Introduction to the Atmel ATmega16 Microcontroller

Learning Objectives:

At the end of this lab you should be able to:

- Identify the Atmel ATmega16 microcontroller, STK500 Development Board, and associated hardware.
- Create a new project in AVR Studio, and populate the project with pre-existing code.
- Use AVR Studio to compile code in ANSI C.
- Use AVR Studio to program the ATmega16 microcontroller.

Components:

Qty.	Item
1	Atmel ATmega16 microcontroller mounted to an STK500 development board
1	Serial programming cable
1	12 VDC power supply
1	6-pin ribbon cable
1	2-wire female-female jumper
2	10-wire female-female jumper

Introduction

A microcontroller often serves as the "brain" of a mechatronic system. Like a mini, self-contained computer, it can be programmed to interact with both the hardware of the system and the user. Even the most basic microcontroller can perform simple math operations, control digital outputs, and monitor digital inputs. As the computer industry has evolved, so has the technology associated with microcontrollers. Newer microcontrollers are much faster, have more memory, and have a host of input and output features that dwarf the ability of earlier models. Most modern controllers have analog-to-digital converters, high-speed timers and counters, interrupt capabilities, outputs that can be pulse-width modulated, serial communication ports, etc.

The microcontroller and the development board used in this lab were donated by Atmel for your use. In industry, you can expect to pay anywhere from $50 to $400 for just a development board and up to $1000 for a professional compiler and programming interface! **SO BE CAREFUL AND RESPECTFUL** of the microcontrollers and development boards! Like any electronic device, they are ***delicate*** and may be ***easily damaged***! **BE ESPECIALLY CAREFUL** of static charges! Before you touch the STK500 board (or any other circuit board with integrated circuits, for that matter), make sure that you have dissipated any static charge that has accumulated on your body. The best way to do this is by using an ESD wrist strap that has been connected to a good earth ground and by placing your circuit board on a grounded ESD mat. If you don't have these ESD supplies, touch a well-grounded metal surface before you handle the circuit board.

The bottom line is: **USE COMMON SENSE,** and **FOLLOW THE INSTRUCTIONS** in the lab assignments. You will ***build upon your experience*** from each lab, and you are therefore ***encouraged*** to learn as much as you can from each experiment and its examples.

The ATmega16 Microcontroller

The ATmega16 microcontroller used in this lab is a 40-pin wide DIP (**D**ual **I**n **L**ine) package chip. This chip was selected because it is robust, and the DIP package interfaces with prototyping

supplies like solderless bread boards and solder-type perf-boards. This same microcontroller is available in a surface mount package, about the size of a dime. Surface mount devices are more useful for circuit boards built for mass production. Figure 1 below shows the 'pin-out' diagram of the ATmega16. This diagram is very useful, because it tells you where power and ground should be connected, which pins tie to which functional hardware, etc.

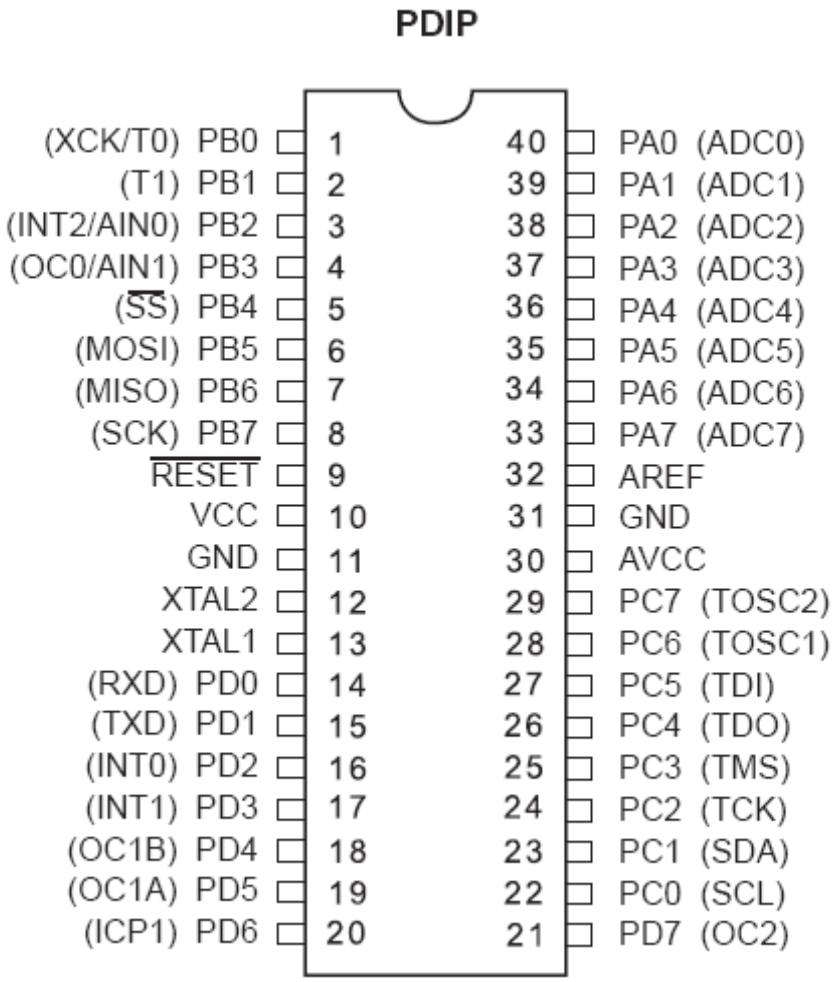

Figure 1. ATmega16 Pin-out diagram. Notice that some of the pins have alternate functions (shown in parentheses).

Throughout the semester, you will need to know things about the ATmega16 (or other components) that are not covered in the lab instructions. Therefore it is important that you become familiar with documentation available from various sources. **Your first task is to locate the ATmega16 manual, and save it for yourself for future reference**. It can be found in a pdf format on Atmel's website (www.atmel.com), AVRFreaks (http://www.avrfreaks.net/), or by searching the web. From the manual, you can find information about the ATmega16's features and how to use them. **How many channels of 10-bit A/D converters are there? How many bytes of in-system reprogrammable flash are there?**

STK 500 Interface Board

The hardware that you will use consists of the STK500 development board. The SKT500 can actually be used with any of the microcontrollers in the AVR family (see the STK500 user guide). It allows a user to work with many different Atmel microcontrollers and easily gain access to their I/O pins. The STK500 has two serial port connectors (one for programming the devices and one as a

spare RS232 port), a power supply switch and connector, eight LEDs and eight switches for general use, and various jumpers for configuring the board. Figure 2 shows a top view of the STK500 interface board and the location of some of the hardware elements that you will use.

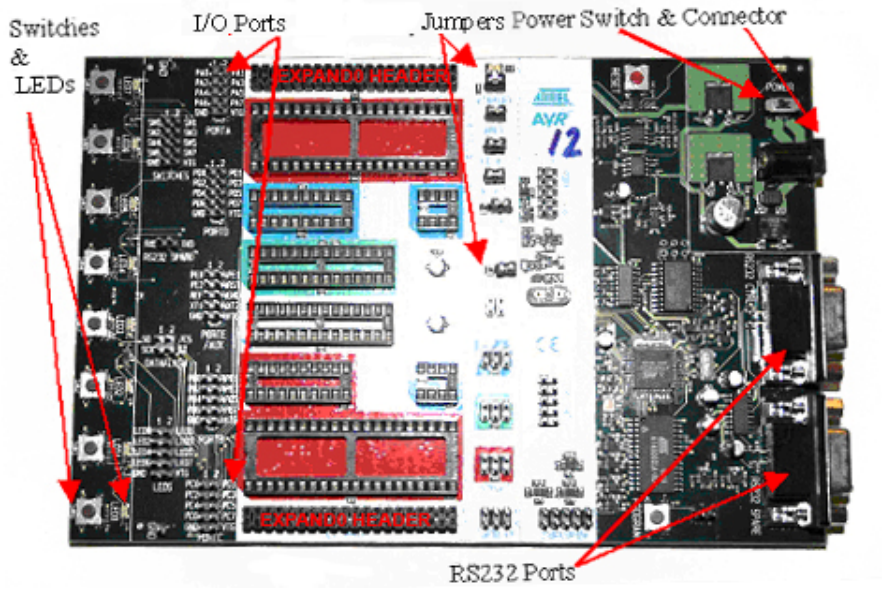

Figure 2. STK500 development board. This is a 'universal' development board for AVR microcontrollers. Note the location of the features as indicated by the arrows.

Before you go further in the procedure, double-check the default jumper settings on the STK500 as shown in Figure 3.

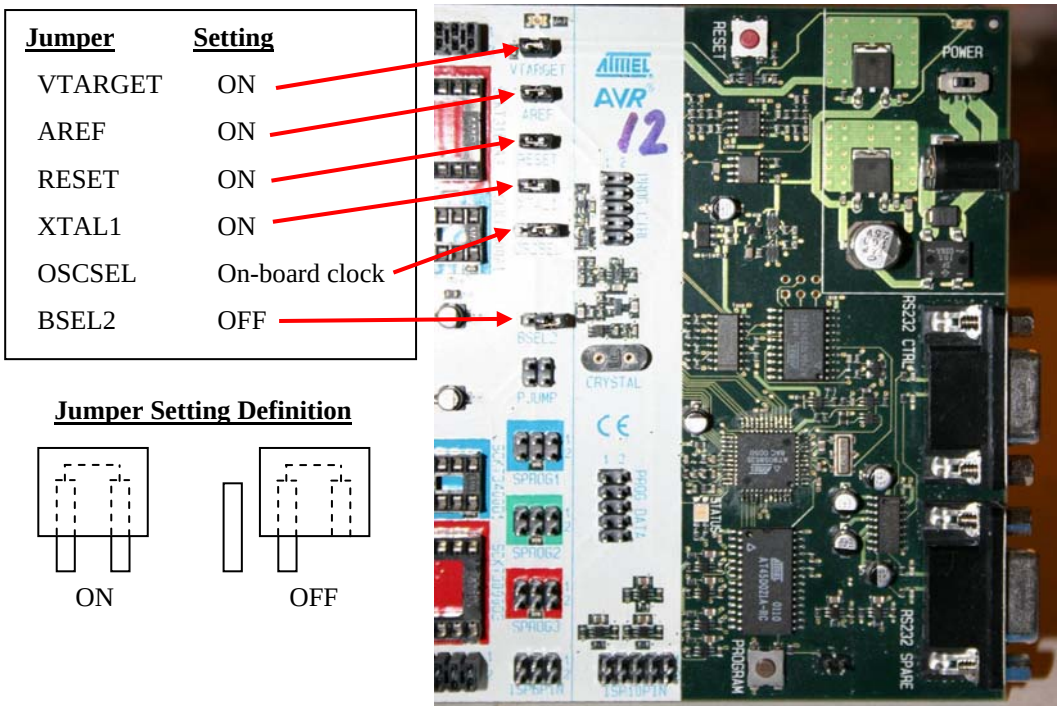

Figure 3. Default jumper settings on the STK500. A jumper consists of two female sockets tied together internally with a metal conductor and surrounded by a plastic housing. Typically, when the pins are connected, the jumper is considered to be in the ON position. When setting a jumper in the OFF position, place the jumper on only one of the two pins. This way the jumper does not get lost and is available in case you need to change its setting. The jumper for OSCSEL is set so that it connects the two right-most pins of the three. Thus set, it selects the on-board clock signal.

You will be using the ISP programming mode for communicating with and downloading your programs to the microcontroller. As a result, you need to connect the 6-pin ISP ribbon cable from the ISP6PIN header on the STK500 to the 6-pin SPROG3 header outlined in red above it. Figure 4 shows this connection on the STK500. **BE SURE** to line up the red, pin1 stripe on the ribbon cable to pin 1 on each header. After completing this connection, the STK500 interface board should be ready for use in your labs.

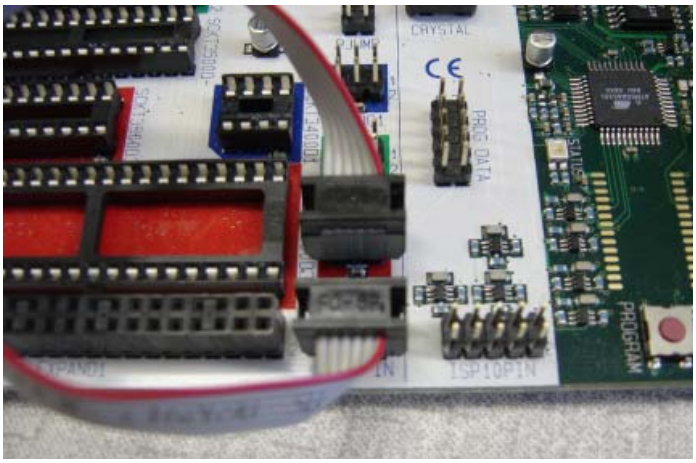

Figure 4. 6-Pin ISP ribbon cable connection for programming the ATmega16. Connect the ISP6PIN header set on the STK500 to the SPROG3 header set outlined in red using the 6-pin ISP ribbon cable. Make sure that you orient the red stripe on the cable, so that it connects to pin 1 on both header sets.

WinAVR

There are several ways that you can write, compile, and download a program to the ATmega16 microcontroller. There are many different text editors, compilers, and utilities available for many different languages (C, BASIC, assembly language, etc.). Some of these are free of charge, and some require a licensing fee to use them. In this class, we will use a freeware package of software tools named WinAVR (pronounced, “whenever”). WinAVR has been installed on the computers in the Mechatronics Laboratory, but you are strongly encouraged to download it and install it on your own computer, so you can work with your microcontroller outside of the lab. Instructions for installing WinAVR are given in Appendix A at the end of this document.
WinAVR consists of a suite of executable, open source software development tools for the Atmel AVR series of RISC microprocessors hosted on the Windows platform. It includes the GNU GCC compiler for C and C++, which is sometimes referred to as avr-gcc.

Traditionally, the microcontroller in embedded systems was programmed directly using assembly language. Assembly language uses only the basic instruction set for a particular microcontroller. While this can produce fast, efficient code, it is limited in that every processor type has its own instruction set. Therefore it is not a practical language to learn unless you are doing a project that is dedicated to a specific microcontroller or has a real need for precise timing and/or memory use. The C language, on the other hand, is commonly used in industry and can be applied over many different platforms. By learning this one language, you will be able to program almost any microcontroller, provided that you have a compiler that can translate C code into assembly language for your controller. The Gnu-C compiler is an open-source, freeware, C compiler that forms the basis for compilers that generate code for many different microcontrollers and various operating systems, such as Windows and UNIX.

AVRFREAKS Are Everywhere!

AvrFreaks.com (http://www.avrfreaks.net/) is a website devoted entirely to the AVR processors from Atmel. It contains links to download locations for the most recent version of software, current information regarding new and old processors from Atmel (including user manuals, technical white papers, etc.), message boards for people just like you who have questions, and general information regarding compilers, programming interfaces, etc. This website can be a useful tool when developing or troubleshooting projects with the ATmega16. Much of the site can be accessed by anyone; however, to access some of the message boards and downloadable items, you must register a username. Don't worry, it's free! **Go to AvrFreaks.net now and register a username for yourself. Write down your AVRFreaks username and turn it in with the rest of your lab write-up.** [Before you post a question on any of the AVRfreaks forums (or any other web forum for that matter), you are strongly urged to read Eric Raymond's famous web post: "How to Ask Questions the Smart Way", which is available at: http://catb.org/esr/faqs/smart-questions.html.]

Procedure

Now, you'll begin building your first project. For this first project, you will use an example program provided for you that demonstrates how to print messages to the serial port. Try to remember these steps, because you will need to go through them in subsequent labs. The more familiar you are with these steps, the faster the labs will go in the future.

You will be using a freeware program from Atmel called AVR Studio to organize all the files needed to compile a program to run on the microcontroller. The next few steps will lead you through the process of setting up a file structure, which will contain the needed files. [Note that the computers in the Mechatronics lab already have AVR Studio and WinAVR installed on them. If you want to work on a microcontroller outside the lab, you will need to install this software on your own computer. See Appendix A at the end of this document for instructions.]

1. Open **AVR Studio** by double-clicking its icon on your desktop or selecting it in the Start Menu.
2. Select **Project -> New Project** from the drop down menus or New Project from the pop-up window if one opens when you launch AVR Studio. You should see the new project window as shown in Figure 5.

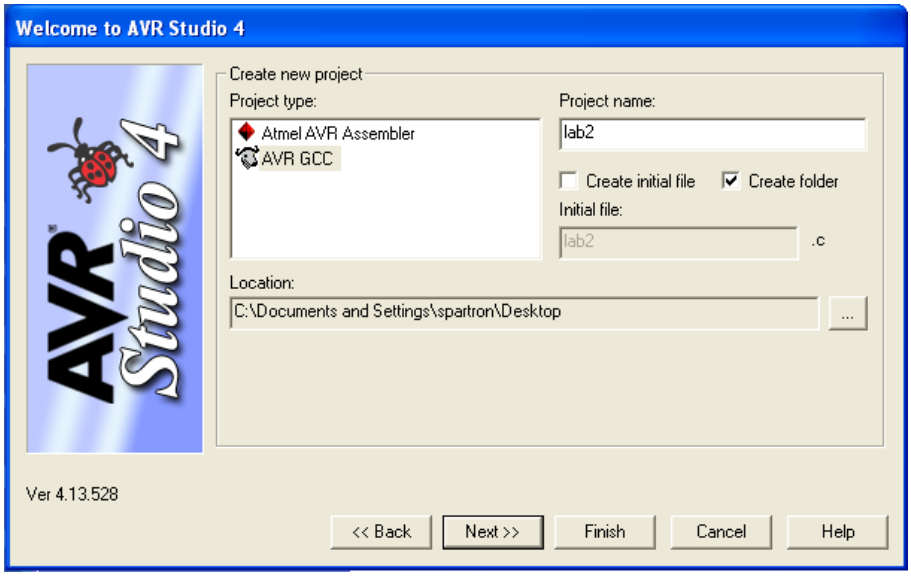

Figure 5. AVR Studio New Project window. Note the settings used for this lab and the location where your folder will be created.

3. In the new project pop-up window:
   - Select **AVR GCC** and type a descriptive name into the **Project name** field (like FirstAVRproject).
   - Since you will be using provided source code *un*check the **Create initial file** box.
   - Check the **Create folder** box.
   - Browse to ensure that the Location (where your project will be saved) already exists, so that you are aware of where you are saving files. Be sure to eventually copy your project files onto a USB drive or email your source files to yourself, because any files left on the lab computers may be modified or deleted at any time.
   - Click the **Next** >> button
   - Select **AVR Simulator** in the **Debug platform** field.
   - Select **ATmega16 (not ATmega16A)** in the **Device** field. (scroll down, it *is* there)
   - Click the **Finish** button
4. Configure your project settings using the **Project -> Configuration Options** drop down menu as shown in Figure 6.
   - Change the **Frequency** box to **8000000**. (8 million without the commas. Note that if you get an error regarding F_CPU undefined, it is because you forgot this step.)
   - Change the **Optimization** box to **–O2**.
   - Click on the **Include Directories button** found on the left side.
   - Click on the New Folder icon (upper right corner) and use the browse ellipse "…" to add your project folder you created in Step #3 (appears as .\ )
   - Click OK

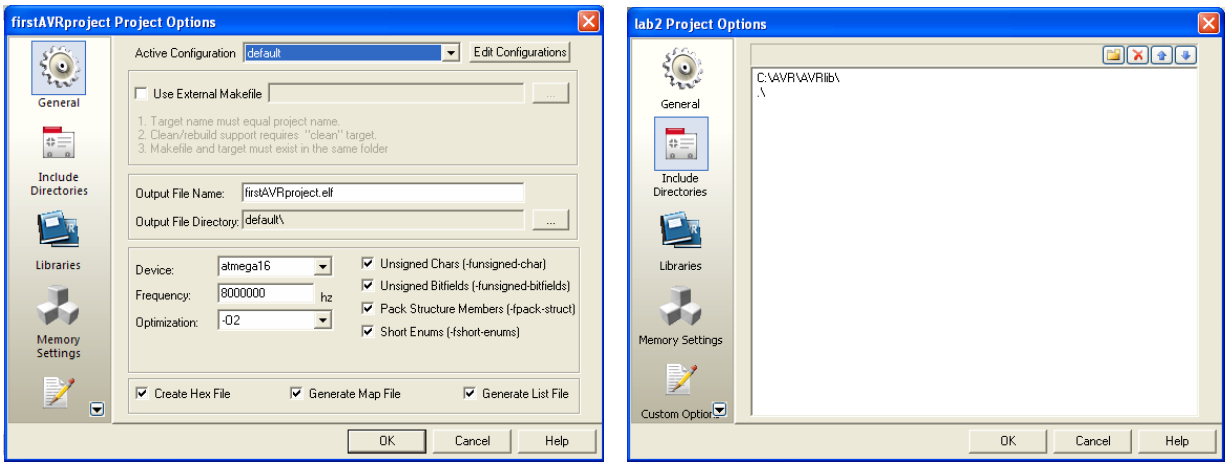

Figure 6. Project configuration options. Any time you create a project using AVR Studio, you should set the Frequency and Optimization as shown above to the left. You also need to include the directory that you have your source code in (denoted .\ ) by going to the Include Directories option, adding it as a new folder, and browsing to where it is on the PC as shown above to the right. If you use code from some other project directory or library, you will need to include the location of where their directories reside. In the example above, AVRlib was also included. You do not need AVRlib for this laboratory.

5. Now navigate to the class website, and download the code for this lab. Save the files into the project file that you created in step 3. The required files are:

uart.c	uart.h	main_intro.c

This code is meant to show you how to send strings of text from the microcontroller to another device, in this case a PC. The protocol being used is RS-232 serial. For more information on RS-232, see:

http://en.wikipedia.org/wiki/RS-232

RS-232 communication will allow us to send binary data between two points, and this will be useful throughout the semester in debugging programs or communicating with external chips, like sensors or even other microcontrollers.

Now go to the Project Directory window on the left hand side of the screen as shown in Figure 7, right-click on the Source Files folder icon, and select 'Add Existing Source File(s)'. In the resulting dialog box, select uart.c and main_intro.c (click on one of the file names, then hold the CTRL key down, and select the other), then click OK. Add uart.h into the Header Files folder following a similar procedure, but first right-clicking on the Header Files folder icon.

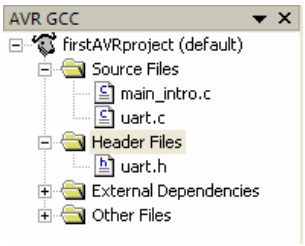

Figure 7. Adding source files. You need to add the C source and header files into the appropriate project folders.

6. You are now ready to compile your first project! From the main menu select **Build → Build**. (You can also use F7 or the Build button on the toolbar in the future). 'Build' compiles your program. If you have any errors, correct them before proceeding. Messages from the compiler will appear in the bottom (Build) window of AVR Studio. If you double-click on a line in this window that indicates an error, it will take you to the offending line in your program. Ask a TA for help if necessary. Always check the Build log window to ensure that the last message states that there are no errors before going further.

   If you had errors, and you fix them, it is advisable to run the 'Clean' routine that will get rid of files generated from any previous attempts to build your project. So, when needed, select **Build → Clean** from the Tools drop-down menu (or use F7 or the button on toolbar in the future) to run Clean.

7. You are almost ready to download your program to the ATmega16 microcontroller! Connect the serial cable from COM1 on your computer to the RS232 CTRL connector on the STK500 board (the connector closest to the power jack). Connect power to the STK500, and turn on the board with the power switch in the upper right hand corner. If the board is functioning properly you should see green lights above VTARGET and STATUS.

8. In AVR Studio, select the AVR Program button as shown in Figure 8 (or use Tools --> Program AVR). A message in the bottom of this window should indicate that you are connected to your processor. If the message indicates that detection of your controller failed, double-check your power and serial cable connections and try again. If you still have problems, get help from your lab instructor.

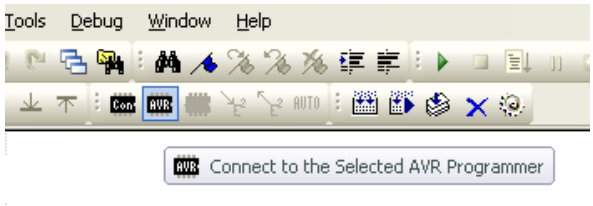

Figure 8. AVR Programmer Toolbar. Clicking the 'Connect' button will open the STK500 Programming Dialog Box. If you see a box that provides you with a list of available programming hardware, make sure that your STK500 is connected and powered on, and then select "STK500" and "Auto."

Now select the **Fuse** tab, and check that the fuses are set properly, as listed in Table 1 and shown in Figure 9. Fuses are bits in the ATmega16 EEPROM that deal with such things as the clock source, oscillator options, brown-out detection, etc. For more information on fuse bits, see the appropriate pages in the ATmega16 manual. The fuse bits only need to be programmed once.

Table 1. Fuse Settings for the ATmega16. Make sure that these are set so that your ATmega16 works properly! Since we are not using a bootloader, it probably won't matter for any of the programs that you write in ME 106 what Boot Flash section size you choose. Choosing the smallest value will give the most space for your program code, however

Fuse Settings to be Checked for the ATmega16 Microcontroller
Boot Flash section size=256 words Boot start address=$1F00;
or
Boot Flash section size=512 words Boot start address=$1E00;
Brown-out detection level at VCC=2.7 V;
Int. RC Osc. 8MHz: Start-up time: 6 CK + 0 ms;

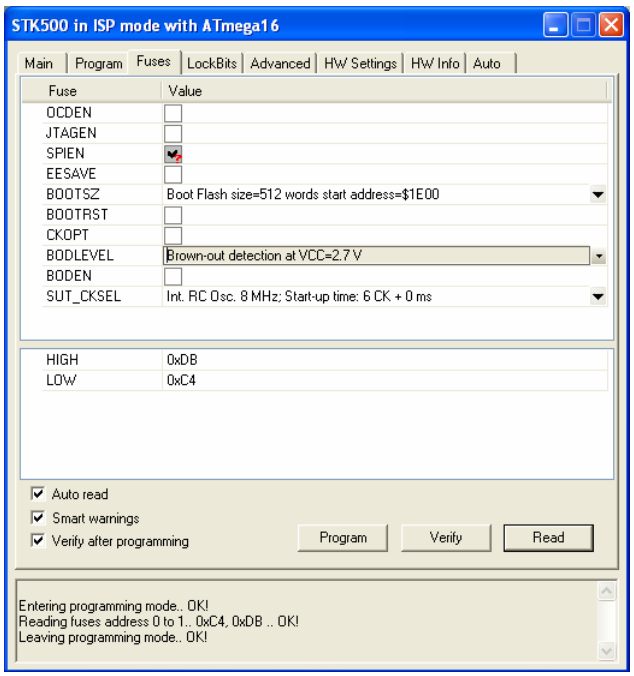

Figure 9. AVR Fuse Selection. These fuses need only to be set the first time a new chip is used.

9. Now select the **HW Settings** tab, and verify that the STK500 clock generator frequency is set to 3.686 MHz and that the ISP frequency field is *LESS THAN* 1/4 of the ATmega16's clock speed (the ATmega's internal clock is set at 8 MHz, so the 1.843 MHz should work fine.) The right

side of Figure 10 shows the HW Settings tab, and the left side shows the Main tab where the ISP frequency can be selected. If you made changes to the fuse settings or ISP frequency, press, 'Program' to burn the fuses to the ATmega16.

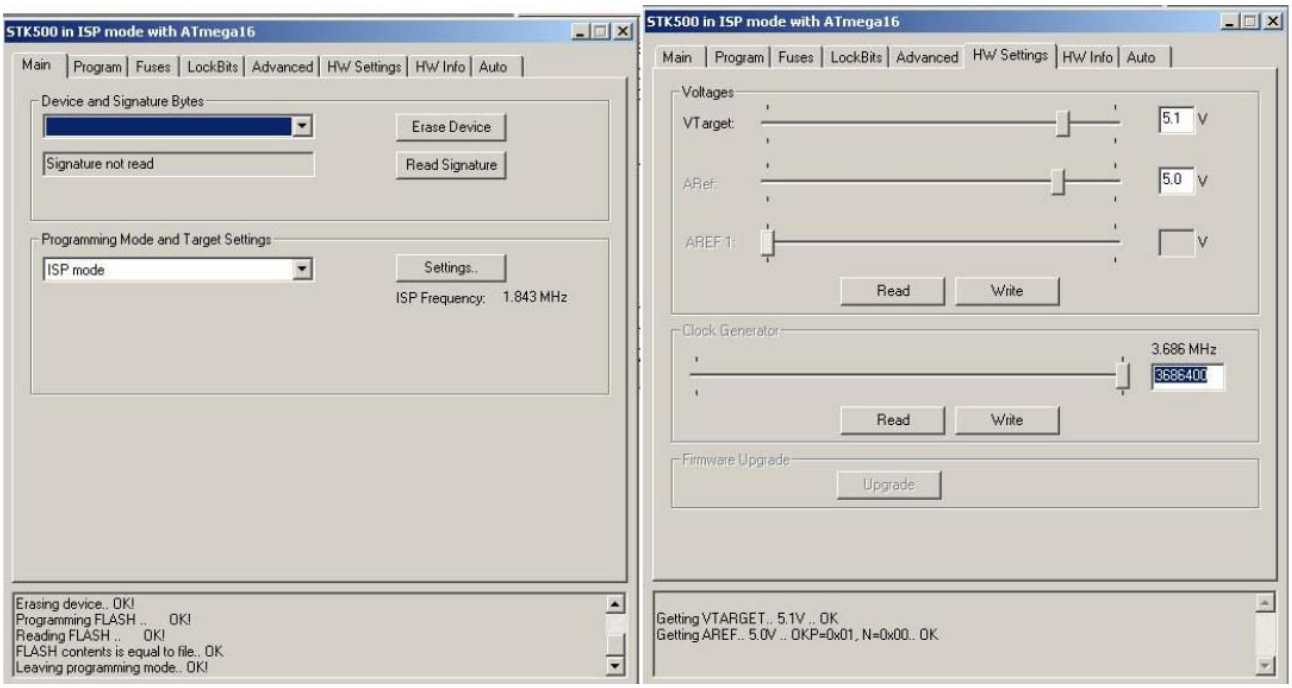

Figure 10. STK500 ISP settings and hardware settings. ISP (In System Programming) is the communication protocol that the STK500 uses to program the ATMega16. If the ISP frequency is set too high, then communication errors will occur. If the ISP frequency is set too low, it will take a long time for your program to be downloaded to the microcontroller.

If both your RS-232 cable AND 6-pin jumpers are installed properly, and you get a dialog box titled, "ISP Mode Error" with some verbiage like, "A problem occurred…", verify that:

- Your STK500 is turned on.
- The serial cable is connected to the "RS-232" (not "SPARE") labeled 9-pin connector.
- The 6-pin ribbon cable is connected between the ISP6PIN and SPROG3 headers
- Hyperterminal is not connected to the COM port that is shared with AVR Studio for programming.

If the problem persists, see Appendix B for another possible solution.

10. Now you're ready to download the program to the chip! See Figure 11 below. Select the **Program** tab. Under the Flash section, use the browse button ("…") next to the 'Input HEX file' dialog box to select the hex file associated with your program. You'll find it by going to the folder you created, and opening the default folder. The hex file will have a .hex extension. The hex file contains the complied program translated into byte code that is readable by the microcontroller.

    Click on the Program button within the Flash section to download the hex file to the microcontroller. You will see the progress bar update for a second or two, and you should see a series of 'OK' messages. If any of these messages are not, 'OK', you may have to ask your lab instructor for help.

11. Now that you have downloaded your program, it should be running! To verify this, you'll use HyperTerminal to capture the output of your program to the monitor of the PC. Power off the

STK500 using the power switch. Disconnect your serial cable from the RS232 CTRL connector, and reconnect it to the RS232 SPARE connector on the STK500 board. We also need to connect the transmit and receive pins from the ATmega16's UART0 port to the RS232 SPARE connector's transmit and receive pins. Use the two-wire jumper provided with your lab kit to connect the PD0 and PD1 pins of the PORTD header to the RXD and TXD pins (RESPECTIVELY! i.e., **PD0 connects to RXD, and PD1 connects to TXD**) of the RS232 SPARE header on the STK500 board. Figure 12 shows this connection. **Note: You will not see output on the serial port unless you make this connection!** (Look in the ATmega16 manual to see the Alternate Functions of the PORTD pins).

For this lab we will also use two 10-pin ribbon cable jumpers, one to connect the 10-pin SWITCHES header with the 10-pin PORTA HEADER, and the other to connect the LED header with the PORTB header. When making these connections, make sure that the red stripe connects pin 1 at each header on both ends of the jumper. See Figure 12 for the proper connection.

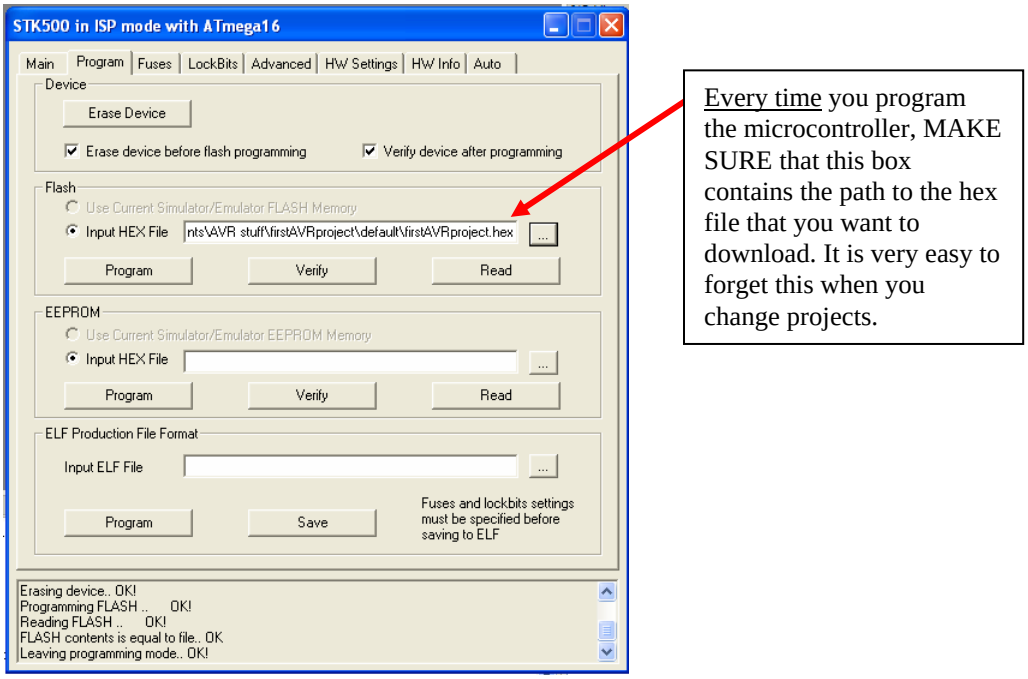

Figure 11. STK500 programming dialog box. Note that when you create a new project, the Input Hex File ***does NOT*** automatically get updated to match your new project settings. Therefore, it is important make sure the desired HEX file is selected every time you open or create a new project.

Now open a serial communications program such as HyperTerminal (which can be found under Start → All Programs → Accessories → Communications). If you are asked to name a New Connection, enter any name you want, and press OK. Select COM1 for, "Connect Using", and press OK. Set COM1 properties with: 9600, 8, N, 1, N, then press OK. Power on the microcontroller, and press the reset button on the STK500. **What is displayed on your screen? Is this what you expected?**

12. You should observe that the floating point number is not displayed properly. Read the comments in the uart.h program header file to find out how to correct this issue.
13. Because you have changed compiler options, you must do a, Build → Clean, before rebuilding your project. Now rebuild your project with, Build → Build, and re-download and re-run your program as before.

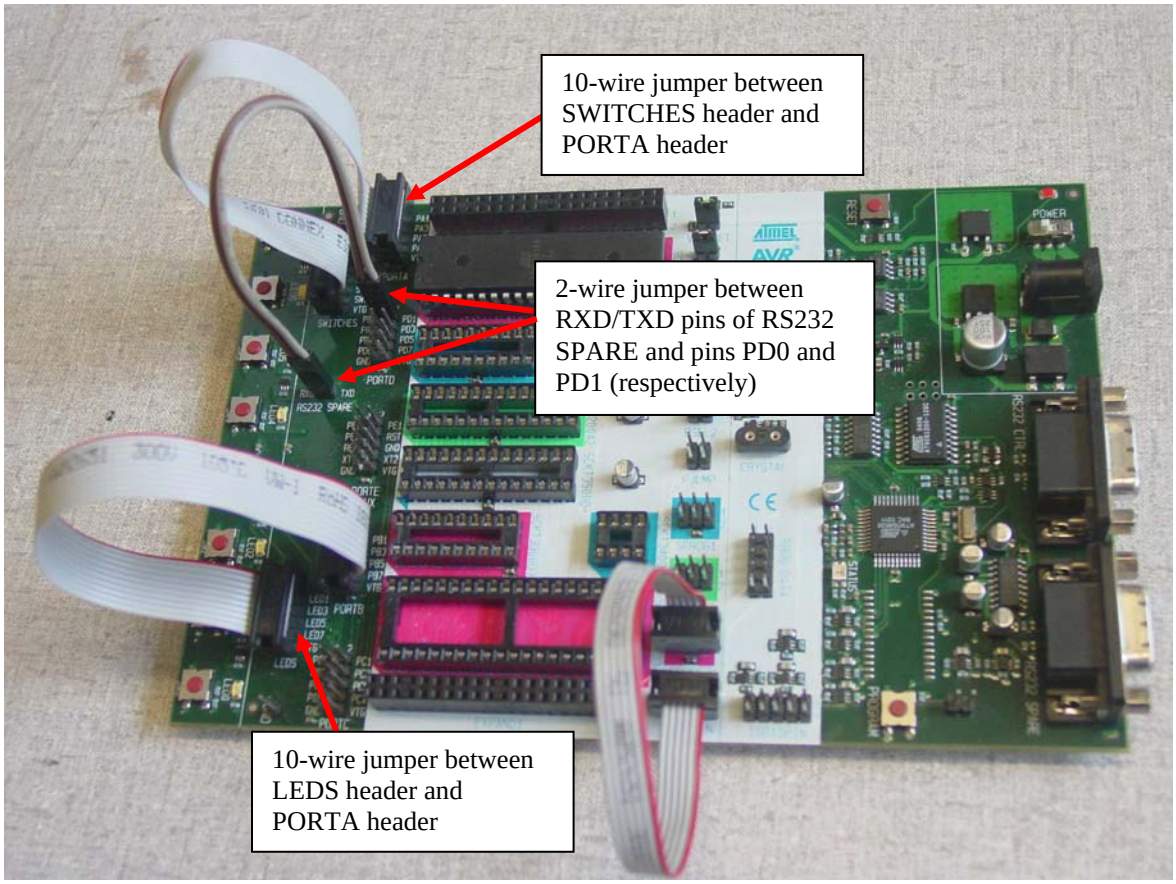

Figure 12. Connection of 2-pin jumper and 10-pin jumpers. The RXD and TXD pins of the RS232 SPARE header connect to pins PD0 and PD1 respectively using a two-wire jumper. Two 10-pin ribbon cable jumpers are used to connect the 10-pin SWITCHES header with the 10-pin PORTA HEADER, and to connect the LED header with the PORTB header. When making all these connections, make sure that the red stripe connects to pin 1 at each header on both ends of the jumper.

Acknowledgment

We acknowledge the generous support of the Atmel Corporation for providing AVR Starter Kits, STK500 boards, and AVR devices for use by the students and faculty of San José State University.

Appendix A – Installing WinAVR and AVR Studio

The following procedure describes how to install WinAVR and AVR Studio. For more help installing on your operating system, refer to www.avrfreaks.net. Note: default directories are used; however, you may change them as desired as long as you make necessary corrections when building projects, etc.

Installing WinAVR

1. Download the latest version of WinAVR from: http://sourceforge.net/projects/winavr
2. Install WinAVR by double-clicking the self extracting installation program: **WinAVR-*version*-install.exe** (where version is the current version number)
3. Accept the default choices unless you have good reasons to do otherwise
4. After installation the README file will open in your browser window. Read this!

Installing AVR Studio

1. Download the latest version of AVR Studio (free registration required) and the latest Service Pack (if any) from: http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725
2. Double click on the installation file.
3. During installation, keep all the default options and program locations. If during the installation AVR Studio asks if you want to install the Jungo USB Driver, check that you want to install it. This will enable you to use USB-based AVR programmers such as the AVRISP mkII and the AVR Dragon.

Congratulations! You are ready to start programming the ATmega16 using WinAVR and AVR Studio.

Appendix B – Handling ISP Mode Error

If you try to program a brand-new ATmega16 microcontroller, you may get a dialog box titled, "ISP Mode Error" with some verbiage like, "A problem occurred…", such as shown in Figure B1. You may get the same dialog box if you neglected to connect the RS232 cable or the 6-pin ribbon cable for ISP programming.

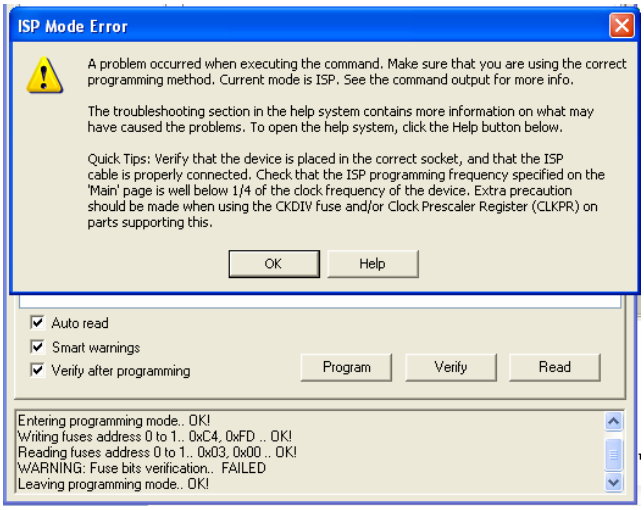

Figure B1. ISP Mode Error. You may get this error trying to program a brand-new Atmega microcontroller or if the RS232 cable or ISP programming cable are not connected properly.

If you get this error, first check that the RS232 cable and the ISP programming cable are both connected properly. If the error continues after you have verified that the cables are connected properly, and re-tried to download your program, you may need to temporarily adjust the ISP programming frequency. To do this:

1. Click the Connect icon to get the programming dialog box open
2. Go to the Main tab, and under the "Programming Mode and Target Settings section, click on the Settings button, and set the ISP frequency to 57.6 kHz as shown in Figure B2.
3. Click on the Write button, then click on the Close button.
4. Return to the Fuses tab, and set the checkboxes and pull-down selections as specified in the instructions earlier, then click, 'Program' to burn the fuses. Verify that the fuses have been set properly (all lines in the log window should end with, 'OK!').
5. Return to the Main tab, and set the ISP frequency back to 1.845 MHz, click on the Write button, then click on the Close button. You should be good to go now!

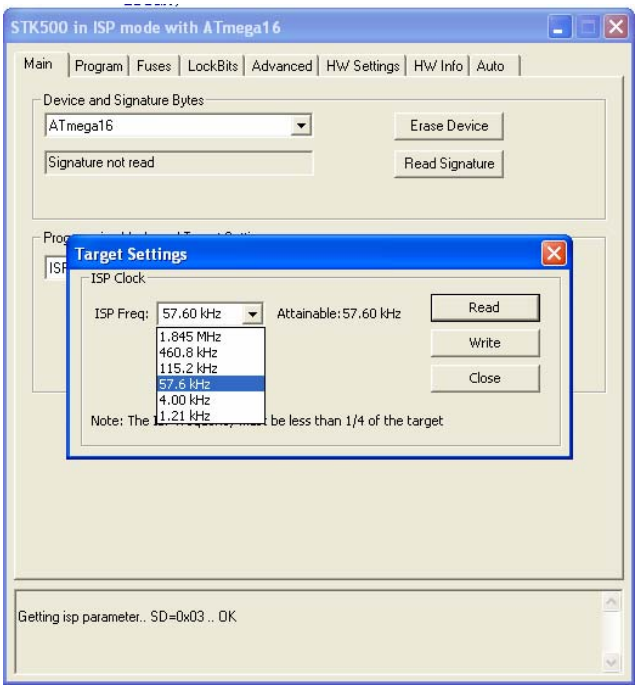

Figure B1. Temporarily changing the ISP frequency. Sometimes with a brand-new microcontroller it is necessary to go to a slower programming frequency, so that the fuses can be set properly. Try 57.6 kHz. After you program the fuses, you should change the frequency back to 1.845 MHz before going on to program your microcontroller.