# Integrating Hardware and Software In Project Based 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
In this paper, we discuss a novel approach to project based learning incorporating hardware skills (soldering, de-soldering, prototyping, using a drill press), application software skills (Multisim[1] circuit simulation software, Ultiboard[2] PC board layout software) and programming skills in the BASIC programming language.

## 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.

## Keywords
Microcontroller, PicAxe, Display.

## 1. INTRODUCTION
Project based learning incorporates a hands-on approach to understanding fundamental concepts and relationships. The student physically builds one or more projects and the building (and troubleshooting) process enhances his skills in the subject. By incorporating both simulation software and a programmable device into this lab the student enhances his physical skills, his critical thinking skills, and his programming skills – all of which are in high demand by employers.

## 2. PROGRAMMABLE DEVICES
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 the lab in order to better prepare them for the demanding requirements of the workplace.

## 3. LAB ACTIVITY 1
In this lab we expect the students to complete many projects in order to enhance their knowledge and skills. We first introduce circuit simulation software (Multisim) and have the students create a simple circuit schematic. Using the Multisim software they connect the various components and produce a virtual oscilloscope trace of the circuit output.

We then move on to a more complex circuit, an 5V to 12V adjustable power supply, which uses many more components and acts as an introduction to computer aided design principals. After an initial lecture on power supplies, the students create the power supply (in sections) and use Multisim to capture oscilloscope traces at various points in the circuit. They learn about various components and real world component tolerances and are able to visually see how the components interact.

We then take the completed circuit design and use the Ultiboard software to actually prepare a Printed Circuit board including placement of components and trace routing. Once the layout is complete the students get to see a PC board being manufactured on our Milling machine.

In the last part of this activity each student is given the power supply components and the PC board and is required to solder all the components onto the PC board and mount the PC board into an enclosure. When complete, the students have a working adjustable power supply which is theirs to keep and can be used in a variety of applications. During the assembly, they must take measurements and scope traces and compare the actual results to those predicted by Multisim and de-solder and re-solder components as necessary.

At the completion of this activity the student should have familiarity with both hardware skills and application software skills.

## 4. LAB ACTIVITY 2
Once the previous project has been completed we move on to the next activity - building and programming a microcontroller.

Revolution Education, the manufacturer of the PicAxe provide 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

[1] Multisim is a product of National Instruments Corp.

[2] Ultiboard is a product of National Instruments Corp.

which is geared for younger students and the powerful PicAxe Programming Editor[3].

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"[4] with 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.

## 5. LAB ACTIVITY 3

Once the PicAxe board has been completed and tested we move on to the next activity which requires interfacing the PicAxe to real word devices, specifically the LM34 Analog temperature sensor 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 temperature and 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.

## 6. LAB ACTIVITY 4

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.

## 7. LAB ACTIVITY 5

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.

## 8. LAB ACTIVITY 6

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.

## 9. LAB DOCUMENTATION

The remainder of this paper consists of sections from the "ET410 Laboratory Manual" and is included to illustrate the hardware and software expectations we have of our students.

## 10. INTEGRATED CIRCUITS[5]

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

[3] Revolution Education

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

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

signals. One large category of ICs is *microprocessors* or *microcomputers.*

## 11. MICROPROCESSORS[1]

A microprocessor IC like those manufactured by *Intel* or *Advanced Micro Devices* (AMD) is very powerful and very fast, but needs a bunch of other components to work with it. In addition to the microprocessor, your home computer has different types of memory. ROM or Read-Only-Memory is one type; RAM or Random Access memory is another. We also have flash memories, EEPROMS, and others.

Your home computer also has some input and output equipment, which we call *I/O* devices, such as keyboards, mice, displays, and printers. In most cases, the interface circuits that let the microprocessor talk to the I/O devices are also separate from the microprocessor chip itself.

The microprocessors used in home computers are designed to be general purpose - that is, they can do lots of different tasks at different times, and usually very fast. They may be used to write a letter today, and calculate checks the next day —that's what we mean by *general purpose.*

But there are other computers which only do specific tasks and may operate very slowly as compared to general purpose microprocessors. For instance, your cell phone has a small computer inside which has very limited ability. So does the microwave oven in your home. So does a CD player or a digital watch.

## 12. MICROCONTROLLERS[1]

Because this type of processor doesn't need much memory or I/O it often has these built into the same IC. The result is then called a *microcomputer or microcontroller* since it contains on a single chip most of the parts that larger computers have on separate ICs. Some people also call it a computer-on-a-chip.

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 designed 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*[6]. 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.

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

## 13. THE PICAXE[7]

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.

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

Unlike other BASIC 'module' based systems, all Picaxe programming is at the 'chip' level. Therefore instead of buying an expensive pre-assembled (and difficult to repair) surface mount module, with the Picaxe system you simply purchase a standard chip and use it directly in your project board.

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 from just 3 components and can be easily constructed on a prototyping breadboard or printed circuit board.

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.

## 14. PICAXE SOFTWARE

The manufacturer of the Picaxe provides a free Programming Editor which I suggest you download and install on your home computer. The Picaxe 'Programming Editor' software is free so the only cost per computer is the low-cost download cable which can be built by the students as part of the class.

The free Picaxe Programming Editor software can be downloaded from http://www.picaxe.co.uk. Connect the cable to the serial port at the back of your computer (if you have a modern laptop without a serial port you will need to purchase a USB-Serial adapter, part USB010 or the USB Download Cable AXE027).

After installing and running the Programming Editor software, click the View>Options menu to put the software into 'Picaxe-08M' mode, as this is the type of chip used on this board. Also make sure the serial port number (COM1, COM2 etc.) corresponds to where the cable was connected at the rear of the computer and then click OK.

## 15. PICAXE INPUTS AND OUTPUTS

The PicAxe provides the following type of inputs and outputs:

### 15.1 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

[7] Portions reprinted from the Revolution Education PicAxe Manual

devices can be interfaced via a transistor, FET or Darlington driver array.

## 15.2 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.

## 15.3 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.

# 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.

| 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:<br>Main:<br>TurnOnGreenLed:<br>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:<br>symbol today = b7;<br>symbol loopCounter = b8; |
| Constants | Constants should be all upper case and use the underscore character for readability. Examples:<br>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. BASIC PROGRAMMING LANGUAGE

## 17.1 History[8]

**BASIC** is a family of general-purpose, high-level programming languages whose design philosophy emphasizes ease of use - the name is an acronym from **Beginner's All-purpose Symbolic Instruction Code**.

The original Dartmouth BASIC was designed in 1964 by John George Kemeny and Thomas Eugene Kurtz at Dartmouth College in New Hampshire, USA to provide computer access to non-science students.

At the time, nearly all use of computers required writing custom software in assembly language, which was something only scientists and mathematicians tended to do. The BASIC language and its variants became widespread on microcomputers in the late 1970s and 1980s when it was typically a standard feature and often part of the firmware of the machine.

BASIC remains popular in numerous dialects and new languages (which were influenced by the original BASIC) such as Microsoft Visual Basic. In 2006, 59% of developers for the .NET Framework used Visual Basic .NET as their only programming language.

## 17.2 PicAxe BASIC

PicAxe BASIC is implemented in the PicAxe firmware and is interpreted rather than compiled. There are many different PicAxe BASIC statements, most of which fall into the following categories:

- Comments
- Labels
- Symbols
- Assignment
- Mathematical
- Program Control
- Input/Output (I/O)

### *17.2.1 Comments*

Comments are perhaps the most important BASIC statements because they allow programmers to annotate their code and explain (to both themselves and others) what their code is supposed to do.

Comments begin with an apostrophe (') or semicolon (;) and continue until the end of the line. Comments are not executed by the computer and they do not consume any run-time resources. Comments should be used in every program.

[8] From Wikipedia

### *17.2.2 Labels*

Labels are used as markers throughout the program. Typically, labels are used to identify a specific line of code which will be used as a starting point for a branch instruction or subroutine and always end with a colon (:). Labels must start with a letter or underscore (not a digit) and can be any word (that is not already a reserved keyword) and may contain digits and the underscore character. For example:

Main:

Start0:

My_Subroutine:

_TestFunction:

Note that the **PicAxe is not case sensitive** so that lower and/or upper case may be used at any time. The values "mylabel:", "Mylabel: ", "MyLabel:" and "MYLABEL:" are identical.

### *17.2.3 Symbols*

Symbols (or aliases) allow programmers to assign their own names to variables and I/O pins rather than use the variable names or pin numbers themselves. Using symbols makes program code easier to read and understand.

### *17.2.4 Variables*

Variables are implemented using the microcontrollers RAM memory, therefore once power is removed from the chip (or the chip is reset) all RAM memory is lost.

The PicAxe 08m has fourteen general purpose byte (8 bit) variables, b0-b15, seven general purpose word (16 bit) variables, w0-w6 and 16 general purpose bit variables, bit0-bit15.

A bit variable can only be 0 or 1, a byte variable can have any value between 0 and 255 while a word variable can have any value between 0 and 65,535.

**Memory Layout**

Word variables are composed of 2 byte variables as follows:

w0 = b1:b0

w1 = b3:b2

w2 = b5:b4

w3 = b7:b6

w4 = b9:b8

w5 = b11:b10

w6 = b13:b12

w7 = b15:b14

The bit variables are part of b0 and b1 as follows:

b0 = bit7:bit6:bit5:bit4:bit3:bit2:bit1:bit0

b1 = bit15:bit14:bit13:bit12:bit11:bit10:bit9:bit8

These variables share a common memory area so that setting w0 to a value also sets b0 and b1 and also bit0-bit15. This can be a source of confusion to programmers. You can use any word, byte or bit variable within any mathematical assignment or command that supports variables. However take care that you do not accidentally repeatedly use the same 'byte' or 'bit' variable that is being used as part of a 'word' variable elsewhere.

### *17.2.5 Assignment Statements*

Assignment Statements are used to initialize variables and perform mathematical operations on the contents of those variables. For example:

Let b0 = 1

b0 = 1

b1 = b0

b1 = b0 + 1

### *17.2.6 Mathematics*

The PICAXE microcontrollers support both byte (8 bit) and word (16 bit) mathematics however internally all math is 16 bits. In addition**, the PicAxe does not support negative numbers or fractions.**

Mathematical operations will 'overflow' without warning if you exceed the byte or word boundary values. If the output target is a byte variable and **if the result of the internal calculation is greater than 255 then overflow will occur without warning.**

For example, using byte variables:

b0 = 254 + 3 = 1

b0 = 2 - 3 = 255.

***Math is performed strictly from left to right. Unlike some computers and calculators, the PICAXE does not give * and / priority over + and -. On our PicAxe, you cannot use parenthesis to force a specific sequence of mathematical operations.***

For example, 3+4x5 is calculated as

3+4=7

7x5=35

Given that the PicAxe does not support fractions, sometimes it is possible to rewrite equations to use integers instead of fractions.

For example:

let w1 = w2 / 5.7 **'**will give an unexpected result that is not valid, but

let w1 = w2 * 10 / 57 **'**is mathematically equivalent and work correctly.

Built-in mathematical operations include the following:

- +, -, *, / add, subtract, multiply, divide
- &, |, ^, ^/bitwise AND, bitwise OR, bitwise XOR, bitwise XNOR
- //, % modulus divide (returns remainder)

### *17.2.7 Program Control*

Program Control statements are used to change the flow of a program. In programming, statements are executed one after the other unless one of the statement changes the flow. For example:

Goto [label] ‘ this forces the program to continue execution at a predefined point

GoSub [label] ‘ this calls a subroutine and once the subroutine finishes, program flow will ‘return to the statement following the gosub

Return ‘Note that the subroutine must end with the “return” statement

If [condition tested is true] then [(goto) label]

If [condition tested is true] then

‘Execute the statements here

Else

‘Execute the statements here

Endif

**For**

For [mathematical expression]

‘Execute the statements here

‘Execute more statements here

Next

### *17.2.8 Output Commands*

The following commands affect the designated output pin:

High 0 ‘ set output pin 0 to high (i.e. Vcc)

low 0 ‘ set output pin 0 to low (i.e. Gnd)

toggle 0 ‘ changes output pin 0 to low if it was high and to high if it was low

pulsout 1, 15 ‘Outputs a timed pulse by inverting an output pin for a given period of time. This example generates a pulse on pin 1 of 15 units where each unit is 10us. The pulse width would therefore be 150 us. If the output is initially low, the pulse will be high, and vice versa.

Pwmout pin,period,duty_cycle

Generate a continuous pwm output using the microcontroller’s internal pwm module. See the manual for more information

SOUND pin,(note,duration,note,duration...)

This command is designed to make audible ‘beeps’ for games and keypads etc.

To play music use the play or tune command instead. Note and duration must be

used in ‘pairs’ within the command.

TUNE pin, speed, (note, note, note...) *(PICAXE-08M only)*

The tune command allows musical ‘tunes’ to be played. Playing music on a microcontroller with limited memory will never have the quality of commercial playback devices, but the tune command performs remarkably well.

Music can be played on economical piezo sounders or speakers

### *17.2.9 Input Commands*

The following commands derive an input value from the designated input pin

**READADC pin, variable**

- pin is a variable/constant specifying the ADC pin

- Variable receives the data byte read.

The readadc command is used to read an 8 bit analog value from the designated microcontroller input pin into a variable. Note that not all inputs have internal ADC functionality - check the pinout diagrams for the PICAXE chip you are using.

**READADC10 pin, wordvariable**

- pin is a variable/constant specifying the input pin (0-7)

- wordvariable receives the data word read.

The readadc10 command is used to read a 10 bit analog value from the designated

Microcontroller input pin. Note that not all inputs have internal ADC functionality - check the table under ‘readadc’ command for the PICAXE chip you are using. Since the result is 10 bits a word variable must be used.

**PULSIN pin, state, wordvariable**

Pin is a variable/constant (0-7) which specifies the i/o pin to use.

State is a variable/constant (0 or 1) which specifies which edge must occur before beginning the measurement in 10us units (4MHz resonator).

Wordvariable receives the result (1-65535). If timeout occurs (0.65536s) the result will be 0.

The pulsin command measures the length of a pulse. In no pulse occurs in the timeout period, the result will be 0. If state = 1 then a low to high transition starts the timing, if state = 0 a high to low transition starts the timing.

Use the count command to count the number of pulses within a specified time period.

It is normal to use a word variable with this command.

**BUTTON pin, downstate, delay, rate, bytevariable, targetstate, address**

Pin is a variable/constant (0-7) which specifies the i/o pin to use.

Downstate is a variable/constant (0 or 1) which specifies what logical state is read when the button is pressed.

Delay is a variable/constant (0-254) which is a counter which specifies the number of loops before a repeat if BUTTON is used within a loop. A value of 255 disables this feature.

Rate is a variable/constant (0-255) which specifies the auto-repeat rate in BUTTON cycles.

Bytevariable is the workspace. It must be cleared to 0 before being used by BUTTON for the first time (before the loop that BUTTON is in)

Targetstate is a variable/constant (0 or 1) which specifies what state (0=not pressed, 1=pressed) the button should be in for a branch to occur.

Address is a label which specifies where to go if the button is in the target state.

When mechanical switches are activated the metal ‘contacts’ do not actually close in one smooth action, but ‘bounce’ against each other a number of times before settling. This can cause microcontrollers to register multiple ‘hits’ with a single physical action, as the microcontroller can register each bounce as a new hit. One simple way of overcoming this is to simply put a small pause (e.g. pause 10) within the program, this gives time for the switch to settle.

Alternately the button command can be used to overcome these issues. When the button command is executed, the microcontroller looks to see if the ‘downstate’ is matched. If this is true the switch is debounced, and then program flow jumps to ‘address’ if ‘targetstate’ = 1. If targetstate = ‘0’ the program continues. If the button command is within a loop, the next time the command is executed ‘downstate’ is once again checked. If the condition is still true, the variable ‘bytevariable’ is incremented. This can happen a number of times until ‘bytevariable’ value is equal to ‘delay’. At this point a jump to ‘address’ is made if ‘targetstate’ = 1. Bytevariable is then reset to 0 and the whole process then repeats, but this time the jump to ‘address’ is made when the ‘bytevariable’ value is equal to ‘rate’. This gives action like a computer keyboard key press - send one press, wait for ‘delay’ number of loops, then send multiple presses at time interval ‘rate’. Note that button should be used within a loop. It does not pause program flow

and so only checks the input switch condition as program flow passes through the command.

### *17.2.10 Miscellaneous Commands*

The following miscellaneous commands are described in the PicAxe manual, please read the manual for a full description of these and many other commands:

**pause, wait, sleep, nap, end**: These commands force the PicAxe to delay execution by a specified period of time.

**Debug**: Used with the PicAxe Programming editor to troubleshoot a program.

‘ Provide a detailed description which may include

‘ intent of subroutine

‘ pre and post conditions

‘ side effects

‘ dependencies

‘ implementation notes

For example:

‘ This function turns on the red LED for a fixed time and then turns it off

‘

‘ It depends on variable redLedOnTime which must be set

‘ before this subroutine is called.

‘

‘ Upon completion, the red LED will be off

TurnOnRedLed:

High RED_LED

Pause redLedOnTime

Low RED_LED

Return

### 17.3 Normal or Body Text

Please use a 9-point Times Roman font, or other Roman font with serifs, as close as possible in appearance to Times Roman in which these guidelines have been set. The goal is to have a 9-point text, as you see here. Please use sans-serif or non-proportional fonts only for special purposes, such as distinguishing source code text. If Times Roman is not available, try the font named Computer Modern Roman. On a Macintosh, use the font named Times. Right margins should be justified, not ragged.

### 17.4 First Page Copyright Notice

Please leave 3.81 cm (1.5") of blank text box at the bottom of the left column of the first page for the copyright notice.

### 17.5 Subsequent Pages

For pages other than the first page, start at the top of the page, and continue in double-column format. The two columns on the last page should be as close to equal length as possible.

## 18. FIGURES/CAPTIONS

Place Tables/Figures/Images in text as close to the reference as possible (see Figure 1). It may extend across both columns to a maximum width of 17.78 cm (7”).

Captions should be Times New Roman 9-point bold. They should be numbered (e.g., “Table 1” or “Figure 2”), please note that the word for Table and Figure are spelled out. Figure’s captions should be centered beneath the image or picture, and Table captions should be centered above the table body.

## 19. SECTIONS

The heading of a section should be in Times New Roman 12-point bold in all-capitals flush left with an additional 6-points of white space above the section head. Sections and subsequent sub-sections should be numbered and flush left. For a section head and a subsection head together (such as Section 3 and subsection 3.1), use no additional space above the subsection head.

### 19.1 Subsections

The heading of subsections should be in Times New Roman 12-point bold with only the initial letters capitalized. (Note: For subsections and subsubsections, a word like *the* or *a* is not capitalized unless it is the first word of the header.)

#### *19.1.1 Subsubsections*

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized and 6-points of white space above the subsubsection head.

#### *19.1.1.1 Subsubsections*

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized.

#### *19.1.1.2 Subsubsections*

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized.

[1]

# Columns on Last Page Should Be Made As Close As Possible to Equal Length