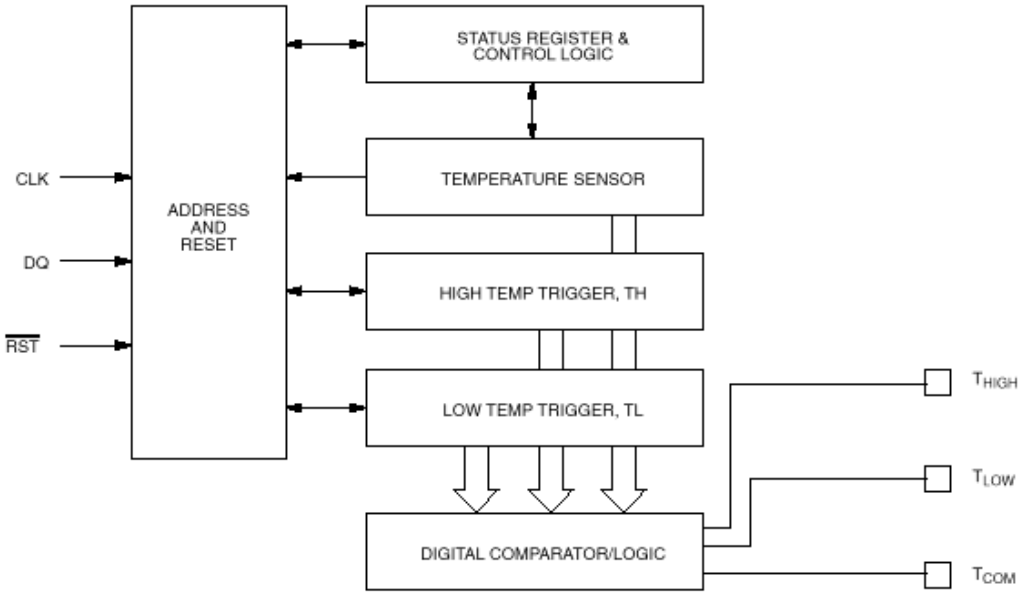
A block diagram of the DS1620 is shown in Figure 1. The DS1620 measures temperatures through the use of an on–board proprietary temperature measurement technique. A block diagram of the temperature measurement circuitry is shown in Figure 2.

The DS1620 measures temperature by counting the number of clock cycles that an oscillator with a low temperature coefficient goes through during a gate period determined by a high temperature coefficient oscillator. The counter is preset with a base count that corresponds to –55°C. If the counter reaches zero before the gate period is over, the temperature register, which is also preset to the –55°C value, is incremented, indicating that the temperature is higher than –55°C.

At the same time, the counter is then preset with a value determined by the slope accumulator circuitry. This circuitry is needed to compensate for the parabolic behavior of the oscillators over temperature. The counter is then clocked again until it reaches zero. If the gate period is still not finished, then this process repeats.

The slope accumulator is used to compensate for the nonlinear behavior of the oscillators over temperature, yielding a high resolution temperature measurement. This is done by changing the number of counts necessary for the counter to go through for each incremental degree in temperature. To obtain the desired resolution, therefore, both the value of the counter and the number of counts per degree C (the value of the slope accumulator) at a given temperature must be known.

DS1620 FUNCTIONAL BLOCK DIAGRAM Figure 1

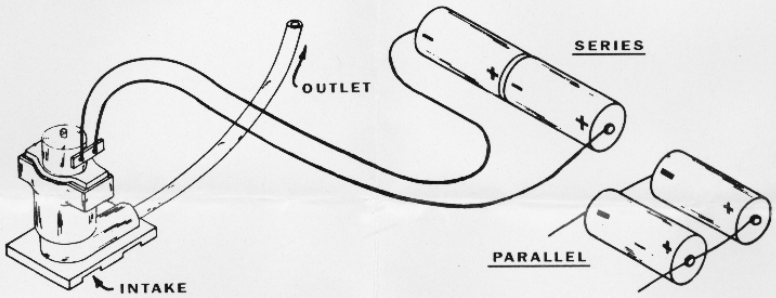


Edmund Scientific

INFORMATION AND INSTRUCTIONS

MINIATURE WATER PUMP
NO.50,345

This small water pump operates on 1½ to 3 volts direct current provided by batteries, either standard "D" cells or longer lasting No.6 ignition cells. The batteries may be connected in series, if you want highest capacity, or parallel for longer life. The illustration, below shows how to connect cells in series and parallel. To connect cells in series, connect the plus terminal of one to the minus terminal of the other; connect the pump wires to the remaining terminals on each battery. To connect cells in parallel, connect both plus terminals to one wire of the pump and both negative terminals to the other.



Capacity: **Approximately 1 pint/minute at 12" head.**
Current: **3 to 4 amperes at 3 volts.**

Although it will pump water from the outlet whem the impeller is rotating either way, the pump is slightly more efficient when the impeller is rotating counter clockwise (as viewed from the top).

Some suggested uses for this water pump:

1. Temperature control and circulation of darkroom chemicals
2. Table fountains and displays
3. Model waterfalls, hydroelectric installations, canal locks, etc.
4. Water filtration and aeration of fish tanks
5. Flow models of circulatory systems
6. Automatic plant watering unit
7. Humidifier

A NOTE OF CAUTION: The pump should not be placed in more than 1" of water as higher water level will flood the motor.

Edmund Scientific Co. 7785 Edscorp Bldg., Barrington, N.J. 08007