

BASIC Stamp Laboratory, Part 1

MEAM 410

Developed by Mike Mistry, Ken McIsaac, and Jim Ostrowski

The purpose of this lab is to explore the Parallax BASIC Stamp II Microcontroller in various mechatronic applications and to expose you to some possible ways in which it can be utilized. The Stamp's flexibility and ease of programming make it a very good choice for sensing and electronic control of mechanical systems. The tradeoff in using it is that it runs at slower speeds than some other microcontrollers. However, for our purposes, the Stamp should provide easily enough processing speed and flexibility. Attached to this assignment is a diagram of the BASIC Stamp with a brief description of the input/output (I/O) pins.

This lab (and its sequel) can be done at your own pace, using the set-up found in the rear table of the GM lab. The manual for the BASIC Stamp is by the computer or can be borrowed from Terry Kientz. This manual can be a very helpful reference, particularly for programming in BASIC and for understanding the Stamp commands. It is NOT to be removed from the GM lab area for any reason.

Here is a brief **outline** of the lab, which has been broken into two components:

Part I:

1. Button timing and sample programs
2. Reading resistances
3. Driving an LED display
4. Driving a speaker

Part II:

1. Analog to Digital Converter (ADC)
2. Interfacing with several sensors
3. Variable speed control of DC motor
4. Driving an R/C servomotor

NOTE: As always when working with electronic circuits, check and double-check your circuit before connecting the power. If you see or smell smoke, disconnect the power IMMEDIATELY! This is invariably a bad sign!

1. Button Timer

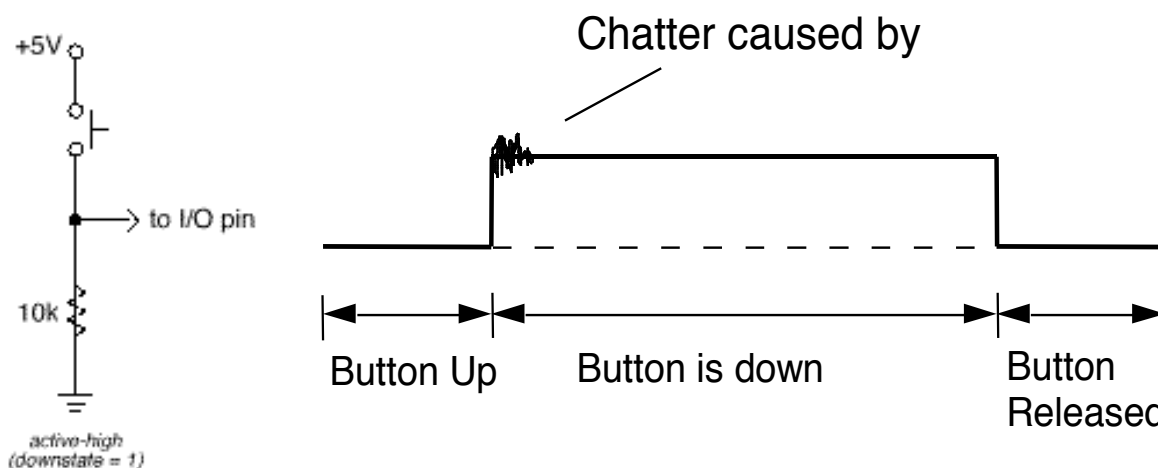
Before programming the Stamp to do anything useful, let's make sure that it's working and we know how to write and run a simple BASIC program. First, the Stamp should be connected to a power supply. The easiest way to do this is to plug in a 9 V battery. The second best way is to connect a power supply (5-15 V) to the pin labeled V_{in} . All connections to the Stamp should be made by plugging connecting wires into the screw down terminals on the Stamp board that you've been provided. Finally, the Stamp must be connected to the PC in order to download programs. This connection is through the serial cable provided. Finally, we can write and execute programs on the Stamp using the editor/compiler "stamp.exe" on the PC. In the directory STAMP2, simply type "stamp" to run this program. Here is a sample program:

```
count var word          'Define the variable count
count = 0                'Initialize count to zero
loop:                    'Start of loop
  count = count + 1
  debug dec count        'Display the count as a decimal
  goto loop              'Go back to beginning of loop
```

This program can be run by typing "Alt-R". The statements following the single quote (" ' ") are comments, and do not need to be entered.

- What happens when it is run?
- About how fast does the program count (i.e., how long does it take to count to 255)?

Next, we will look at interfacing the Stamp with external circuitry. As an example of the Basic Stamp's functionality, let's suppose that we needed to know the length of time a push button is held down. Prepare the circuit according to the left-hand figure below, such that when the button is held down, a 5-volt supply is connected to pin 3. When the push button is released, pin 3 will be grounded. Thus, when the button is held down for a period of time and then released, pin 3 will receive an input pulse of 5 volts.



Let's start by trying the simplest thing we can do by simply connecting the circuit as above to the Stamp (i.e., connect the line "I/O pin" above to pin3 on the Stamp). Next, display the state of pin3 using the following program (note, the write-up for this lab requires you to hand in code used in each section, so be sure to save a copy of each piece of working code to floppy or print it out):

```
value var word      'Define a variable called value
input 3             'Set pin 3 as an input
loop:
    value = in3      'Read the value of pin3 (0 or 1)
    DEBUG ? value    'Echo value to the screen
goto loop
```

The second line ("input 3") sets pin 3 as an input pin (conversely, "output 3" would make pin 3 an output pin). The line "loop:" simply acts as a location marker in the program and does nothing else. The real action occurs in the next line, where the value of pin3, either a 1 or a 0, is stored in the variable value, and then displayed.

Next, combine the two programs in order to count the amount of time between when the button is pressed and when it is released. You will need to add an "IF" statement, which has the standard syntax: "IF value = 1 then place", for example, where place is a location marker for the if-then branch.

- How does this program work in practice?
- Do you get values that make sense for the amount of time the button is held down?

The problem with reading in the length of time that a button is pushed is that in reality the button contact experiences a very fast series of mechanical bounces that introduce error into the measurements. For this reason, the readings that one gets are often not accurate, since they reflect the time between bounces, and not the time between when the button is pushed and when it is released. There are several ways to "debounce" a push-button, in order to fix this problem. First, in (electronic) hardware, we can attempt to smooth the signal using a capacitor. Try connecting a capacitor (say 0.1 μ F) in parallel with the resistor.

- Why should this circuit help debounce the button (try to explain it using your knowledge of resistors and capacitors hooked together)? It may be helpful here to use an oscilloscope to see what's happening when the button is being pressed.
- Approximately how many counts do you find the Stamp makes per second?

Alternatively, we can debounce the circuit in software. One way to do this is to take two measurements, separated by a time interval longer than the expected time of the mechanical bounces. That is, we measure for the button being pushed, and then check again after a small period of time to see that it is still pressed. This is essentially what is done in the Stamp's BUTTON command. A sample button command might look like:

```

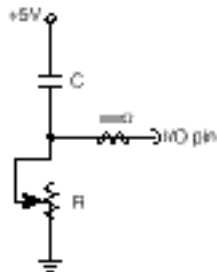
store var word
loop:
    Button 3,1,255,0,store,1,done 'see if pin 3 is pressed
    debug dec ? store
goto loop
done:
'start next commands...

```

There are several arguments to the BUTTON command that are interesting-- a copy of the manual pages for it has been included. We have also used a variable name "store" to hold the value of the Button command. In general, you should use either a declared variable, like "store", or one of the Stamp's internal variables, like b1, w1, etc., but not both at the same time.

2. Reading Resistances:

The Stamp provides a built-in function for reading resistances from 5 to 50 k Ω in a passive circuit. In the BASIC Stamp I, this is called the "pot" (short for "potentiometer") command. A modified (and enhanced) version of this command is given by the RCTIME command for the Stamp II. A description of this command is attached to this document for reference. To summarize, the RCTIME command measures the RC discharge time of a network connected to one of its input pins. If one knows the capacitance value, this gives a means of determining the value of the resistance. For our purposes, we will use a 0.1 μ F capacitor that should be placed between +5V and the input pin used to measure the resistance. A sketch of this setup is given here.



A sample program to store and display the value of the potentiometer is also given:

```

result var word          ' Variable to hold result.
loop: high 7              ' Discharge the cap
    pause 1               ' for at least 1 ms.
    RCTIME 7,1,result     ' Measure RC charge time.
    debug ? result        ' Show value on screen.
goto loop

```

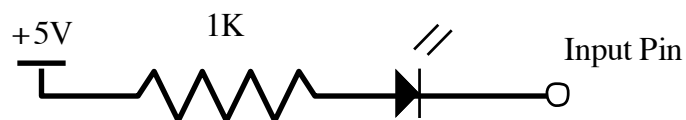
In this command, the Stamp reads pin 7, and stores the discharge time in the variable `result`. Notice that pin 7 must be held high for 1 ms to start the loop. The reason for this is described in more detail in the description of the RCTIME command.

Test out this feature by measuring the resistance of several resistors. Use this to derive a rough calibration for the measurements. Then try it on one of the potentiometers provided, by connecting the middle lead on the potentiometer to the input pin on the stamp (and to +5V through a $0.1\mu\text{F}$ capacitor), and either of the other leads to ground.

- What happens as you vary the potentiometer?
- Derive a function (analytically) that will tell you how long it should take the voltage on the I/O pin to fall from 5V to 2V given a resistance, R .

3. LED display

The LED display provides a very simple means of communicating data back to the user. The LED is turned on when current flows across it (it is biased to only work in one direction, so you will need to figure out which direction this is, and should mark it on the LED block). Since the LED is just a (light-emitting) diode, it offers very little resistance. If we connect a power supply directly across it, the resulting current will be very large and will likely burn out the LED (not a good thing). To protect the LED, we connect it to the Stamp through a resistor, as shown:



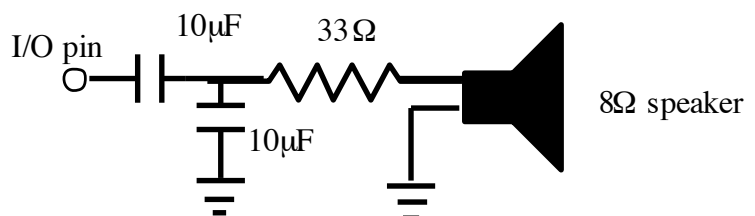
You can try this with the fixed power supply just to make sure it is working. For what value of the pin output will the LED be on or off? Next, build an LED display that acts similar to the output level LED display found in a stereo receiver/amplifier. Start by connecting six of the Stamp's pins to one of the LED blocks (each through a resistor). Then write the necessary BASIC code that will display the magnitude of a given number by lighting up a proportional number of LED's (for example, if the number is in the range 0-25, no LED's light up, 26-50 one LED, 51-75 two LED's, and so on). Finally, test this out by having the Stamp perform a counting routine as given above in the sample program and displaying the result on the LED.

- Describe what the output looks like on the display.

4. Driving a speaker

The Stamp provides a very convenient means for generating sounds, when connected to a speaker, using the `FREQOUT` command. A speaker for this lab has been provided, and should be hooked up as shown below. Can you write a little song with this? The

Stamp manual has more details on how to generate specific tones, for example, middle C is around 523 Hz. when using the `FREQOUT` command (for example, the command `FREQOUT 0,350,523` would play a 'C' on pin 0 for 350 ms.). Beware, the speaker is very quiet, so you may need to hold it up to your ear! (It could use some amplification).



BASIC Stamp Lab, Part II

5. Analog to Digital Converter:

Since most physical quantities are analog, it is important for digital microcontrollers to be able to read analog signals. Although some new microcontroller chips have built in A/D capability, the Basic Stamp is a completely digital device and so cannot read in an analog signal directly. For example, the Stamp would interpret both a 5 V and a 30 V input as the logic value 1, at least until the device started to emit smoke. As a result, the Stamp needs some external circuitry if it is going to be used in an analog sampling application. In this section of the lab, we are going to be experimenting with the National Semiconductor ADC0831 (a description is attached), an 8-bit, serial, analog-to-digital converter. Although devices used in industry can have higher resolutions (12- and 16- bits are not uncommon), most ADCs function in a manner very similar to the ADC0831.

The ADC0831 works by successive approximation. A more detailed description of this technology will be given in class, but a quick review here is worthwhile. Analog voltages are always computed as a fraction of some reference voltage (typically 2.5V or 5V). Therefore, each bit in the result is a descending power of 1/2 times the reference voltage. To illustrate:

```
b10000000 = 128 gives a value of 128/256 or 0.5*reference
b01000000 = 64 gives a value of 64/256 or 0.25*reference
b00100000 = 32 gives a value of 32/256 or 0.125*reference
and so on.
```

Thus, the measured voltage is represented as a binary number, which for an 8-bit ADC gives a range of 0 to 255. The actual voltage is then the fraction of that number over the whole range (here, 256) multiplied by the reference voltage. One interesting thing to observe is that it is typically impossible to represent the reference voltage itself, since the "all 1" bit pattern b11111111 corresponding to 0.996 times the reference voltage is the largest number possible in this scheme. Successive approximation is the simplest technique of determining the digital representation of an analog signal. Basically, each bit starting from the most significant (bit 7) to the least significant (bit 0) is "tried" one after the other. If the resulting voltage is larger than the sampled voltage, the bit is rejected, and the next one is tried. Here's an example of how the process works, assuming an input of 3.1V and a reference of 5V:

Current bit	Test voltage (V)	Keep?	Binary representation
7	2.5	Y	b1XXXXXXX
6	3.75	N	b10XXXXXX
5	3.125	N	b100XXXXX
4	2.8125	Y	b1001XXXX
5	2.96875	Y	b10011XXX
6	3.046875	Y	b100111XX
7	3.0859375	Y	b1001111X
8	3.10546875	N	b10011110 = 158

So the resulting representation would be b10011110 or $158/256 \times 5 = 3.086V$. You should notice two interesting points about successive approximation. The first is that because it always underestimates the correct analog signal, it is possible to choose the “wrong” approximation. 3.1054 is a much closer approximation than 3.086, but it is rejected because it is an overestimate. Techniques exist to avoid this problem, but we will not explore them here. The second thing to notice is that individual bits are correct as soon as they are determined. This is why it makes sense to use serial access for an ADC. If we assume that each bit is tried during one sample of a serial clock, then we are able to shift out bit 7 during the second clock cycle, and the complete answer is available after 9 cycles--only 1 cycle more than was required for the conversion itself.

Communication with the ADC8031 is performed using a simple serial interface. Please refer to the timing diagram on page 9 of the ADC8031 data sheet. The chip select line begins the transmission with a falling edge. At that point the clock begins. The first bit of the result (bit 7, as we saw above) will be available on the rising edge of the third clock cycle, and each successive bit will be available on a further rising edge. The least significant bit can be read on the rising edge of the 10th clock cycle. After the 10th clock cycle, a new conversion can be initiated by raising and lowering the chip select line.

The clock must have a frequency of at least 10kHz, which precludes “bit-banging” (using HIGH and LOW instructions for programmed control of the clock line), so you will have to use the Stamp’s SHIFTIN instruction. You can find the documentation for SHIFTIN in the Stamp programmer’s reference. You will have to program SHIFTIN to receive 10 bits of data on the rising edge. Don’t forget in your programs that the two most significant bits contain garbage and should be ignored.

Now that you know how to use the ADC you are ready to write programs with it. For this part of the lab, you need to program the BASIC Stamp to be an analog voltmeter, where it will light up to six LED’s corresponding to an input voltage of 0 to 5 volts.

1. Turn on the variable power supply and connect it to the ADC, and connect the ADC to the Stamp.
2. Write a program to send the necessary serial pulses to the ADC and to read in the voltage being measured (using the SHIFTIN command). Display the value on the computer screen. Make sure that the serial port cable is attached to the Stamp so that the Stamp can send data to the computer. Have your program run in a continuous loop. Record the values returned for various voltages from 0 to 5V. Make a graph of pulse widths verses input voltages and discuss anything interesting that you observe.
3. Modify the program that you’ve already written for the LED’s to work as a 6 LED voltmeter. As the analog input is increased, LED’s on the meter should light up linearly. All LED’s should be off if the input is less than 5 volts, and one LED should light for every 5 volts, until all 6 LED’s are lit at approximately 5V.

6. Interfacing with various sensors

In this section, you are to develop an interface with several types of sensors, and will use the LED display to provide an output. The types of sensors you can choose from are the following:

GROUP A

1. Phototransistor
2. Thermistor

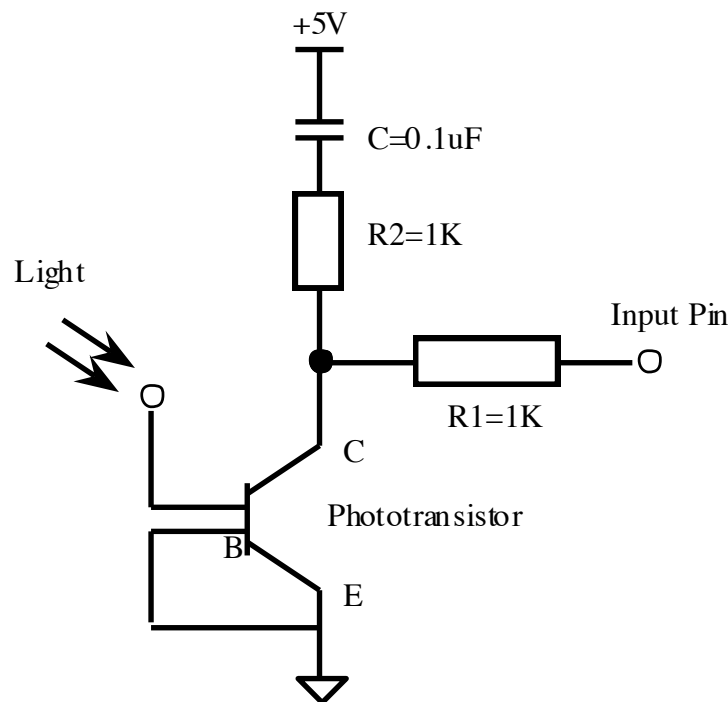
GROUP B

1. Ultrasound ranger
2. Ball-type tilt switch

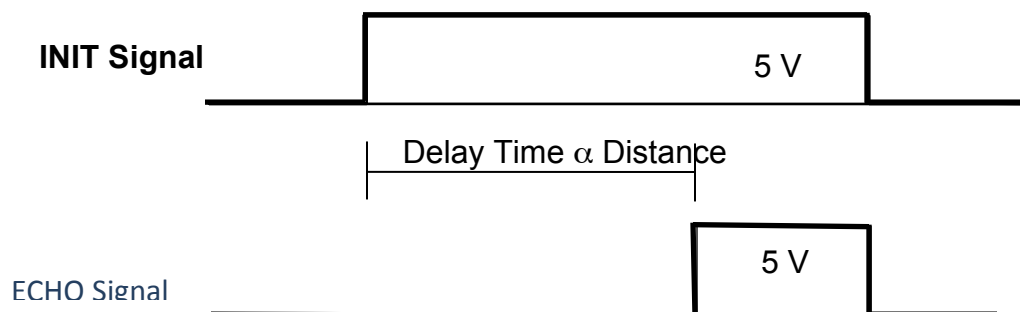
You are required to use at least one sensor from each of GROUP A and B. In each case, you should use the LED display (to display the outputs). At least one of them must be hooked up to the LED display to run as an analog display, with at least four LED's being driven to show the level (as was done in the A/D converter above).

We will describe each of these elements briefly, though the basic function of each of these should be recalled from class:

1. Phototransistor: The phototransistor is a light sensitive device. Essentially, we will use it like a potentiometer, in almost the exact same way as was done in Part I, except that now instead of turning a knob, the control input is light. The circuit given below mimics the one used for the potentiometer, except that we have added a resistor in series with the capacitor, in order to limit the current flowing through the circuit (this probably would have been a good idea with the original circuit, as well). There are three leads on the phototransistor (Emitter, Base, and Collector—EBC). There is a tab on the actual part that designates the Emitter. From there the leads go in order, EBC. Essentially, the phototransistor is turned on (its resistance is reduced) when light is shown on it. When the bulb is covered, or is in a dark setting, the transistor yields a fairly high resistance. Use the transistor and the Stamp's RCTIME command to build a light sensor. What values for the time constant do you get for ambient (room) light, or for completely dark?



2. Thermistor: This element acts like a potentiometer, where the variation in resistance is caused by changes in temperature, instead of mechanically. Use the calibration from the resistor measurements in Part I to get an approximate range of the resistances offered by the thermistor. (The thermistor is the green electronic element that looks like a capacitor, given to you in your kits.)
3. Ultrasound Rangefinder: Using the given ultrasound kit and a Basic Stamp, create a distance meter. The ultrasound unit is operated by first switching its INIT pin from low to high. At that moment sonic waves are sent out. The waves will hit a target and bounce back, returning to the transducer. When the waves return, the unit's ECHO signal will go high. The delay time between INIT going high and ECHO going high is proportional to the distance between the target and the transducer. Note that the INIT signal must remain high throughout the entire cycle.

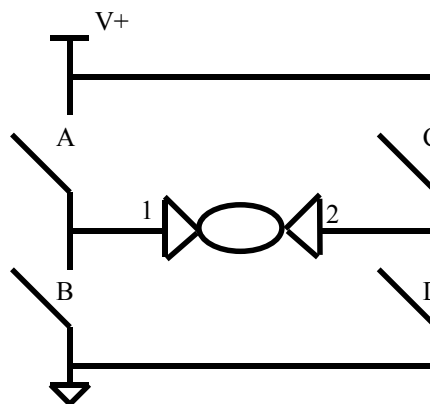


Write a Basic Stamp program to trigger the INIT signal, and to measure the ECHO delay time. On the Ultrasound module pin 1 is V+, pin 2 is ECHO, pin 5 is INIT, and pin 8 is ground. You'll need to write a counter program to count between when the INIT signal is first set high and when the ECHO signal is observed. Display your delay time variable on the screen (using DEBUG).

- Estimate the resolution of your range finder. What could be done to improve this resolution?
 - Explore the angular cone over which the sonar can pick up objects that are in its path. How far off the normal line can objects be observed?
4. Ball-type tilt switch: This is similar to the mercury switch discussed in class. The tricky part is how to wire this so that it can provide useful measurements of tilt. You will also need at least two LED's to indicate the direction of tilt.
- Include a description of the method you've chosen to mount the tilt switch and display the results in your report.
 - How could you provide a better resolution on the angle of tilt (e.g., by using multiple tilt switches, etc.)?

7. Variable speed control of DC motor:

Finally, we would like to use a measured quantity to drive a mechanical device. For this lab, we will keep it simple and drive a DC motor open loop using a pulse-width modulated (PWM) signal. To review the relevant theory, a typical motor drive system can be assumed to have the following form:



The motor turns when opposing pairs of switches are closed ([A, D] or [C, B]). We fix the voltage at one side by closing the appropriate switch. Then we pulse the switches on the other side so there is a voltage drop across the motor for a percentage of the time proportional to the voltage we are trying to drive on the motor. In slightly less abstract terms, suppose we close switch A. Then side 1 is at $V+$ volts. If we close switch C, there is no voltage drop across the motor, so it is not activated. If we close switch D, then side 2 is at 0 volts, so the motor turns at its maximum speed. If we keep switch D closed half the time, then the motor integrates the voltage drop across it (which is $V+$ volts half the time, and 0 volts half the time) to yield an effective voltage drop of $V+/2$. If, on the other hand, we close switch B, side 1 is at 0 volts. Then, keeping switch C closed half the time will drive the motor *in the other direction* by an effective voltage drop of $V+/2$.

We are fortunate that we don't need to handle all this switching, because we are using the L293D driver chip, which decreases the amount of work we do by half. Referring to the data sheet, which is attached, pin 1 is an ENABLE signal. If it is set to logic low, the part does nothing, so it should be held at logic high always. Pins 2 and 7 are logic inputs that control the analog outputs 3 and 6 respectively. If you put logic high (5V) on pin 2, then pin 3 will be at VC (the analog power supply level). If you then put a pulse width modulated (PWM) signal on pin 7, you will drive the motor in one direction (call it clockwise) with a speed proportional to the low part of the PWM signal. If you hold pin 2 low (0V), then the PWM signal on pin 7 will drive the motor in the other direction (call it counter-clockwise) with a speed proportional to the high part of the PWM signal.

The other thing you need to know is that the digital outputs from the Stamp are apparently very weak driver signals. The Stamp has a PWM command that creates a noisy signal with a duty cycle approximately what you want it to be. However, that PWM signal can't drive the L293D, so you need to create your own PWM command. A good way is to use two output signals with the PULSOUT command. Consider the following bit of STAMP code:

```
SPEED con XX
onTime var word
offTime var word

onTime = 10*SPEED
offTime = 255-SPEED
offTime = 10*offTime
LOOP:
    pulsout 12,onTime
    pulsout 13,offTime
goto LOOP
```

The signal on pin 12 will be high for a time proportional to $SPEED/255$, corresponding to a duty cycle of $SPEED/255\%$. Of course, in reality, we can't quite get a 100% duty cycle, because the pin must be low for the time when the "goto" instruction is being executed (and because the Stamp is running an interpreter, that's a long time). Pin 13 is not actually being used here, but is a "dummy" pin for the pulsout command used to waste the right amount of time in which pin 12 is kept low. Could you do the same thing with the pause command?

Of course your program does not use only one speed, so it will look like:

```
speed var word
onTime var word
offTime var word

LOOP:
    'Calculate speed = new motor speed (0-255)

    onTime = 10*speed
    offTime = 255-speed
    offTime = 10*offTime
```

```

    pulsout 12, onTime
    pulsout 13, offTime
goto LOOP

```

Now all the time spent calculating `speed`, `onTime`, and `offTime` is time when pin 12 can't be high. This is why we use the (10*) multiplier--we try to define the PWM signal so the pulse width is proportional to a 5ms period (anything bigger than that and you'll start hearing "beats" in the motor). All the same, you probably won't be able to exceed a 50% duty cycle. (Incidentally, this would be the case even using the built in PWM command, since the PWM signal is only generated when nothing else is happening).

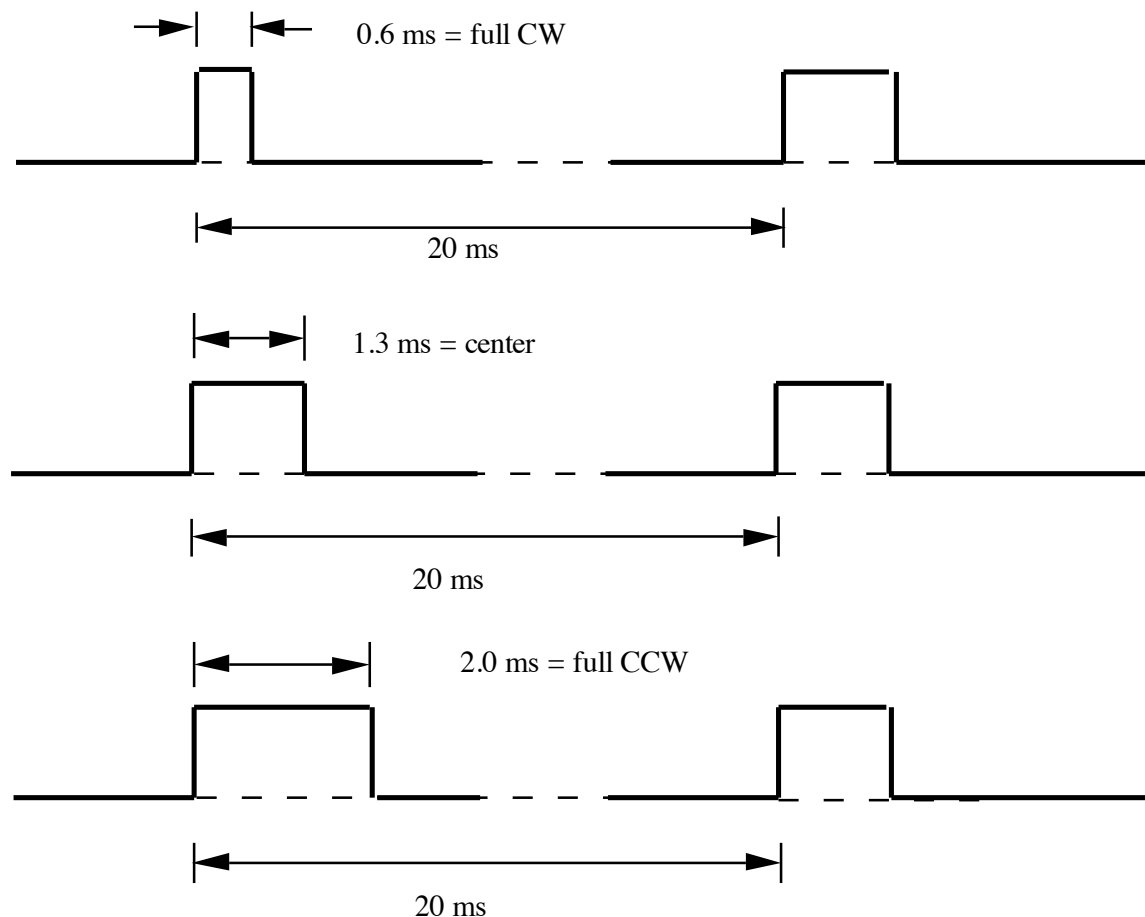
Now you are ready to proceed:

1. Wire the Stamp to read in the potentiometer, and display the value. You can use a 9V battery to power the stamp, or use the 12V power supply that is connected to the motor power amplifier. Be sure that the ground wire is connected to the ground wires of the power amplifier (this can be done at the potentiometer).
2. Connect the Stamp to the motor driver. Be sure to connect pins 4,5 to ground, pin 16 (Vss) and pin 1 (enable) to your digital 5V signal and pin 8 to your analog supply voltage (either the 9V battery, or an external DC supply). Then choose three Stamp output pins (call them Out1, Out2 and Waste). Connect Out1 to pin 2, Out2 to pin 7 and leave the third pin (Waste) as the "wasted" pin in the PWM scheme above. Finally, connect the motor between pins 3 and 6. Then by appropriately controlling the voltage on pins Out1 and Out2, and using the PWM scheme, you should be able to control both the direction and speed of the motor. You should probably experiment with the fixed speed program above (in both directions) until you are comfortable with this motor driving scheme.
3. Now you are ready to do something interesting. We will use the potentiometer to determine the speed and direction of the motor. To start, simply use the motor running in one direction, and scale the value of the potentiometer reading so that one end of the pot drives the motor at full speed and the other end stops the motor. Using the debug command, determine what the threshold (deadband) is for when the motor starts (i.e., is it greater than 0?).
 - Remove the debug command. Do you notice a change in the performance of the motor?
 - What is approximately the maximum speed of the motor? How can this be increased?
 - If you want a further challenge, write a program that will convert the potentiometer commands into a signal that will drive the motor CW or CCW, depending on the potentiometer position.
4. Finally, hook up one of the DC motors that is supplied with your kit, and experiment with driving it. These motors are not high performance (as you may have guessed), but can be used to give motion to your devices. It will probably be easiest to drive the DC motor with the gearhead on it, but you should explore using both motors.

8. Controlling an R/C servomotor

Lastly, we will utilize the R/C (radio-controlled) servomotor that is given in your bag of parts. This type of motor is commonly used in many hobbyist applications, such as remote controlled cars and planes. These motors are particularly useful for two reasons: (1) they have a built-in gearbox, so that the torque from the DC motor driving them is increased, and the speed is decreased; and (2) they have built-in position feedback control so that one can directly control the angular rotation of the motor, instead of only its speed.

There are three leads to the servomotor: power, ground, and control signal. The control signal is input on the middle lead, using 0/+5 V signals from the Stamp. As mentioned above, R/C servos are unusual in that they are position controlled. Thus, the input signal given to the servo controls must determine its position. This is done by sending the servo a signal that has a certain type of pulse in it. Generally, the basic signal is sent every 20 ms, and has a pulse length (where the signal is high) anywhere between approximately 0.6 and 2.0 ms. The position of the servo is dependent on the pulse length that is used:



Connect the servo to the stamp, and see how it works.

- What happens if you try to turn the servo away from the given position?
- Approximately how fast can you make the servo turn (say, in degrees/second)?

Report:

Write a short report based on your experiences with the BASIC Stamp. Discuss its advantages and its limitations. Draw a conclusion on how useful you believe the BASIC Stamp is in real life mechatronic applications. Include in your report the A/D converter graph, all your program code, and your answers to the posed questions.

Since you will be using the Stamp in your term-long team project, you should give some thought as to how it might work for you in your designs. It may also be quite informative to browse through the Stamp application notes to get some ideas. Although they were originally developed for the Stamp I, they can all be modified for use with the Stamp II.