Using Software to Enhance Student Learning

Mike J. Metaxas, PE
Queensborough Community College
Department of Engineering Technology
222-05 56th. Avenue
Bayside, NY 11364
718-631-6207
mmetaxas@qcc.cuny.edu

ABSTRACT

1.1.1 In this paper, we expand on the benefits of using hardware and software in project based learning (see my previous paper ***Integrating Hardware and Software In Project Based Learning***[1]) and break down the skills required and learned in successfully completing a microcontroller based project. Some of these skills include hardware assembly (soldering, de-soldering, prototyping), software development (system analysis, software development, flowcharting and BASIC language programming) and troubleshooting (using DMM's and oscilloscopes).

Categories and Subject Descriptors

B.0 [**General Hardware**], B.6 [**Logic Design**], D.1 [**Programming Techniques**]: D.3.3 [**Programming Languages**]:

General Terms

Algorithms, Measurement, Design, Experimentation, Languages, Theory. Troubleshooting

Keywords

Microcontroller, PicAxe,

1. INTRODUCTION

In my Electronic Projects Class, the students physically build and program a microcontroller based project. The building, programming and troubleshooting of this board provides a means for the students to hone their skills. By incorporating Project Based Learning concepts and including both software and hardware into this project the student enhances his physical skills, his critical thinking skills, and his programming skills – all of which are in high demand by employers.

2. PROJECT BASED LEARNING

Project based learning incorporates a hands-on approach to understanding fundamental concepts and relationships.

[1]Professor. Mike J. Metaxas, Queensborough Community College

I have attempted to include the following project based learning objectives[2] into my class.

1. Provide "Real" world examples with positive feedback
2. Encourage the use of higher order thinking skills and learning concepts as well as basic facts
3. Utilize a hands-on approach
4. Provide for in-depth understanding
5. Promote meaningful learning by connecting new learning to students' past performances
6. Learning utilizes real time data by investigating results and drawing conclusions
7. Learning cuts across curricular areas and is multidisciplinary in nature

3. INTEGRATED CIRCUITS[3]

Integrated circuits are quite common. Some really small ones may have just three leads while others may have hundreds of leads. Some may cost just a few pennies while others may cost hundreds of dollars. We often abbreviate the name integrated circuit to "*IC*" and frequently refer to an IC as a chip.

ICs come in many types. Some may be amplifiers, some may regulate a voltage, and some may generate or detect specific signals. One large category of ICs is *microprocessors* or *microcomputers.*

4. MICROCONTROLLERS

In general, microcontrollers are designed to do specific tasks and typically operate very slowly as compared to general purpose microprocessors. For instance, your cell phone has a small computer inside. So does the microwave oven in your home. So does a CD player or a digital watch.

Because this type of processor doesn't need much memory or I/O it often has these built into the same IC as opposed to general purpose microprocessors which access to Gigabytes of memory using external interface chips.

[2]Based on an article written by Nancy Kraft, formerly of RMC Research Corporation, Denver, Colorado titled *"Criteria for Authentic Project Learning"*

[3] Professor Peter A. Stark, Queensborough Community College

The result is then called a *microcontroller* since it contains on a single chip most if not all of the basic components required to run a program including RAM, ROM and physical I/O pins.

Many modern devices, for example Microwave Ovens, Refrigerators, Stoves, Blenders, Thermostats etc., incorporate Microcontroller chips in their design. The inclusion of a Microcontroller allows a manufacturer to change the way a device operates by changing the software in the chip rather than redesigning the hardware in the controller board. By introducing these concepts to our students we expose them to "real-world" problems and solutions in order to better prepare them for the demanding requirements of the workplace.

These smaller computers are generally called *embedded* computers. That is, they are *built into* or *embedded* in some other piece of equipment. They are not general purpose; instead, they are programmed to do a specific job, day in and day out as part of some other piece of equipment. For example, the microcomputer in a washing machine has a bunch of inputs from a keyboard or switches on the control panel, and a bunch of outputs to the motors and valves. In response to your commands from the keyboard, it decides what to do when, and then sends output signals to the various motors and valves to do your wash.

Microcomputers come in various sizes, prices, and abilities. The one we use in our lab is called a *Picaxe*[4]. Picaxe's also come in various sizes. The one we use is the model 08M - it has just 8 pins. Some of the larger microcomputers have 14, 18, 28, or even 40 pins. Obviously the larger ones can do more. But even with just 8 pins, ours is a complete little computer.

5. THE PICAXE[5]

The Picaxe system exploits the unique characteristics of the new generation of low-cost 'FLASH' memory based microcontrollers. These microcontrollers can be programmed over and over again (typically 100,000 times) without the need for an expensive programmer.

Revolution Education, the manufacturer of the PicAxe provides a free suite of programming tools specifically written for the PicAxe and the students are encouraged to download the tools to their home computers. These tools include a visual programming tool which is geared for younger students and the powerful PicAxe Programming Editor.

The Picaxe uses a simple version of the BASIC programming language that enables students to start generating programs within an hour of their first exposure. It is much easier to learn and debug than industrial programming languages (C or assembler code).

The power of the Picaxe system is its simplicity. No programmer, eraser or complicated electronic system is required - the microcontroller is programmed via a 3-wire connection to the computers serial port. The Picaxe download circuit uses just 2 components and can be easily constructed on a prototyping breadboard or printed circuit board.

[4] Picaxe® is a registered trademark licensed by Microchip Technology Inc.

[5] Portions reprinted from the Revolution Education PicAxe Manual

If you wish to make your own PCB some reference designs are available at the PCB section of the Picaxe website at www.picaxe.co.uk PCB samples are available for educational use in the popular realPCB and PCB Wizard formats.

6. THE PROJECT

The project consists of building and programming a microcontroller.

Before they actually build the microcontroller the students are required to build and test a serial programming cable for the device. They are given a schematic diagram of the programming cable and must correctly solder the two different connectors to the cable. Invariably they make mistakes and must de-solder and re-solder the wires to the correct pins on the connectors.

Successful completion of the cable is crucial because it is used in the remaining lab activities.

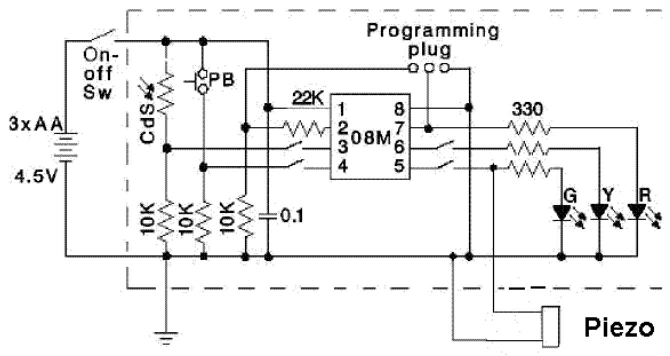
At this point, an initial lecture explaining the concepts and capabilities of microcontrollers is given and each student is required to solder and test a AXE092 student experimenter's board. This board is a complete, functioning microcontroller with LED's for outputs, a light dependent resistor (LDR) for analog input and a pushbutton switch for digital input.

Although small and relatively inexpensive, this board provides a powerful platform for explaining and experimenting with the microcontroller – specifically Analog and Digital inputs, Digital Output and Pulse Width Modulation. The student experimenter's board is pictured below.

Students are provided with extensive documentation, the "ET410 Laboratory Manual"[6] which contains detailed assembly instructions, illustrations, board level testing hints, and programming examples. Students are urged to ask questions before actually building the projects in order to prevent errors and rework.

Students are given a lecture incorporating the fundamentals of programming, the BASIC programming language, and good programming practices. Throughout the manual there are many sample programs available.

[6] Professor Mike J. Metaxas, Queensborough Community College



7. PICAXE INPUTS AND OUTPUTS

The PicAxe provides the following type of inputs and outputs:

7.1 Digital Outputs

The microcontroller can sink or source 20ma on each output pin, maximum 90mA per chip. Therefore low current devices such as LEDs can be interfaced directly to the output pin. Higher current devices can be interfaced via a transistor, FET or Darlington driver array.

7.2 Digital Inputs

Digital input switches can be interfaced with a 10k pull down resistor. The resistor is essential as it prevents the input 'floating' when the switch is in the open position which would give unreliable operation. Note the 10k resistors are pre-fitted to the project board inputs.

7.3 Analog Inputs

Analog inputs can be connected in a potential divider arrangement between V+ and 0V. The analogue reference is the supply voltage, and the analog signal must not exceed the supply voltage.

8. PROGRAM 1

Students are given the following program to flash an LED and asked to download it to their Picaxe and explain its operation. Note that I/O #0 on their Picaxe board is connected to the on-board Red LED:

```
main:
        high 0
        low 0
        goto main
```

This program illustrates the concept of using an output device, labels, and looping. After loading the program most students say that the device is not working correctly because all they see is the Red LED on.

We then discuss the sequential operation of programs and the execution time of the microcontroller. Since they are melding hardware and software I encourage them to connect an oscilloscope to the LED so that the can see it turning on and off. We then change the program to insert a delay as shown:

```
main:
        high 0
        pause 500
        low 0
        goto main
```

Once again most students say that the device is not working and I ask then to follow the program step by step and explain what is going on. They are encouraged to use the oscilloscope if needed. The next step is to add the off time delay as shown below:

```
main:
        high 0
        pause 500
        low 0
        pause 500
        goto main
```

At this time they have a working program and I introduce the concepts of comments and symbolic references (or aliases) and end up with the final version shown below:

```
‘ Picaxe program to flash the red led
Symbol RED_LED = 0

main:
        high RED_LED     ‘  red LED on
        pause 500        ‘ wait for 500 msec
        low RED_LED      ‘  red LED off
        pause 500        ‘ wait for 500 msec
        goto main        ‘  jump back to start
```

9. PROGRAM 2

Students are given the following program and asked to explain its operation and to add the appropriate comments and to add aliases for all three LED's and for the delay time

```
main:
        high 0
        pause 500
        low 0
        high 1
        pause 500
        low 1
        high 2
        pause 500
        low 2
goto main
```

The expected result is shown below

```
‘ Picaxe program to blink each LED for 500 msec
‘ make sure switches 1 & 2 are on

Symbol RED_LED = 0
Symbol YELLOW_LED = 1
Symbol GREEN_LED = 2

Symbol WAIT_TIME = 500
main:
        high RED_LED             ‘ red LED on
        pause WAIT_TIME          ‘ wait 500ms
        low RED_LED              ‘ red LED off
        high YELLOW_LED         ‘ yellow LED on
        pause WAIT_TIME         ‘ wait 500ms
        low YELLOW_LED          ‘ yellow LED off
        high GREEN_LED          ‘ green LED on
        pause WAIT_TIME         ‘ wait 500ms
        low GREEN_LED           ‘ green LED off
        goto main               ‘ jump back to start
```

10. PROGRAM 3

Students are given the following program and asked to explain its operation and to add the appropriate comments and to add aliases as necessary

```
main:
        If pin3 = 1 then
                high 0
        else
                low 0
        EndIf
        goto main
```

This is the expected result

```
Symbol PUSH_BUTTON = pin3
Symbol RED_LED = 0

main:
        If PUSH_BUTTON = 1 then
                high RED_LED    ‘ turn on the red led
        else
                low RED_LED     ‘ turn off the red led
        EndIf

        goto main
```

In addition, the following assignments are given:

- Modify the above program so when the switch is pressed the red LED is on and the yellow LED is off and when the switch is released the yellow LED is on and the red LED is off. Be sure to use symbols and comments.
- Modify the above program so that the piezo plays a sound when the button is pressed. Hint: look up the sound command in the Picaxe programming manual. What else happens when you run the program?

11. PROGRAM 4

Now that we can read the switch, how can we read the Light Dependent Resistor (LDR)? Since the voltage across the device will change based on the amount of light we need to use an *Analog to Digital Converter* (ADC). The easiest way to experiment with an ADC is by using the built in debug facility.

Students are given the following program:

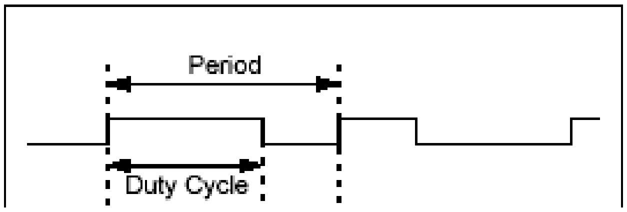
```
Symbol LDR = 4

Main:
        readadc LDR, b0    ‘read the analog voltage into b0
        debug
        pause 100          ‘ wait for 100 msec
goto main
```

When you run the program a debug window will open and you can see the value of the analog input voltage by looking at variable b0.

Students are given the following assignments:

- Now that you know how your individual LDR works you can write a program that will light the red LED if it is bright, the yellow LED for ambient light and the green LED when the room gets dark. Make sure to use and use debug in case your program is not working as expected.
- Expand the previous program to play tones in addition to lighting a LED. Should every condition play a tone? What happens to the green LED?
- Expand the ADC resolution by using the readadc10 command.
- After a brief lecture on pulse width modulation students are asked to use the pwmout command to vary the intensity of the green LED based on:
    1. The switch being pressed. Rotate between 3 distinct values – dim, normal, and bright – each time the switch is pressed.
    2. The light intensity detected by the LDR. The intensity should be continuously variable from dim (when the detected light is low) to bright (when the detected light is bright.



Note that the pwmout command uses the internal pwm module of the microcontroller and there are certain restrictions to its use:

1. The command only works on certain pins.
2. Duty cycle is a 10 bit value (0 to 1023). The maximum duty cycle value must not be set greater than 4x the period, as the mark ‘on time’ would then be longer than the total PWM period (see equations above)! Setting above this value will cause erratic behavior.
3. The servo command cannot be used at the same time as the pwmout command as they both use the same timer.
4. pwmout stops during nap, sleep, or after an end command
5. pwmout is dependent on the clock frequency.

6. To stop pwmout on a pin it is necessary to isssue a 'pwmout pin, off' command. The pwmout command initializes the pin for pwm operation and starts the internal timers.

Example:

```
init:
        pwmout 2,150,150  ‘ set pwm period and duty cycle
        pause 1000        ‘ pause 1 s
        pwmout 2,150,50   ‘ set a different duty cycle
        pause 1000        ‘ pause 1 s
        goto init         ‘ loop back to start
```

12. PROGRAM 5

Once the PicAxe board has been completed and tested we move on to the next activity which requires interfacing the PicAxe to real world devices, specifically the LM34 Analog temperature sensor which has a linear 10mV per degree C output and the DS18B20 digital temperature sensor, both of which are extensively described in the Lab Manual.

We begin with the LM34 and this is an opportune time for a lecture on A/D conversion which explains how to use the PicAxe to read analog voltages (both 8 and 10 bit readings are available) and how to use the PicAxe Programming Editor to debug a program.

Students are given a variety of programming assignments for the LM34. The first assignment is to read the LM34 (using debug) then convert the reading to an actual temperature (again using debug) and finally to blink an LED for the tens digit, pause and then blink the LED for the units digit. For example, if the temperature was 73 degrees then we would blink a LED seven times, pause and then blink the LED three times.

Some other assignments could be to light one LED when the temperature is high, light a different LED when the temperature is low and light another LED when the temperature is at a comfortable level. Another assignment could play a sound to alert when a certain temperature has been exceeded.

Students are given the following program as a starting point:

```
' LM34-1
‘
‘ AXE092 Program to read the LM34 (8 bits)
'
' I/O Pin Aliases
symbol RED_LED = 0
symbol YELLOW_LED = 1
symbol GREEN_LED = 2

' new aliases
symbol LM34 = 4   ‘ Use I/O #4.  Make sure dip switch is off

' Variables
symbol tempIn      = w0      ' b0, b1

Main:
        readadc LM34, tempIn          ‘read the analog voltage
        debug
        pause 100                     ‘ wait for 100 msec
goto main
```

When you run the program a debug window will open and you can see the value of the analog input voltage by looking at variable w0 (also shown as tempIn). Using debug you can watch variable tempIn change as you touch the LM34.

We now need to apply a factor which will give an output in degrees Fahrenheit. Change the code as follows:

```
'
' LM34-1
‘
‘ AXE092 Program to read the LM34 (8 bits)
‘ and convert the reading to degrees Fahrenheit

' I/O Pin Aliases
symbol RED_LED = 0
symbol YELLOW_LED = 1
symbol GREEN_LED = 2
symbol PIEZO = 2
symbol PUSHBUTTON = pin3
symbol LDR = 4

' new aliases
symbol LM34 = 4   ‘ Use I/O #4.  Make sure dip switch is off

' Variables
symbol tempIn      = w0      ' b0, b1
symbol tempF       = w1      ' b2, b3

Main:
        readadc LM34, tempIn          ‘read the analog voltage

' Since our power supply voltage is 4.5 volts the voltage/step
' for an 8 bit A/D converter is (4.5/256) = 0.0175 volts (17.5 mV)
' which is equivalent to 1.75 degrees F.
‘
' The step sizes or quantizing level limits the temperature
' resolution to 1.75 degree F. so we need to multiply the input
‘ by 1.75.  Since we only have integer arithmetic we multiply by
‘ 175 and then divide by 100 to get the correct value
'
        tempF = tempIn *  175 / 100
        debug
        pause 100           ‘ wait for 100 msec
goto main
```

Students download the above program and verify that variable tempF is correct. Hopefully they understand the comments.

Now that we can read a temperature from the LM34 we need to be able to display it. We will do this by pulsing one of the LED's.

In order to make things easier we will impose some limits on the temperature. Remember that the single power supply for the LM34 already limits us to 0 degrees and we also will fix the highest temperatures to 99 degrees. This gives us a two digit range for tempF that is between 0 and 99 degrees.

Once we have the temperature we will split the tens and units and pulse a LED the appropriate number of times for each digit with a pause between them.

For example, if tempF = 68 degrees we want to set the LED to flash 6 times, pause, and then flash 8 times.

Here is a skeleton program for you to start from. Make the necessary changes and save it as LM34-2.bas

```
'
‘ LM34-2
‘
‘ AXE092 Program to read the LM34 (8 bits)
‘ and convert the reading to degrees Fahrenheit.
‘ Display the value by blinking the red LED

' I/O Pin Aliases
symbol RED_LED = 0
symbol YELLOW_LED = 1
symbol GREEN_LED = 2
symbol PIEZO = 2
symbol PUSHBUTTON = pin3
symbol LDR = 4

' Global Constants
symbol BLINK_DELAY = 200

' new aliases
symbol LM34 = 4   ‘ Use I/O #4.  Make sure dip switch is off

' Variables
symbol tempIn       = w0       ' b0, b1
symbol tempF        = w1       ' b2, b3
symbol singleDigit  = b4
symbol cnt          = b5

Main:
            readadc LM34, tempIn          ‘read the analog voltage

' Since our power supply voltage is 4.5 volts the voltage/step
' for an 8 bit A/D converter is (4.5/256) = 0.0175 volts
' which is equivalent to 1.75 degrees F.
‘
' The step sizes or quantizing level limits the temperature
' resolution to 1.75 degree F. so we need to multiply the input
‘ by 1.75.  Since we only have integer arithmetic we multiply by
‘ 175 and then divide by 100 to get the correct value
'
            tempF = tempIn *  175 / 100

            singleDigit = ?????             ' get the tens digit
            GoSub BlinkLed

            singleDigit = ?????             ' get the units digit
            GoSub BlinkLed

            debug
            pause 1000                      ' wait for 1000
goto main


BlinkLed:
            for cnt = 1 to singleDigit
                        (add the code here)
            next
Return
```

Note that the lines “singleDigit = ?????” will give an error. I expect that the students will attempt to insert the correct code at each statement.

Also the line “(add the code here)” will additionally cause an error. Students are required to add the code to blink an LED (which they have previously been given)

Additional assignments are given such as:

- Change the above code to play a tone instead of blinking a LED
- Change the previous code to read a ten digit value using readadc10. Remember you will need to figure out the step size, etc. Display your results by either blinking a LED or playing tones
- Due to a variety of reasons (air flow, someone touching the sensor, etc.) a single reading from the LM34 (or any other analog sensor), even though the reading itself is correct, may give inaccurate results in a given application. We can use a technique known as averaging which takes multiple readings over a given time period and then averages all the readings to get a single result. We would use a “for..next” loop to read the sensor and sum the results. At the end of the loop we divide by the number or readings to get the average. Take one of the above programs and add averaging to it.
- Convert the temperature from degrees F to degrees C. Note that 0 degrees Fahrenheit converts to minus 17.8 degrees C (-17.8) so you will may need to make some additional assumptions in your program. Hint: you could use one LED for positive temperatures and a different LED for negative temperatures.
- The conversion formula is: C = 5/9(F-32)

13. PROGRAM 6

Now that we have a working LM34 thermometer we would like to have some way of displaying the actual temperature rather than blink LEDs.

I have designed a cascadable single digit display module which interfaces directly to the PicAxe and the students build, test and incorporate two of these modules into their microcontroller thermometer. They a given a lecture explaining how the modules work including the operation of shift registers, BCD to seven segment decoders and seven segment common anode LEDs. They use an oscilloscope to display signals and troubleshoot.

Again, sample programs are provided but the students must learn what to program usually by trial and error.

14. PROGRAM 7

At this time we introduce the DS18B20 digital thermometer IC. It provides eight or twelve bit accuracy and interfaces to the PicAxe using a single built-in command. Students are required to integrate the DS18B20 into their thermometer and write the appropriate program.

They then write a program to display the readings from both sensors sequentially and they learn why the reading may differ.

15. PROGRAM 8

We go over some advanced features of the PicAxe including pulse width modulation (PWM), understanding and using interrupts, and multitasking. At this time a term project is assigned.

16. PROGRAMMING GUIDELINES

The following paragraphs are intended to provide software developers with some programming (coding) guidelines to be used in this class and hopefully to instill a structured approach to writing computer programs regardless of the language used. We will limit this discussion to the BASIC language as implemented in the Picaxe series of microcontrollers but many of guidelines are also applicable to other languages.

Standardization is especially important in a large development organization where any individual programmer may need to look at another programmer's code. It must be clear what the code does, how it should be used, how it can be extended, etc. Hopefully these guidelines will provide a framework upon which we can all create code which is easily readable and maintainable by any developer working on the project.

16.1 Naming Conventions

Naming conventions make programs more understandable by making them easier to read. They can also give implicit information about a subroutine or variable – for example, whether an item is a constant – which can be helpful in understanding the code.

The following naming conventions are suggested but by no means intended to be the only possibility. Naming conventions are very individual and every programmer develops their own.

Item	Convention
Labels	Should be in mixed case with the first letter of each word capitalized. Try to keep the names simple and descriptive. Examples: Main: TurnOnGreenLed: ReadTemperatureSensor:

Variables	In Picaxe basic all variables are global (i.e. they are available throughout the program) and should be in mixed case beginning with a lowercase letter with each subsequent new word in uppercase, and subsequent letters in each word in lower case. Examples:
	symbol today = b7;
	symbol loopCounter = b8;

Constants	Constants should be all upper case and use the underscore character for readability. Examples: symbol DAYS_IN_WEEK = 7;
Grammar	Avoid code that embeds many operations in a single line. This kind of code is error prone, difficult to decipher, and hard to debug

17. CONCLUSIONS

Using this project based learning approach I find that students are both interested and excited in completing their projects, especially since the completed projects are theirs to keep. They especially liked the fact that they have something tangible to show their parents, siblings and friends.

Many students have remarked favorably about the class and expressed their desire to either enhance their project or to develop a new project. Some students who have continued on to four year institutions have returned and thanked me for the skills they learned.