# Earth Measurements

## Student Guide for Experiments #1 through #6

**Version 1.0**

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 alterned 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.

# Preface

The subject of these six lessons is Earth Measurements. Think of yourself as a geologist, wanting to know more about *El Nino*, 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 a lot of interesting careers that can combine knowledge of electronics with a love for the earth environment. Scientist, engineer, modern farmer. Why not put computers to work for the good of our planet?

# 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, 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. This series was written by Dr. Tracy Allen, who chose to include only the answers to the Challenge section in a separate Teacher's Guide. In comparison, the "What's a Microcontroller?" Teacher's Guide included answers to fill-ins and basic questions. The Earth Measurement Teacher's Guide will be available by e-mail request to stampsinclass@parallaxinc.com.

## 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. For example, the commercially available OWL2C uses a BASIC Stamp II or IISX microcontroller, providing reprogramming capabilities for a customer who doesn't 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. 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 lesson, you will start off by experimenting with a transducer that converts electrical impulses from the BASIC Stamp into musical tones. This audio enunciator will provide useful feedback about the operation of the BASIC Stamp throughout this series of lessons. The main project in this lesson 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 lessons, 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 lesson 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 lessons. 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 lesson is simply a buzzer, a sound output device. In fancy terms, it is an "enunciator", or a "**piezoelectric transducer**". It will be a big part of our user interface in the projects to come in this set of lessons 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 enunciator 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.

### 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.

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. The schematic is shown in

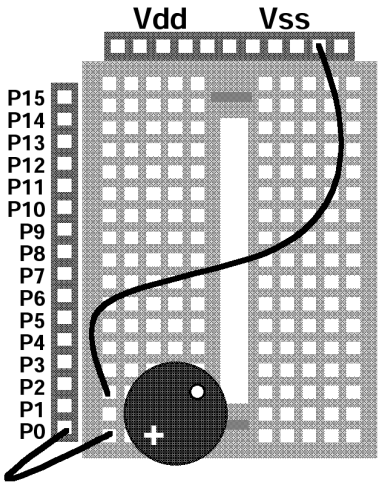



**Figure 1.1: Peizo Transducer Pictorial**

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

Please note. The six lessons 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 (BoE). To avoid having to rewire things later, please follow the suggested parts placement.

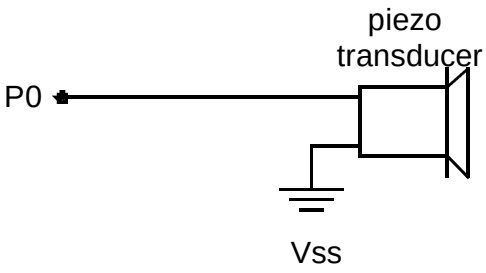



**Figure 1.2: Peizo Transducer Schematic**
Electrical schematic of pictorial shown in Figure 1.1


**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 Stamp2.exe editor on the PC and type in the following PBASIC program:

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

That's it. A one-line program.

**In case of difficulty during download:**

If ALT-R 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 BoE has a good power supply and that the power supply indicator light on the BoE 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 sim;ply won't work, then check the wiring on your Board of Education. Compare it to the wiring shown on the pictorial.

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". 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 BOE, 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 from the BASIC Stamp 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.


**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
```

Run it using ALT-R. 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 (Appendix C has a complete Morse code table):

| 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 | 11100 |
| 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 lessons, 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 lesson 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 B). 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 BoE 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 BoE, 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 double check it, before you reconnect the power. The schematic for the DS1620 is shown in Figure 1.4.

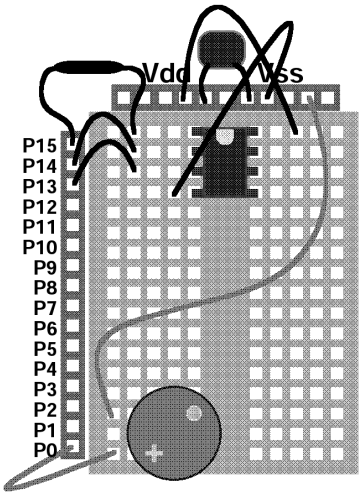



**Figure 1.3: DS1620 Pictorial**
Plug the DS1620 in at the very end of the BOE. Observe that there is a notch at one end of the DS1620 package to indicate the polarity. Be careful not to reverse the power supply.

- 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 2 wired to BASIC Stamp P13

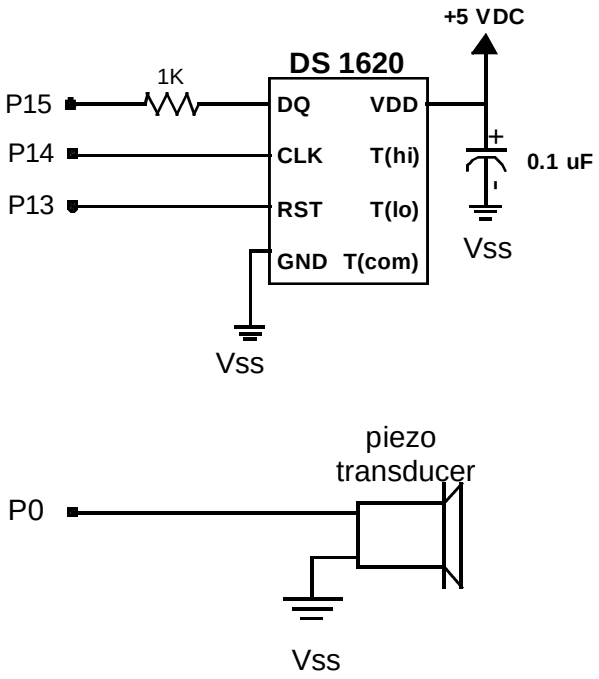



**Figure 1.4: DS1620 Schematic**
Schematic of the circuit pictured above. Remember – the piezo transducer portion of the circuit has already been built.

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 the polarity. 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.

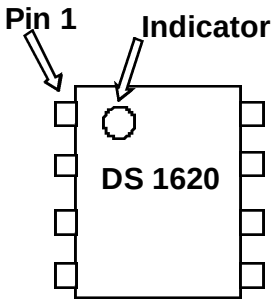
Enter the following program. Again, you don't have to enter the remarks after and including the '.

```
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. Run (ALT-R) the program.

**What's an Indicator:**

The DS1620 is an 8-pin DIP package. The indicator denoting the DS1620's pin 1 is a small circle in the upper left-hand corner. 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, spot of paint, 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, and 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 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.

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).

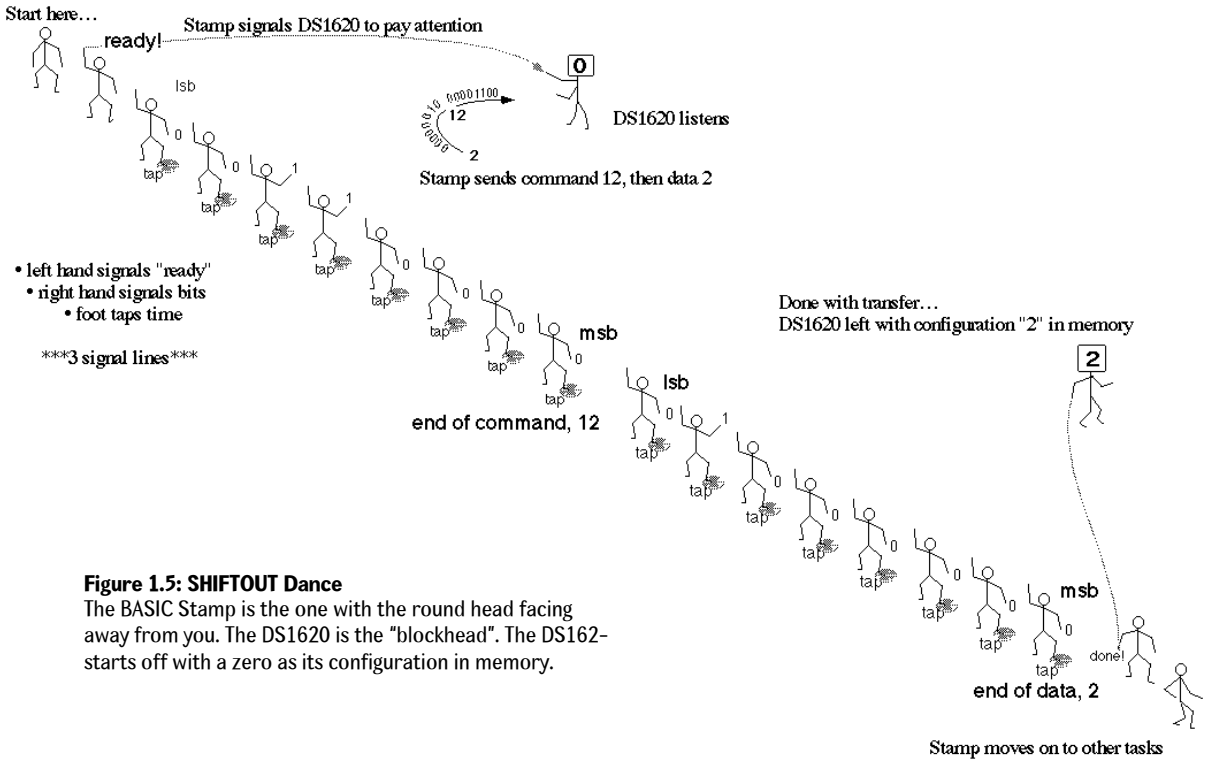



**Figure 1.5: SHIFTOUT Dance**
The BASIC Stamp is the one with the round head facing away from you. The DS1620 is the "blockhead". The DS162- starts off with a zero as its configuration in memory.

It isn't over yet. The DS1620 is now waiting for another binary number to follow the 12. The 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 would 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: oooo5555555555555555555555555555555555555555555555555555555ooo CS
P14: ooooooo5oo5oo5oo5oo5oo5oo5oo5ooooooo5oo5oo5oo5oo5oo5oo5oo5ooooo Clock
P15: ooooooooooo555555ooooooooooooooooooooo555ooooooooooooooooooooooo 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=%1111111111111111          ' 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
```

Run the program using ALT-R.

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 BoE?!) 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 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=%1111111111111111     ' 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 lessons. 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
```

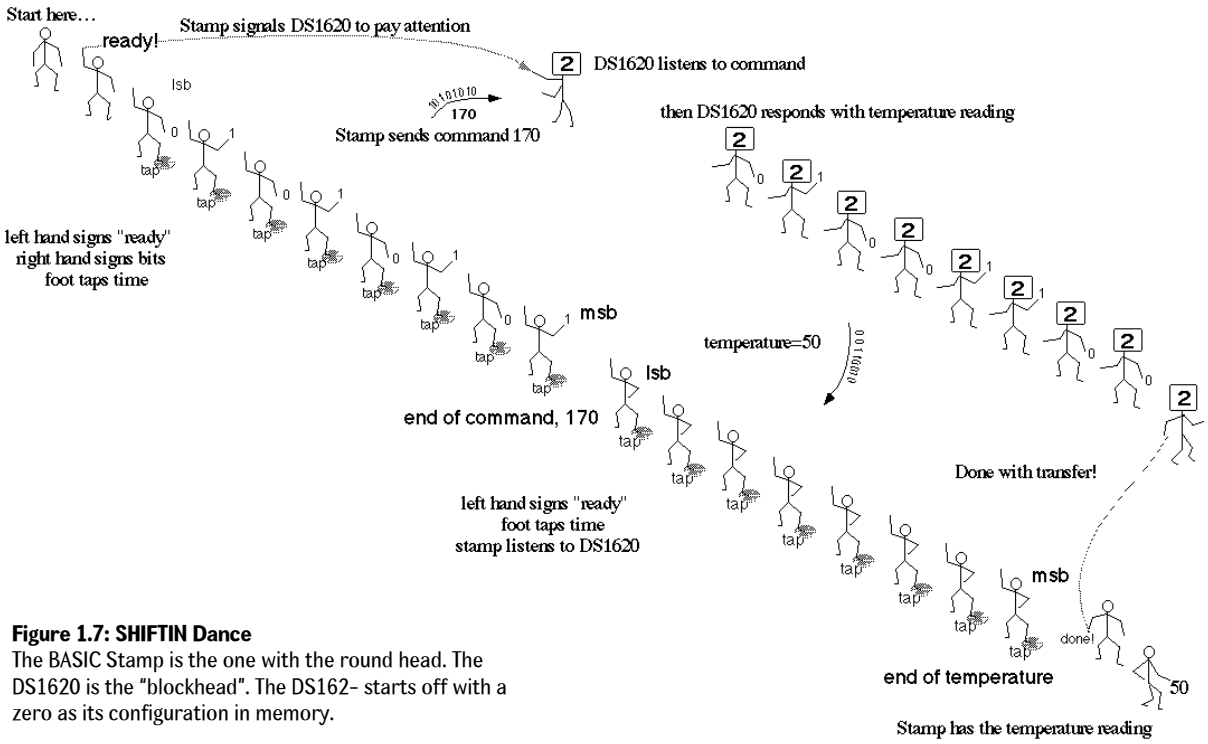



**Figure 1.7: SHIFTIN Dance**
The BASIC Stamp is the one with the round head. The DS1620 is the "blockhead". The DS162- starts off with a zero as its configuration in memory.

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. For your edification, 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**
Engineer's picture of how the SHIFTIN command works.

```
time: ----------------------one millisecond---------------------->
P13: oooo555555555555555555555555555555555555555555555555555ooo  CS
P14: oooooo5oo5oo5oo5oo5oo5oo5oo5ooooooo5oo5oo5oo5oo5oo5oo5ooooo  Clock
P15  ooooooooo555ooo555ooo555ooo555ooooooo555ooooooo55555ooooooooo  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 lesson, you should save the program you have just typed in and debugged. In this series of lessons, 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.

**STAMP2W.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 Beetoven'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 = degC * 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, a 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 (stardate). 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 lesson 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 lessons to come. Also, you will improve on your DS1620 thermometer from the previous lesson, 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 Lesson 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 lesson. You already have an audio enunciator for output. Now, install a pushbutton for input, as shown in Figure 2.1. The schematic is shown in Figure 2.2.

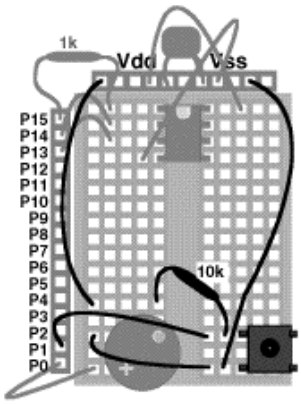



**Figure 2.1: Pictorial**
Install a pushbutton (PB) at the very end of the BOE, across from the piezo transducer. Two of the pins will hang over the edge of the plastic terminal block, so as to leave a couple of holes free for wiring, as shown. Hold the

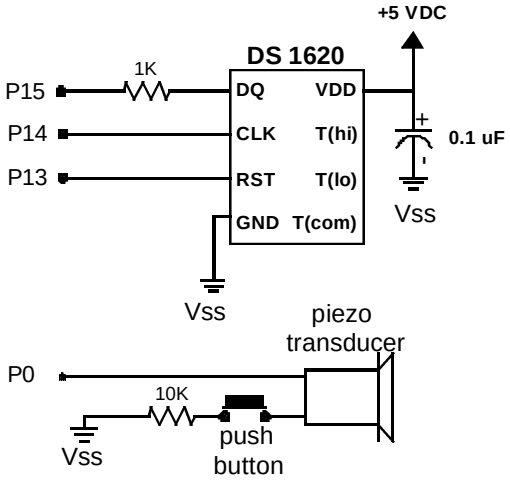



**Figure 2.2: Schematic**
Install a pushbutton (PB) at the very end of the BOE, across from the piezo transducer. Two of the pins will hang over the edge of the plastic terminal block, so as to leave a couple of holes free for wiring, as shown. Hold the loose end of the pushbutton in place with a piece of strong platic 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
```

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? The troubleshooting is left to you.

### What's all this DEBUG stuff?

In these lessons, 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 lesson 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**

**00000011111111111111111111000000011111...**

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

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

```
' Earth Measurements program 2.2
' buzzer
click:       ' 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 life 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. No wonder it's hard to program a VCR!

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 lessons, 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.

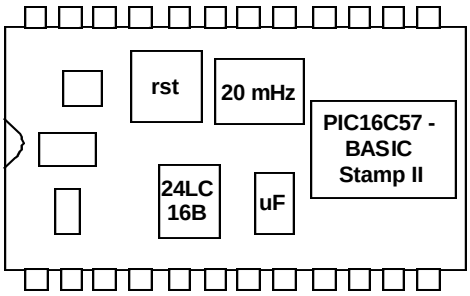



**Figure 2.3: BASIC Stamp Memory**
The PIC16C57 chip is the BASIC Stamp's RAM and central processor. The 24LC16B is the EEPROM, which holds your PBASIC program and data we will be storing.

Variables are created in the RAM 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 where 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 lessons 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                               ' here to 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
```

We leave it to you 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, can you modify the program as follows:

1. Add a data statement with a label of "dur" and make 7 0 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 "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, STAMP2W.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 lesson 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 lesson 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: DS1520 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 mx
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 lesson 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 lesson, 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.

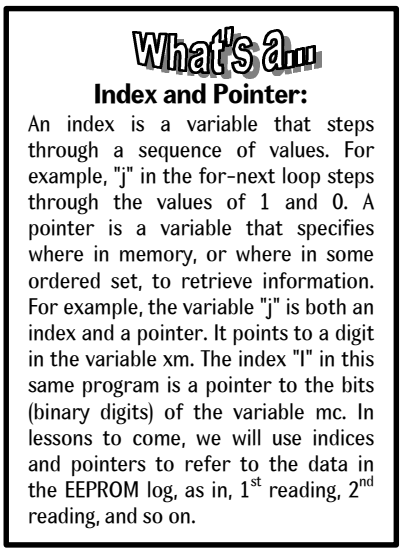
The central idea of the Morse subprogram is held in the binary pattern, %11110000011111. The % sign mark it as a binary number. This is the pattern of zeros and ones as they are actually stored in binary brain of the BASIC Stamp. This binary number does have a standard numerical value (=15391), but the 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                                  ' ❒
```




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 2) 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 ----^
  =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. Above 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.bit0` 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)   diffferent 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.

## 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 | 6 |
| 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 | 6 |

**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 | 3 |
| 150-01020 | 1K ¼ watt 5% resistor | 1 |
| 150-01030 | 10K ¼ watt 5% resistor | 1 |
| 150-01040 | 100K ¼ watt 5% resistor | 1 |
| 150-01000 | 10 ohm 1W resistor metal oxide | 1 |
| 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 | 1 |
| 400-00001 | Pushbutton | 1 |
| 604-00002 | DS1620 Digital Thermometer (Dallas Semiconductor) | 1 |
| 604-00010 | 555 timer, 8-pin DIP | 1 |
| 700-00002 | 4-40 machine screws | 2 |
| 700-00003 | 4-40 x 3/8" nut | 2 |
| 800-00016 | 3" jumper wires | 10 |
| 800-00021 | 16" red jumper wire | 2 |
| 800-00022 | 16" black jumper wire | 2 |
| 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 |
| N/S | Cup spanner made of plastic or printed circuit board, with two holes for screws | 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.**<br>3805 Atherton Road, Suite 102<br>Rocklin, CA 95765 USA<br>(916) 624-8333, fax (916) 624-8003<br>http://www.stampsinclass.com<br>http://www.parallaxinc.com | Parallax and Stamps in Class source. Manufacturer of the BASIC Stamp. |
| United States | **Peter H. Anderson**<br>915 Holland Road<br>Bel Air, MD 21014<br>(410) 838-6500, fax (410) 836 8526<br>http://www.phanderson.com | Parallax Stamps in Class distributor and professor. Stocks many do-it-yourself BASIC Stamp kits using the "home-brew" approach. |
| United States | **Digi-Key Corporation**<br>701 Brooks Avenue South<br>Thief River Falls, MN 66701<br>(800) 344-4539, fax (218) 681-3380<br>http://www.digi-key.com | Source for electronic components. Parallax distributor. May stock Board of Education. Excellent source for components. |
| United States | **Mouser Electronics**<br>345 South Main<br>Mansfield, TX 76203<br>(800) 346-6873, fax (817) 483-6899<br>http://www.mouser.com | Source for electronic components. Parallax distributor. May stock Board of Education in 1999. Excellent source for components. |
| Australia | **Microzed Computers**<br>PO Box 634<br>Armidale 2350<br>Australia<br>Phone +612-67-722-777, fax +61-67-728-987<br>http://www.microzed.com.au | Parallax distributor. Stamps in Class distributor. Excellent technical support. |

| | | |
|---|---|---|
| Australia | **RTN**<br>35 Woolart Street<br>Strathmore 3041<br>Australia<br>phone / fax +613 9338-3306<br>http://people.enternet.com.au/~nollet | Parallax and Stamps in Class distributor. |
| Canada | **Aerosystems**<br>3538 Ashby<br>St-Laurent, QUE H4R 2C1<br>Canada<br>(514) 336-9426, fax (514) 336-4383 | Parallax distributor and Stamps in Class distributor. |
| Canada | **HVW Technologies**<br>300-8120 Beddington Blvd NW, #473<br>Calgary, AB T3K 2A8<br>Canada<br>(403) 730-8603, fax (403) 730-8903<br>http://www.hvwtech.com | Parallax distributor and Stamps in Class distributor. |
| Germany | **Elektronikladen**<br>W. Mellies Str. 88<br>32758 Detmold<br>Germany<br>49-5232-8171, fax 49-5232-86197<br>http://www.elektronikladen.de | Parallax distributor and Stamps in Class distributor. |
| New Zealand | **Trade Tech**<br>Auckland Head Office, P.O. Box 31-041<br>Milford, Auckland 9<br>New Zealand<br>+64-9-4782323, fax 64-9-4784811<br>http://www.tradetech.com | Parallax distributor and Stamps in Class distributor. |
| United Kingdom | **Milford Instruments**<br>Milford House<br>120 High St., S. Milford<br>Leeds YKS LS25 5AQ<br>United Kingdom<br>+44-1-977-683-665<br>fax +44-1-977-681-465<br>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

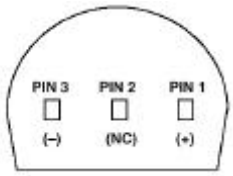
The Earth Measurements experiments use the AD 592 temperature transducer. The part needs to be protected before being inserted into liquid. Parallax builds a custom temperature probe (#28130), but you can do this yourself from these plans. An abbreviated datasheet for the AD 592 is included in Appendix F of this text.

**You'll need the following materials:**

(1) AD592 Temperature Transducer in plastic TO-92 case
(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

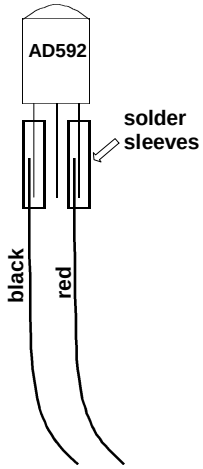
**To build the probe:**

1. Identify the AD 592's (-), NC, and (+) pins from this picture as viewed from the bottom.



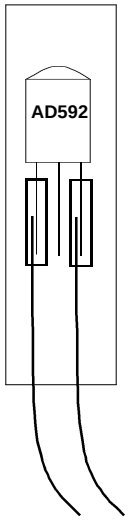

2. Slip the solder sleeve over the black wire and pin 3 (-). Slip another solder sleeve over the red wire and pin 1 (+). Heat up the connections until the wires are joined.

   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.

Bands 1 through 3 tell us what the actual value is, measured in "ohms". The higher the value, the less current is permitted to flow through it (at a given voltage).

The color values are as follows:

| | |
|---|---|
| Black | 0 |
| Brown | 1 |
| Red | 2 |
| Orange | 3 |
| Yellow | 4 |
| Green | 5 |
| Blue | 6 |
| Violet | 7 |
| Grey | 8 |
| White | 9 |

To determine the value of a resistor, look at the first color, determine its value from the above chart & write it down. Do the same for the second band. The third band "is the number of 0's to write down". For example:

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 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 |