ME 2110 Controller Box Manual

Version 2.3

I. Introduction to the ME 2110 Controller Box

A. The Controller Box
B. The Programming Editor & Writing PBASIC Programs
C. Debugging Controller Box Problems

II. Controller Box Accessories

A. Microswitches
B. Solenoids
C. Shape-Memory Alloy Linear Actuators (SMAs)
D. DC Motor
E. Stepper Motor
F. Flex Sensor
G. IR Distance Sensor

III. Programming Format

A. Recommended Programming Format
B. Labels
C. Loops
D. Subroutines

IV. PBASIC Commands

A. CON	Defines constants in a program
B. DEBUG	Prints information in the debug window
C. END	End of code
D. FOR…NEXT	Loop with counter
E. FREQOUT	Playing music
F. GOSUB…RETURN	Call a subroutine
G. GOTO	Jump to a location in the program
H. HIGH	Make the corresponding pin 5 volts
I. IF…THEN	Conditional branching command
J. LOW	Make the corresponding pin 0 volts
K. PAUSE	Place a delay in the program
L. PWM	DC motor speed control
M. TOGGLE	Toggle the state of a pin
N.VAR	Defining variables

V. Using the Controller Box Accessories & Sample Programs

A. Microswitches, Solenoids, & SMAs
B. DC Motor
C. Stepper Motor
D. Flex Sensor
E. Distance Sensor

I. Introduction to the ME 2110 Controller Box

A. The Controller Box

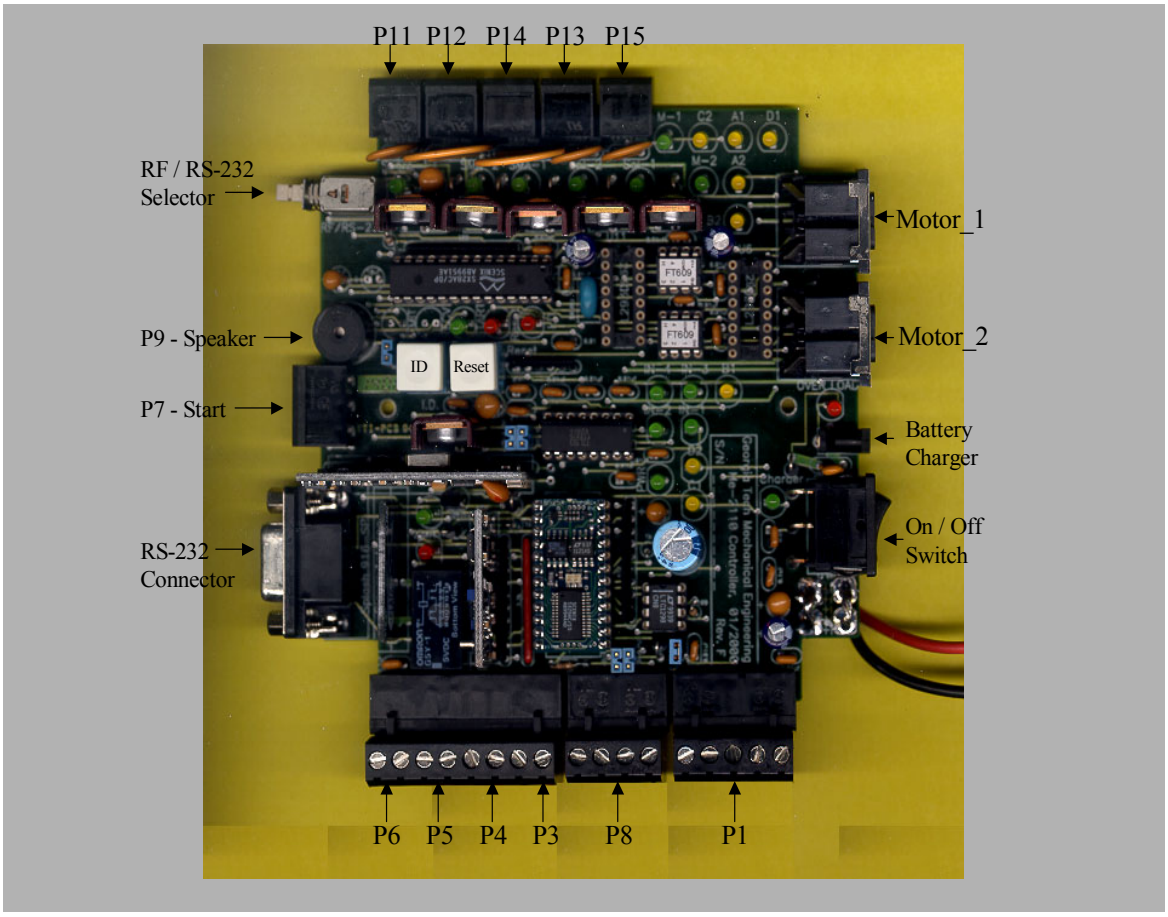

Figure 1. ME2110 Controller Box

Figure 1 shows an overhead picture of the controller box with the important features labeled. The cover should never be removed from the box due to the possibility of foreign material shorting the circuit board and destroying valuable equipment. The on/off switch is located on the right side of the box in Figure 1. When the box is turned on a green light should illuminate in the circuit board. The box is powered by 120 V AC.

The box can be programmed using a computer with a wireless radio frequency (RF) connection or a direct RS-232 cable, which has been provided. To select between the two modes of communication, a selector switch is located in the upper-left corner of the box. When the selector is pushed in, then the controller box is expecting RS-232

communication. For RS-232 communication, the RS-232 cable should be connected to the box and the COM1 port in the back of the computer.

In the top of the box there are two holes, one over the "ID" button and one over the "RESET" button. The ID button is used for RF communication and the RESET button performs the same action as turning the box off and on again.

In Figure 1, there are various locations denoted by the prefix "P." These are the locations where motors, solenoids, microswitches, and other accessories can be connected. The connectors are purposely made different sizes, so only the appropriate devices can be connected to certain locations. This will be discussed further in Section II.

Finally, there are two locations marked MOTOR. These are the locations where the stepper motors should be connected. The plugs are very tight, so it is important to ensure they are fully plugged in. Only about a centimeter of metal should be showing on the connector when fully plugged in.

B. The Programming Editor & Writing PBASIC Programs

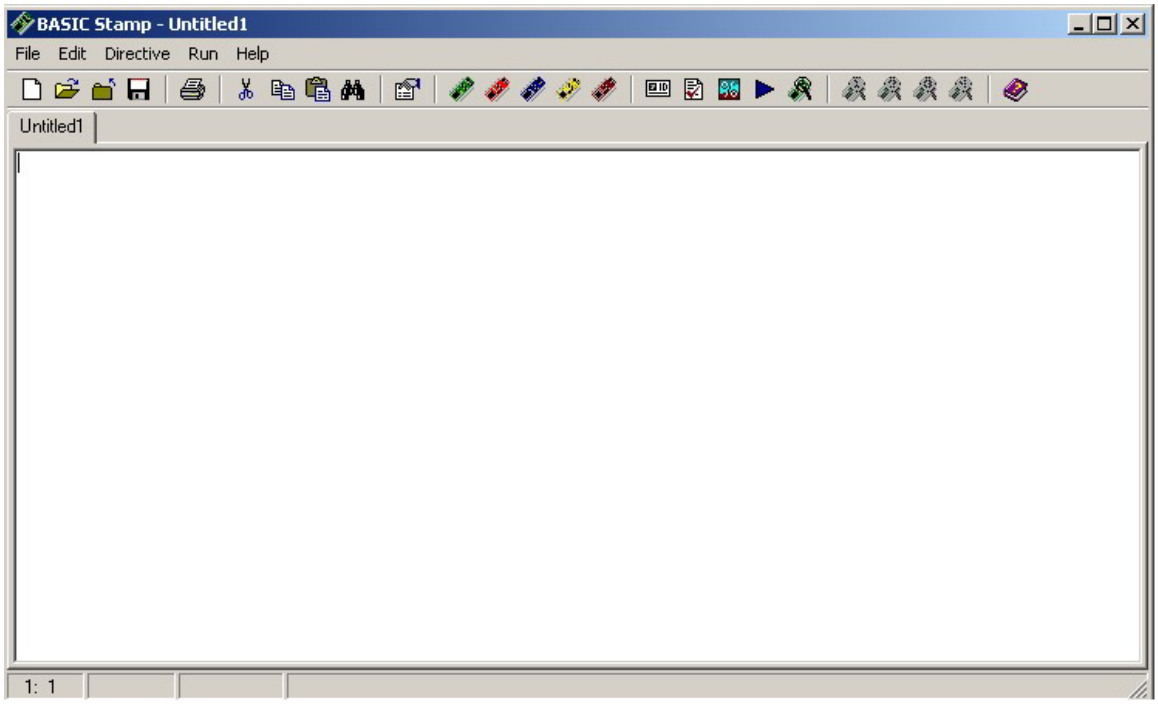

Figure 2. BASIC Stamp Programming Editor

Figure 2 shows the BASIC Stamp Programming editor. The filename for this program is ProgStamp.exe and is available on the class website and on the desktop of the computers in the ME2110 studio.

The editor window shown in Figure 2 should then appear. The toolbar contains many buttons used by other windows-based programs. Two additional buttons of interest are the "play" button, which is depicted by a triangle pointing to the right and the "debug" button, which is depicted as a magnifying glass over an electronics chip. These commands are also available in the "Run" menu. The editor is similar to a word processor. The program should be typed into the editor window and must be saved before downloading. When saving, make sure to save the program as a ".BSX" file, which is not the default.

Also the basic stamp BS2sx must be selected for the communication to work properly. Go to the menu Directive with the sub-menu Stamp and select the BS2sx chip. This will insert an initial line of code into your program.

Once the program is saved, the controller box should be activated, the selector switch pressed in for RS-232 communication, and then play button pressed in the Basic Stamp Programming editor. This will store the program in the box, so every time the box is turned on the program will execute, whether it is connected to the computer, or not.

C. Debugging Controller Box Problems

The following are some problems commonly encountered with the controller boxes and some possible solutions.

If you get the error: "Not detected on any COM port" check:

- Is the controller box is turned on?
- Is the selector switch is pressed in?
- Did you Add COM1 upon entering the editor program?

If you get the error: "Detected, but not responding," check:

- Is the file saved as a ".BSX" file?
- Did you add COM1 upon entering the editor program?

If you are having problems with your stepper motors, check:

- Are they fully plugged in?
- If so, when you run the program, are the 4 yellow lights in the controller box coming on?

II. **Controller Box Accessories**

A. Microswitches

The microswitches provided are simply two wires connected by a lever arm attachment. In its relaxed state, the two wires are in the "open" state, not connected. When the lever is pressed down, the two wires are "shorted" and a current flows. The microswitches can be connected to ports P3 – P7. When connecting the microswitches to the three-pronged plug for P7, the two wires can be attached to any of the three locations, as long as one of the wires is connected to the middle location.

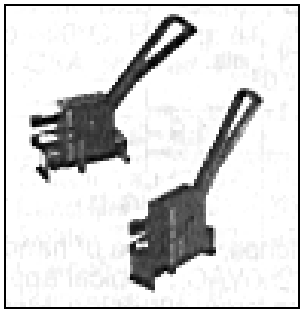

Figure 3. Microswitches

B. Solenoids

Figure 4 shows a cross-sectional view of a solenoid. The inside of a solenoid consists of a coil of wire which, when powered, acts like an electromagnet and pulls in the plunger. The solenoids provided can be connected to ports P11 – P15 on the controller box. Orientation of the wires is not important.

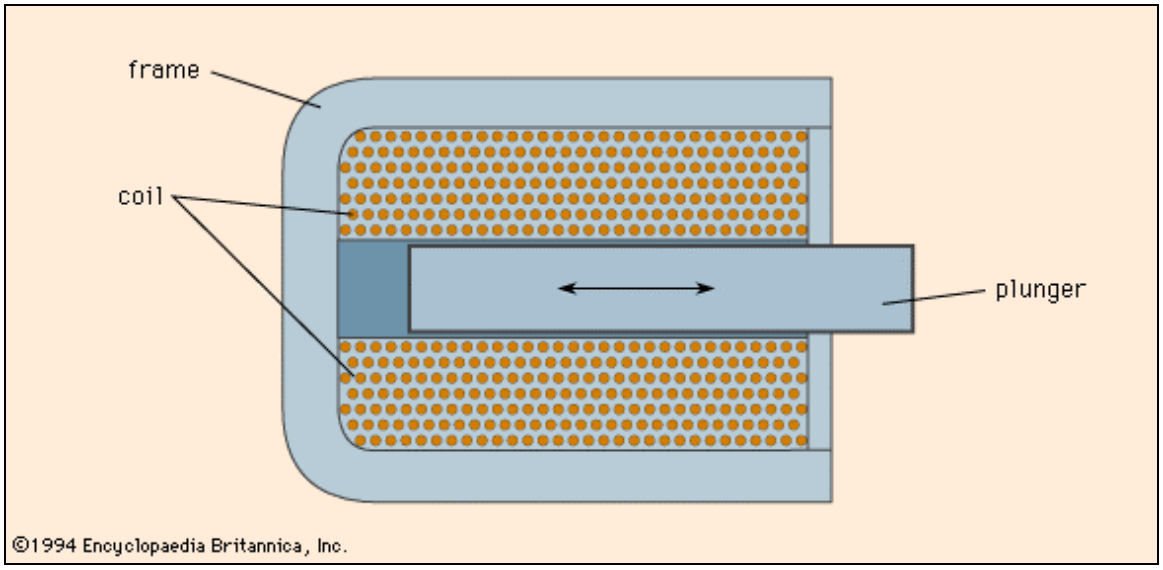

Figure 4. Cross-sectional view of a solenoid

C. Shape-Memory Alloy Linear Actuators (SMAs)

The SMAs perform the same action as the solenoids, but in a different manner. Shape-memory alloys are metal alloys that "remember" the shape that they were originally formed in and, when heated, return to this shape. The SMAs provided, shown in Figure 5, have a plunger like the solenoids. The plunger can be pulled out, which deforms the shape-memory alloy. When the SMA is activated, a current is sent through the coil, heating the metal, and the plunger is pulled in as the metal reverts back to its original shape. The coil must have a sufficient amount of time (several minutes) to cool down before it can be used again.

Figure 5. SMA Pistons

The SMAs can be connected to the controller box at the ports P11 – P15.

D. DC Motor

One of the two types of motors provided is a DC Brush Motor, shown in Figure 6. The motors also have a gearbox attached to the front of the drive shaft. These motors can be connected to ports P11 - P15. The motor will turn in one direction when the wires are connected one way, and if reversed, the shaft will move the opposite direction.

Figure 6. DC Brush Motor

The DC Motor works by inducing an electromagnetic field using coils of wire around an armature. Around the inside of the casing are permanent magnets, and as the shaft turns, the brushes reverse the polarity of the armature, causing the shaft to rotate. More information on how a DC Brush Motor works can be found at www.howstuffworks.com/motor.htm.

The controller box is only able to run the motor at 5 volts, which corresponds to its top speed. The speed of the DC Motor is proportional to the applied voltage, so in order to run the motor at intermediate speeds the voltage must be varied. This can be accomplished with the PWM command, which is explained in the PBASIC commands section of this manual.

The DC motor, although reliable, has several disadvantages. The motor can only rotate in one direction unless the wires are reversed. Also, the motor takes a short period of time to ramp up and ramp down before reaching top speed.

E. Stepper Motor

The second type of motor provided is a Stepper Motor, shown in Figure 7. A stepper motor works differently than the DC Motor and is preferable in certain situations. For example, the stepper motor can move in both directions and requires (effectively) no time to ramp up or down from its current speed. Also, the speeds are easier to control and are more repeatable than the DC Motor. A disadvantage of the stepper motor is that when turned off, the shaft is no longer held in place by the stator and is free to rotate.

Figure 7. Stepper Motor

Stepper motors work in a very different manner than the DC motors. The stepper motors provided have four wires rather than the two wires of the DC motors. The rotor of the motor is a permanent magnet that is rotated based on the polarity of the surrounding electromagnets (stator). As can be seen in Figure 8, which is a simple example of stepper motor control, the sequencing of these electromagnets and turns the shaft of the motor.

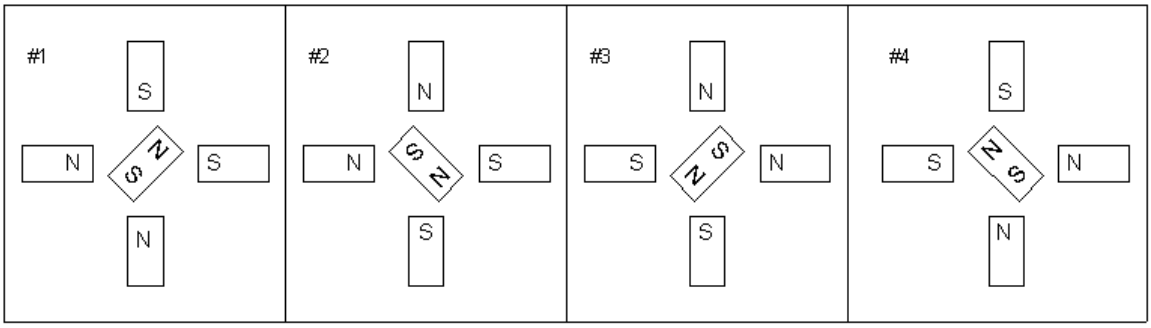

Figure 8. Stepper Motor Control

The stepper motors should be connected to the ports on the controller box labeled "Motor_1" and "Motor_2." Only one motor should be connected to each port or the circuit will pull too much current and will fail. As stated previously, the stepper motor plugs are hard to plug in. To ensure the plugs are fully attached, only a centimeter of metal should be showing on the plug.

F. Flex Sensor

Each group is provided with one flex sensor shown in Figure 9. This sensor can detect when it is bent and in what direction, much like a cat's whisker. The sensor works similarly to a strain gage. As the strip is bent, the copper wire is stretched and its resistivity changes. This change in resistance is sent through an A/D converter and converted to a number, which can be used to detect deflections. The flex sensor should be connected to P1, and will only work when connected to this port. The Flex sensor connects to the five pin connector and if they are numbered looking at the screws from left 1 to 5 right the two wires connect to connections 2 and 3.

Figure 9. Flex Sensor

G. IR Distance Sensor

Finally, each group is also given an infrared (IR) distance sensor, as shown in Figure 10. This sensor can measure the distance to the nearest object in the range of 4" to 31" from the sensor. This is accomplished by emitting an infrared beam. If there is an object in the detection range, the beam will be reflected off the object and detected by the receiver photodiode optics. An output voltage proportional to the distance of the object is sent to the A/D converter, which converts this voltage into a number between 0 and 255. More information can be found at http://www.hvwtech.com/irods.htm#how

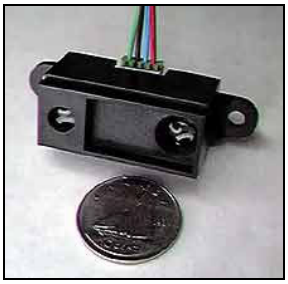

Figure 10. IR Distance Sensor

It is important to note that this voltage is not linearly proportional to distance. Therefore, in order to determine whether an object is a given distance away, a test should be performed to determine the number between 0 and 255 which corresponds to this

distance, and this number should be used to compare with the actual distance value measured by the IR sensor.

The IR sensor is connected using a four pin connector the wires should be placed in the following order: yellow red green black, when looking at the screws from left to right. The distance sensor should be connected to P8, and will only work when connected to this port.

III. Programming Format

A. General Programming Format

When programming the Basic Stamp, the following programming sequence is suggested:

- Constants Table
- Variable Definitions
- Main Program Loop
- Subroutines

Before variables and constants can be used, they must be defined, so these actions should be performed at the beginning of the program. Subroutines are typically placed at the end of the program in order to set them apart from the main program loop. Although not mandatory to program the Stamp, using this programming format will help ensure the correct flow of the program and allows for easier debugging.

B. Labels

Labels are like "bookmarks" in a program. They don't perform any function; they simply mark a specific location in a program that can be referenced by other commands. Labels are primarily used by the GOTO, IF…THEN, and GOSUB…RETURN functions.

Labels are denoted by a word followed by a colon. An example of the use of a label is shown in the code below. Note: Reserved words cannot be used as labels.

```
Label:
        DEBUG "The label marks a place in a program", cr
        GOTO Label
```

C. Loops

Loops are used to perform the same task multiple times. There are two types of loops: variable-length and fixed-length.

Variable-length loops are used when the number of loop iterations is not known or when the program needs to remain in the loop until an external event occurs. These types of loops are created using the GOTO command and a label. The following program shows how variable-length loops can be implemented in a program.

```
Label:
        IF Button_State = 1 THEN Exit_Loop
GOTO Label

Exit_Loop:
```

Fixed-length loops are used when the number of iterations is known, such as when an action needs to be performed a set number of times. These types of loops are created using the FOR…NEXT command. An example is shown below.

```
Counter VAR byte

FOR Counter = 1 TO 10
        DEBUG dec Counter, cr
NEXT
```

D. Subroutines

Subroutines are used to organize and simplify the execution of a program. A subroutine should perform a complete task such as sort a list, run a motor in a certain sequence, or measure the distance to an object. Subroutines are often used several times in a program, at different locations in the code. This helps reduce the size of the program by not having to write the same exact code several times in the program.

Subroutines are created using the GOSUB…RETURN command. One of the advantages of using subroutines is that the RETURN command jumps back to the line after the original GOSUB command, eliminating the need for another label in the main program. An example of this method is shown in the code below. The subroutine turns a solenoid located at P11 on for one second and then turns it off again.

```
GOSUB Run_Solenoid
DEBUG "The solenoid was just turned on for one second", cr
GOSUB Run_Solenoid
DEBUG "The solenoid was again turned on for one second", cr
END

Run_Solenoid:
        HIGH 11
        PAUSE 1000
        LOW 11
```

IV. PBASIC Commands

A. CON – Defining Constants

Constants are used to identify what numbers in a program mean. They are primarily used to make the code easier to follow. For example, the constant "cards_in_a_deck" could be used to represent the number 52. These differ from variables because constants do not change during the execution of the program. Constants are typically defined at the top of a program before the variable declarations.

Another advantage of using constants is in the case of code modification. If the same number is going to be used several times in a program and later it is discovered that the value must be changed, it is easier to change the value of the constant at the top of the program than to change every instance of the number in the program.

Constants are defined using the CON command. The following is an example of the use of constants in a program.

```
Cards_in_a_deck    CON    52
Counter  VAR byte

FOR Counter = 1 TO Cards_in_a_deck
        DEBUG "Dealing Card", dec Counter, cr
NEXT
```

B. DEBUG – Printing to the Debug Window

The DEBUG command is used solely for the purpose of observing the execution of a program; it is not able to change values of variables or affect the flow of a program. It can be helpful in finding the bugs in a program or ensuring that the program is working as planned. The debug window can be opened several ways. In the Basic Stamp editor there is a button on the toolbar that looks like an electronics chip covered by a magnifying glass, as shown in Figure 11, below. If you click on this icon, the debug window will open. The debug window can also be opened by selecting "Debug-New" from the Run menu. The debug window will also automatically open when the program encounters a DEBUG command.

Figure 11. Open Debug Window Button

The DEBUG command controls the output to the debug window. This command can print text and the values of variables, as well as control the output screen appearance.

Command	Description
DEBUG “Insert Text Here”	Prints the string between the quotes to the debug window.
DEBUG dec variable	Prints the value of “variable” to the debug window. “dec” prints the value as a decimal number, so this parameter should never change.
DEBUG cr	“cr” is the carriage return parameter. It moves the cursor to the next line in the debug window.
DEBUG cls	Clears the debug window screen.

A number of these commands can be placed in one line of code. In order to place multiple commands together, separate the commands by a comma. An example of using the DEBUG command is show below.

```
Variable VAR byte
Variable = 27

DEBUG cls, “The value of the variable is: ”, dec Variable, cr
DEBUG “This is the second line”
DEBUG “Variable =”, dec Variable

END
```

The following is the resulting output in the debug window:

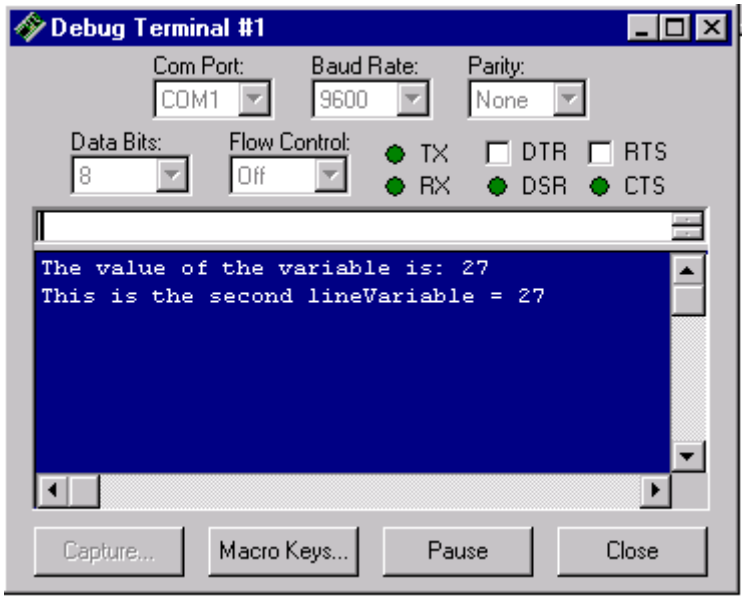

Figure 12. Debug Window Output

C. END – Denotes the End of a Program

The END command is similar to that used in other programming languages. Its sole purpose is to mark the end of a program. This command can be strategically used, though, by placing it in useful locations.

For example, when using the recommended programming format, the end command is necessary to stop the program from continuing and running the subroutines placed after the main program loop. The END command can also be placed within loops to stop the program if a designated condition is satisfied.

D. FOR…NEXT – Looping with a Counter

As stated in the looping section of this manual, the FOR…NEXT command is used when the number of loops is known. In order to use this command, a counting variable will need to be defined at the top of the program.

In the following program, the variable *Counter* is defined to be of size *byte*, which means that its values can range from 0 to 255. The FOR…NEXT loop in this program varies *Counter* from 1 to 10, increasing its value by one after each iteration of the loop and then printing this value to the Debug screen. If needed, the value of *Counter* can be varied a number of different ways instead of incrementing by one after each iteration of the loop. This information can be found in the BASIC Stamp Manual on the Parallax website.

```
Counter VAR byte

FOR Counter = 1 TO 10
        DEBUG dec Counter, cr
NEXT
```

E. FREQOUT – Playing Music

The FREQOUT command can be used to emit tones or play songs through the speaker on the controller box. The following table is a key to the some of the common notes available.

Frequency	Note
440	A
500	B
523	C
587	D
659	E
740	F
784	G

The syntax for the FREQOUT command is:

FREQOUT 9, duration, frequency

In this command line, the '9' is the pin on the BASIC Stamp microcontroller chip that is connected to the speaker, so this number will never change. The duration is the number of 0.4 millisecond time units that the tone will be played. The frequency is a number between 0 (silence) and 32767 corresponding to the frequency of the note played, such as the numbers in the table above.

The following shows how this command is used in a program:

```
FREQOUT 9, 250, 440     'Plays an A note for 0.1 seconds
FREQOUT 9, 500, 523     'Plays a C note for 0.2 seconds
```

For more information on how to play music, see the BASIC Stamp Manual on the Parallax website.

F. GOSUB…RETURN – Calling Subroutines

As stated in the "Programming Format" section, the GOSUB…RETURN pair of commands is used to jump to a subroutine, run the subroutine's code, jump back to the main program, and then perform the next command in the main program. Details on the use of this command can be found in the "Programming Format" section of this manual.

G. GOTO – Jump to a Different Program Location

The GOTO command is used to jump to the location in the program marked by the designated label. This command can be used to create a loop of indefinite length or simply to navigate through existing code.

A label is simply a word followed by a colon. It performs no function; it simply serves a "bookmark," marking a location in a program. At any time during the execution of a program, the GOTO command can be used to jump to a different location marked by a label.

The syntax for the GOTO command is simply the command "GOTO" followed by the name of the label to jump to. An example is shown below:

```
Label:
        DEBUG "Inside the loop"
GOTO Label
```

H. HIGH – Make the Corresponding Pin 5 Volts

The HIGH command can be used to activate several devices, such as the solenoids and SMAs. It also can be used to run the DC motors at its top speed. The HIGH command sends five volts to the specified pin.

The syntax for this command is:

HIGH pin

The pin is the number of the pin that is being controlled. An example of the use of this command is shown below:

```
HIGH 11        'Activates the device connected to P11
HIGH 15        'Activates the device connected to P15
```

I. IF…THEN – Conditional Branching

This command is similar to IF…THEN commands used in other programming languages. This command is used when an action needs to be performed if a certain condition is met. The available conditions are shown in the following table.

Symbol	Meaning
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

The syntax for this command is:

IF *condition* THEN *label*

In this line of code, the condition includes one of the above symbols and is evaluated to the true or false. If the condition is true, then the program jumps to the location marked by the specified label. Two or more conditions can be included in one IF…THEN statement by using the NOT, AND, OR, and XOR operators.

Two examples of the IF…THEN command are provided below:

```
Label:
        IF Button_State = 1 THEN Exit_Loop
GOTO Label

Exit_Loop:
```

```
Counter VAR byte

FOR Counter = 1 TO 1000
        IF Button_State = 1 AND Counter > 50 THEN Exit_Loop
NEXT

Exit_Loop:
```

J. LOW – Make the Corresponding Pin 0 Volts

The LOW command can be used to deactivate several devices, such as the solenoids, SMAs, and DC motors. The LOW command sends zero volts to the specified pin.

The syntax for this command is:

LOW pin

The pin is the number of the pin that is being controlled. An example of the use of this command is shown below:

```
LOW 11      'Deactivates the device connected to P11
LOW 15      'Deactivates the device connected to P15
```

K. PAUSE – Placing Delays in a Program

The PAUSE command suspends the execution of a program a specified amount of time. This command is can be useful in controlling the timing of a program. This command does not deactivate the box, but simply pauses the controller from executing any for a given time interval. For example, if a motor is turned on and the program encounters a five-second PAUSE command, the motor will continue to run for these five seconds.

The syntax for the PAUSE command is as follows:

PAUSE *delay_time*

In this command line, the delay_time is the amount of time (in milliseconds) the program should be paused in its current state.

An example of a program using the PAUSE command is provided below:

```
HIGH 15     'Activates the device connected to P15
PAUSE 5000  'Pause 5000 milliseconds, device continues to run
LOW 15      'Deactivates the device connected to P15
```

L. PWM – DC Motor Speed Control

The PWM command, which stands for Pulse Width Modulation, is used to vary the average voltage at a port on the controller box to a value between zero and five volts. This command can be used to run the DC motor at speeds slower than its top speed.

The PWM command works by quickly switching a port on and off, over a number of cycles. The syntax for this command is:

PWM *pin, duty, cycles*

pin: The pin number of the port to control.
duty: A number between 0 and 255 that specifies the analog voltage level (0-5volts). The larger the number, the larger the voltage and the faster the motor. 255 = top speed.
cycles: A number between 0 and 255 that specifies the number of 0.4-millisecond time units that the motor should be run. If the motor needs to be run for a longer duration, it is recommended that *cycles* is set to 250 = 0.1 seconds and placed in a loop.

It is important to note that the speed of the DC motor is not linearly proportional to the average voltage. Values of *duty* below 100 barely move the motor. Also, the DC motor requires a ramp-up and ramp-down time, so the time that is specified in the *cycles* parameter will not be exact.

An example of the PWM command is shown below:

```
Num_Cycles VAR byte

FOR Num_Cycles = 1 TO 10
        PWM 11, 220, 250          'Runs the motor for 10*0.1 second = 1 second
NEXT
```

In the above code, the motor, located at P11, is run at speed 220 for 0.1 seconds. Because this PWM command is placed in a loop, the motor actually runs for 1 second because it performs this command 10 times.

M. TOGGLE – Toggle the State of a Pin

The TOGGLE command determines the state of a pin and reverses it. So, if a pin is "high" and a TOGGLE command is sent, it will be made "low." Similarly, if a pin is "low" and a TOGGLE command is send, it will be made "high."

The syntax for the command is similar to the HIGH and LOW commands. An example is provided below.

```
HIGH 15         'Activates the device connected to P15
TOGGLE 15       'Deactivates P15, P15 is now "low"
TOGGLE 15       'Activates the device connected to P15, P15 is now "high"
```

N. VAR – Defining Variables

A variable must be defined before it can be used. In order to define a variable, the size must be specified. The possible sizes are in the following table.

Size	**Number Range**
Bit	0-1
Nib	0-7
Byte	0-255
Word	0-65535

To define a variable, use the following format:

variable_name VAR *size*

The following provides some examples of defining variables:

```
Counter VAR byte        'Counter can be a number between 0 and 255
Switch  VAR bit         'Switch can be either 0 or 1
```

Note: If a variable is assigned a value outside of its specified range, the editor will not provide an error message, but the program will not work properly and my cause problems.

V. Using the Controller Box Accessories & Sample Programs

A. Microswitches, Solenoids, & SMAs

The microswitches provide an input signal in one of two possible states. A "1" corresponds to a closed circuit, the condition when the lever arm is pressed, while a "0" corresponds to an open circuit, which is the condition when the lever arm is not pressed.

Each pin location has a corresponding register that holds the state of that port, either a 0 or a 1. In order for the BASIC Stamp to determine the state of a switch, the corresponding register should be read. The register for P6 is IN6, for P5 is IN5, etc. In order for a program to act upon the state of the switch, and IF…THEN command should be used. An example of this is below:

```
Loop:                                   'The program says in this loop until IN3 = 1
        IF IN3 = 1 THEN Out_of_Loop     'If switch on P3 is pressed, jump out of loop
GOTO Loop

Out  of  Loop:
```

The solenoids and SMAs are controlled the same way. To activate the devices, a "HIGH" command should be sent to the corresponding port. To deactivate the devices, a "LOW" command should be sent to the corresponding port. An example of how to control these devices is below:

```
HIGH 11                                 'Turn on device at P11

Loop:                                   'The program says in this loop until IN3 = 1
        IF IN3 = 1 THEN Out_of_Loop     'If switch on P3 is pressed, jump out of loop
GOTO Loop

Out_of_Loop:
LOW 11                                  'Turn off device at P11
END
```

B. DC Motor

The speed of a DC motor is proportional to the voltage supplied to the motor. At zero volts the motor is stationary and at five volts the motor runs at its maximum speed. The PWM command is used to vary the voltage. The syntax for this command is:

PWM *pin, duty, cycles*

pin: The pin number of the port to control.

duty: A number between 0 and 255 that specifies the analog voltage level (0-5volts). The larger the number, the larger the voltage and the faster the motor. 255 = top speed.

cycles: A number between 0 and 255 that specifies the number of 0.4-millisecond time units that the motor should be run. If the motor needs to be run for a longer duration, it is recommended that *cycles* is set to 250 = 0.1 seconds and placed in a loop.

It is important to note that the speed of the DC motor is not linearly proportional to the average voltage. Values of *duty* below 100 barely move the motor. Also, the DC motor requires a ramp-up and ramp-down time, so the time that is specified in the *cycles* parameter will not be exact.

In order to run the DC motor continuously, this command can be placed in a generic loop. An example is below:

```
Loop:
        PWM 11, 230, 250        'Each iteration of the loop runs the motor for 0.1 seconds
        GOTO Loop
```

If the DC motor needs to be run for a specific amount of time, the PWM command can be placed in a FOR…NEXT loop:

```
Num_Cycles VAR byte

FOR Num_Cycles = 1 TO 10
        PWM 11, 220, 250        'Runs the motor for 10*0.1 second = 1 second
NEXT
```

C. Stepper Motor

The Stepper Motors are controlled through a series of serial commands. These commands are translated and transmitted to the stepper motor drivers. Both of the motors are controlled with the same pin, so it is important that the correct motor is selected before an action is performed. Once a motor has been selected, it remains selected until another is selected.

In order to move a motor for the first time, the following actions must be performed (and in this order):

1. Select Motor
2. Set Direction
3. Set Motor Speed
4. Send Go Command

The syntax for the stepper motor commands is:

SEROUT 10, 17405, 50, [*command1, command2, command3, ...*]

In this command line, everything before the brackets will remain the same. These numbers specify the serial transmission parameters. The commands to the motors are sent within brackets. These commands are shown in the table below:

Command Number	**Action**
1-240	Motor Speed (Smaller is Faster)
250	Motor 1 Select
251	Motor 2 Select
252	Stop
253	Go
254	Clockwise
255	Counter-Clockwise

The following is an example of how to use the stepper motor SEROUT command:

```
SEROUT 10, 17405, 50, [250,254,5,253]   'Run Motor 1 CW at speed 5
PAUSE 5000                              'Run motor for 5 seconds
SEROUT 10, 17405, 50, [252]             'Stop current motor (Motor 1)

SEROUT 10, 17405, 50, [251,255,10,253]  'Run Motor 2 CCW at speed 10
PAUSE 5000                              'Run motor for 5 seconds
SEROUT 10, 17405, 50, [252]             'Stop current motor (Motor 2)
```

A few final notes:

- The motor must be stopped before changing directions
- The motor speed can be changed without stopping the motor

D. Flex Sensor

The Flex Sensor acts similarly to a cat's whisker and is able to detect when it is touching an object. As the sensor is deflected, its resistance changes and the direction and amount of deflection can be calculated based upon the output from the A/D converter.

Some sample code for using the Flex Sensor is available on the class website under "FlexSensor.bsx." This program outputs the value of the flex sensor to the debug window. The variable that stores the value from the flex sensor is "flex_value." Based on the value of this variable, programs can be written to perform a variety of actions.

In order to implement the flex sensor in future programs, the exact same subroutine should be placed in the new program, and calls should be made to the subroutine to determine the value of the flex sensor.

E. Distance Sensor

The Infrared (IR) Distance Sensor can be used to detect objects within a given range, about 4" to 31" from the sensor. The distance sensor should only be plugged into P8, and will not work in any other port.

When a reading from the distance sensor is needed, the following occurs:

1. A 'Low' signal is sent to the clock – this turns on the A/D converter and tells it to take a reading.
2. An analog voltage is read from the distance sensor by the A/D converter, which converts this voltage to a number between 0 (0 volts) and 255 (5volts).
3. Once a reading has been taken, pin 8 goes 'high.' This means that the value is ready from the A/D converter.
4. The value is 'shifted' in from the A/D converter one bit at a time
5. A 'high' signal is sent to the clock, which turns off the A/D converter.

As with the flex sensor, the readings from the distance sensor decrease with distance, but are not linear. If the object is closer than the minimum range of the sensor, the reading will not be valid.

A sample program for the distance sensor is available on the class website, called "DistanceSensor.bsx." This program reads in a value from the distance sensor, stores the value in the variable "dist," and displays the value in the debug window. As with the flex sensor, the subroutine in the sample program should be placed in future programs, and calls should be made to this subroutine to determine the distance value.