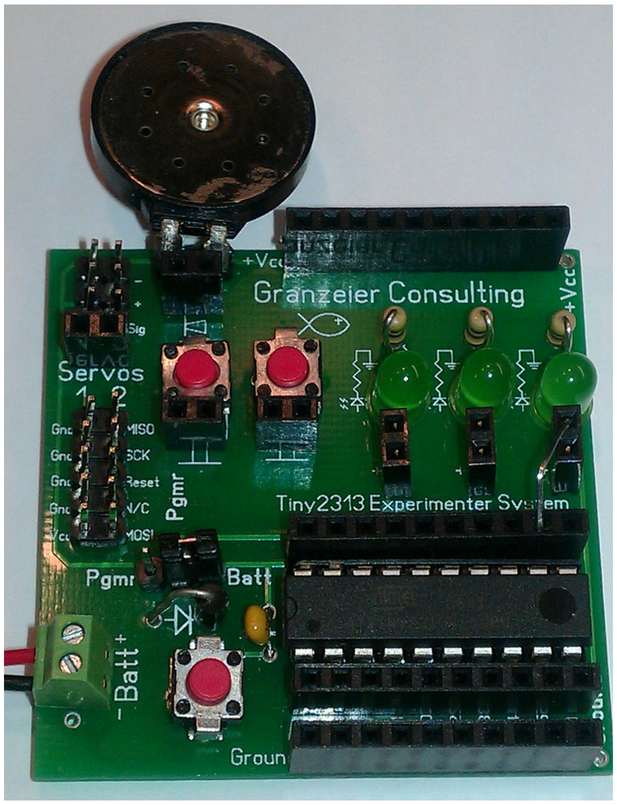
Helping to Build a Better Engineer.



Introduction to Microcontrollers
using the Granzeier Consulting Tiny2313 Experimenter System with BASCOM-AVR.

Author: Art G. Granzeier III, President

Congratulations on your purchase of your Granzeier Consulting Introduction to Microcontrollers book, and the Tiny2313 Experimenter System. This book is to help the absolute beginner get started using their new Tiny2313 Experimenter System. This is one of the easiest, lowest-cost and quickest ways to get into the field of electronic control systems, whether you wish to just have a fun hobby, or you would like to go on to work professionally in industrial controls or robotics. We are assuming that you have access to a computer and that you have used, and are familiar with, Microsoft Windows, but that you are a beginner in electronics and micro-control systems. There will be a moderate amount of soldering required to get started, but after that, you will simply plug wires into sockets and use the built-in LED lights, pushbuttons and speaker on your Tiny2313 Experimenter System.

You do not need to worry about getting shocked because the Experimenter System uses extremely low voltage levels. In fact, you can do much more harm to the Tiny2313, just from the static build-up in your body, than the Tiny2313 can do to you. For that reason, you will need to exercise caution when touching your system. You should always touch the board only by its edges, and you should try to avoid touching any of the components on the printed circuit board. That said, the Tiny2313 Experimenter System is actually a pretty robust system, and with only a minimal amount of caution, the two of you should get along great for many years.

Following this short tutorial will take you a few days, and when you are done, you will be comfortable with your Experimenter System and will be able to set it up, add inputs and outputs to it and then program it to perform control tasks. This is not an advanced tutorial and so we will not cover any of the more advanced (and thus more difficult) tasks. However, after this short tutorial, you will be able to put your Tiny2313 to practical use.

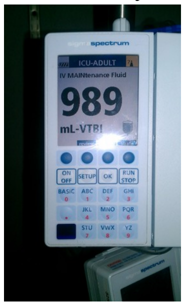
Acknowledgements:
I would like to take this opportunity to acknowledge and thank several people for their help in this book:

- First, I could not have done anything at all unless God had given me the skills and abilities. He is the Ultimate Engineer and the reason that engineering works at all. To learn more about Him, please go to my faith web site (http://faith.granzeier.com), and click on the "Good News" link on that page.
- Second, my wife is the motivating force behind most of my projects. It is her patient behind-the-scenes work that allows me to pursue projects like this one.
- Without the fine work of MCS Electronics, the total beginner would have had quite a bit more effort to get up and running. Their BASIC compiler has brought simple controller development to thousands of engineers and hobbyists for many years.
- Finally, Atmel must be recognized for the fine controllers, with simple programming protocols and very low costs. This has helped an entire generation of hobbyists and engineers get into embedded systems design work.

Many thanks to each of these, and to you the ATtiny2313 professionals of the future.

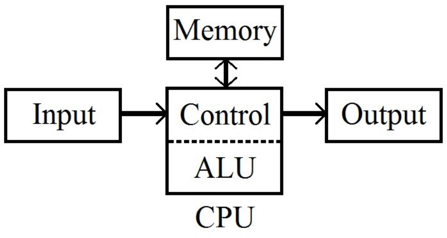
What is a Micro-Controller?

Before you can design with a microcontroller, you really need to know what one is. Microcontrollers are all around you, although you may not even realize it. Many people seem overawed by the thought of a microcontroller. A while ago, I spoke to a woman who, when I told her that I work with microcontrollers, replied that those things were far beyond her. I responded that maybe she could not design with them yet, but that she uses those small computers all the time. She appeared to think that I was nuts to suggest that she could even be involved, in any way, with that kind of "high tech" stuff. These microcontrollers are the brains in many products around the home and office. These microcontrollers can be found in watches, microwave ovens, telephones, cars and trucks, DVD players and robots, in fact almost every product that has a display will use at least one of these microcontrollers. So, most people in industrialized nations are already microcontroller users, even though they don't think of themselves that way.



What is a computer?

A microcontroller is a type of computer, so what is a computer? All computers are made up of four major units. These include the Input unit, the Output unit, the Central Processing unit and the Memory unit. The Input unit is the way that the computer gets the data on which it will do it's work. Most people know about input devices attached to their desktop computers, devices like their keyboard and their mouse. The Central Processing unit performs the work of running the computer's programs. This is made up of a Control Unit and an Arithmetic/Logic Unit (also called an ALU). Together these two units make up the Central Processor Unit, or CPU. While working, the CPU keeps it's instructions, or the program, along with the data on which it is working, in the Memory unit. After the computer does it's computing, it will present the results of that work thru the output unit. Some of the output devices attached to your desktop computer would be the video display and the printer. A *microprocessor* combines the two parts of a computer's CPU (the Control Unit and the ALU) into one single integrated circuit (IC) or chip.



A computer can do a vast number of different jobs. The computers with which you may be most familiar are the ones that sit on or next to your desk and run Windows or Linux or maybe MacOS. These general-purpose computers will help you type a letter or term paper, balance your checkbook or your accounting books, or maybe research different topics by letting you browse the Web. A whole other type of job is for the computer to run (or control) a piece of equipment. By connecting the electrical lines in a computer's input unit to different switches or sensors, and connecting the electrical lines in that computer's output unit to lights, motors and other such devices, a computer can control things like an assembly line, a microwave oven, a stereo, or even something like a plane or space ship. In fact, anything that can be operated by one or more switches can be connected to an output line and controlled by a computer. These computers, which are dedicated to the control of equipment, are called Control Computers, or simply Controllers. In the same way that a Microprocessor combines the two parts of a Control Unit into one IC, a *Microcontroller* combines the four parts of a computer onto one single IC – these ICs are specially crafted to include things useful for controlling equipment. Thus, they are called microcontrollers. As you can see, there is nothing inherently complex about microcontrollers; they are simply small control computers integrated onto a single chip. The microcontroller on your Tiny2313 Experimenter System is an IC (or chip) from Atmel, called the AVR ATtiny2313A.

The Atmel Family

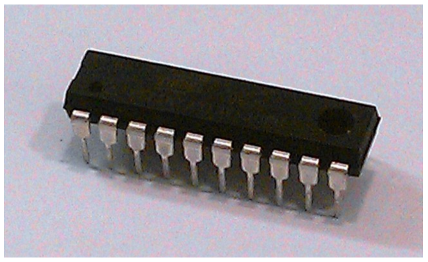
There are many companies currently producing microcontrollers. You may have heard of the PIC microcontroller series from Microchip, or the Texas Instruments MSP-430 family. You may possibly be familiar with microcontrollers from Freescale, Intel, Zilog, Luminary Micro, ST Microelectronics, Harris, Renasas, or any of the other companies out there. There are controllers capable of handling everything from a simple night-light or one of those "musical greeting cards" all the way up to a nuclear reactor, or the flight computer on a space shuttle or the International Space Station (although those higher-end systems are often more a network of control systems rather than a single microcontroller).

After having worked in this field for many years (decades even!), I have been very pleased with the products from Atmel, Inc. All of Atmel's products are very powerful, use very little electrical power, and are very economical (even in single quantities).

When I have compared the different families in packaging, power usage, powerful instructions, ease of programming and usage, price and availability, I usually end up with the Atmel series at the top of my list. There are other good microcontrollers out there, some may even be better than Atmel's in some way or another, but I have long liked what Atmel produces. For my design work, I needed to choose one family, and I chose Atmel's products for this Experimenter System (as well as many other professionally developed projects on which I have worked.)

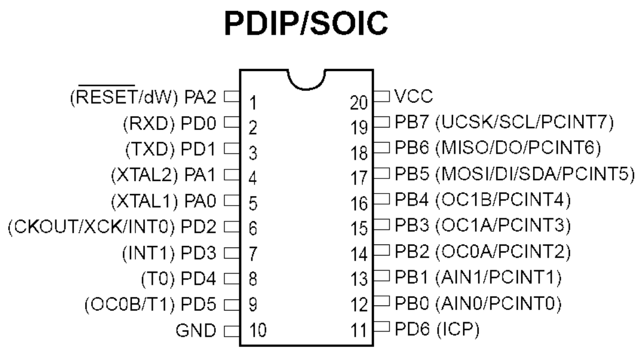
Atmel is a company, headquartered in California, which specializes in building IC chips for the control industry. One of their product lines is the AVR family of microcontrollers. The AVR chips are some of the most powerful microcontrollers available for exceptionally low prices. Atmel produces the specific microcontroller that is used as the control system for your new Tiny2313 Experimenter System.

The Tiny2313



As I mentioned, the microcontroller that we use in the Experimenter System is called the AVR ATtiny-2313A. This is an upgrade of a slightly older chip, the AVR-ATtiny2313. You may notice me referring to the older chip throughout this book; the differences are so small, that (for the purposes of this book) you may ignore them. It is also common practice to refer to microcontroller chips by various shortenings of the full name; thus you will see your controller referred to as the ATtiny2313, the Tiny2313A and even simply the '2313, too. The Tiny2313A is one of Atmel's mid-range microcontrollers. It has up to eighteen input/output (I/O) pins, including two inputs to an analog comparator. Each I/O pin can read a sensor, such as a temperature or light sensor or a switch, or can activate an output device, such as an LED indicator, a motor or a speaker for audio alerts, or even a relay for high-power applications. In fact, any electrical device that can be turned on or off can be controlled by an output pin of a microcontroller. Another two of the I/O pins can be dedicated to a serial I/O port for communication with another controller, or with a desktop/notebook computer. There are 2K bytes of flash program memory, 128 bytes of RAM and 128 bytes of data storage EEPROM. If you don't understand these terms, don't worry; you will soon be able to make use of this great controller.

This is an incredibly powerful microcontroller, and yet with just a tiny bit of effort, you can have it performing tricks for you as well as your dog does (or better if your dog is like ours – I think that our dog is part cat, normally he wants us to do the tricks.)



The ATtiny2313A chip, from Atmel, is a microcontroller with a lot of capabilities. For the purposes of this introduction, we can look at the Tiny2313 as a microcontroller with seventeen Input or Output (I/O) pins. These pins are broken down among three different ports, labeled Port A through Port D (Port C is actually only available on the '2313's bigger brothers.) We refer to those pins by the letter P (for Port) and then the port letter (A-D.) Each port contains, up to, eight pins, labeled 0 through 7. Because the ATtiny2313A is actually a scaled-down microcontroller, not all of the ports have all eight pins. Look at the diagram of the Tiny2313 chip (above, or in appendix B) and you will see that they range from Port A having only 2 pins (PortA2 is actually used as the reset pin, changing that is possible, but then you lose the ability to easily program the controller) to port B having all 8 pins. Each one of those port pins on your Tiny2313 can serve as either input or output pins.

In this book, you will be connecting simple devices to some of the Tiny2313's I/O ports. We will then go through learning how to program your Tiny2313 to make use of those devices. Along the way, you will become familiar with the fundamentals of computer programming. You will also learn a little about electronics, as it relates to control systems. Most importantly, as Bill Cosby used to say: "if you're not careful, you just might learn something."

Quiz 1

1) A microprocessor combines which parts, of a computer, onto a single IC? _____

   A) Input and Output
   B) RAM and ROM
   C) Control and ALU
   D) CPU and Memory

2) A control computer is a computer which is designed to do just about any general task, such as balancing your checkbook, writing letters or browsing the internet. (true or false) _____

3) The microcontroller on your Tiny2313 Experimenter System is: _____

   A) The Atmel AVR-ATtiny13
   B) The Atmel AVR-ATtiny2313A
   C) The Motorola MC68HC11
   D) The Texas Instruments MSP-430

4) A fan would be connected to which major unit of a computer? _____

   A) Input
   B) Control
   C) Memory
   D) Output

5) By connecting switches to the _____ lines on a control computer, the computer can tell which switch has been closed and which has not.

   A) Memory
   B) Input
   C) Output
   D) Control

Extra Credit:

How many pins on the Tiny2313 chip can be used for Input/Output, without disabling the reset? _____

   A) 10
   B) 17
   C) 23
   D) All of them

Tell me what you want me to do.

A computer is only as useful as a doorstop between uses, without a program to tell it what to do. The process of giving the computer instructions to perform a task is called programming. A program is pretty much just a list of instructions to the computer. Since a computer only knows about the presence or absence of voltage levels, the Tiny2313 chip uses voltage levels to tell it what to do. All of the instructions and data inside the Tiny2313 are actually voltage levels of either (about) 5 volts or 0 volts. We represent these voltage levels as ones and zeros for our convenience, not for the computer's convenience. However, as good as those representations are, they are still not very easy for humans. Shortly after computers started coming on the scene, humans wrote tools to help with programming these machines. Some of the first tools were language translators, which took a more human-like language, and translated that into ones and zeros for the computer. Higher level languages, or languages which are closer to human languages, make it easier for a programmer to tell the computer what to do. The process of translating these "high" level languages into the computer's machine language is called compiling.

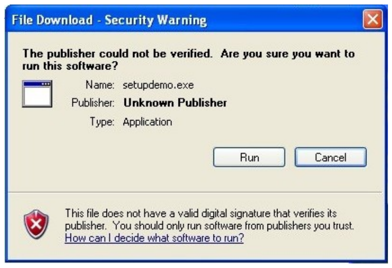
MCS Electronics produces a language compiler called BASCOM-AVR for Atmel's AVR family. BASCOM is a variant of the BASIC programming language. This language was invented by two professors, at Dartmouth College in the 1960s, specifically for beginners. In fact, the name BASIC is an acronym, which stands for Beginners All-purpose Symbolic Instruction Code; in other words, it is a general (*All-purpose*) *Code*, for *Beginners* to give the computer *Symbolic Instructions* to perform a job. It was designed to be easy to learn and to allow non-computer students to quickly be able to write programs for the university's computer system. The free trial version, of this BASIC, is limited only in the size of the program. This limit is 4K bytes, which makes BASCOM-AVR an easy match for learning about microcontrollers with the Tiny-2313. You can download this language from MCS's web site; we will do that a little later in this tutorial.

We will be using this language for this book because it is a dialect of BASIC, it is also extremely easy to learn, and yet powerful enough to perform many useful tasks. At the end of this tutorial, I will be giving you some ideas for continuing on with the Tiny2313, some of the advanced topics will include more advanced programming using Assembly or C. As I mentioned earlier, don't worry if you do not know those terms, by the time you are done with this introduction, you will be ready to learn those other languages.

One more caveat: throughout this book, I will be giving you one way to complete each step. There is, almost always, more than one way to do anything with computers. When I tell you to do a task in one way, if you know another way, and are able to compensate for the differences between how I describe to do the job, and your way, by all means do it the way you feel comfortable. As an example, in a program, I may tell you to click on File and then click on New to create a new document. If you are comfortable with just typing Ctrl-N to create the document, then do that. The same idea will apply if you would rather type Ctrl-S to save a document, rather than following my (longer, but simpler) directions. Just make sure that you understand the process well enough that you can keep up with our ideas in this book.

<u>Getting the Latest Version</u>
In this book, we will be installing and using BASCOM-AVR. You will need to get the newest version from MCS Electronics, the publisher:

1. Go on the web to http://www.mcselec.com and click on Downloads on the left side of the screen.
2. On the Downloads page, click on the BASCOM link at the bottom of the Downloads list on the right side of the page.
3. On the BASCOM page, click on the BASCOM-AVR link at the top of the list of downloads.
4. On this page, click on the link to download the BASCOM-AVR Demo version.
5. On the Terms and Conditions page, click the radio button for "I agree" and then click the button to proceed. This will open the download dialog box, make sure that the "Save file" radio button is selected and click the OK button.
6. In the Save to dialog box, select your desktop and click the Save button.
7. Click on the back button on your browser to return to the BASCOM AVR download page and also download the manual. This has lots of information and is very well organized.
8. Also, on the BASCOM-AVR page, you will find more information about BASCOM-AVR, including articles written about this compiler.
9. Close, or minimize, your browser.

<u>Installing BASCOM-AVR</u>

File Download - Security Warning

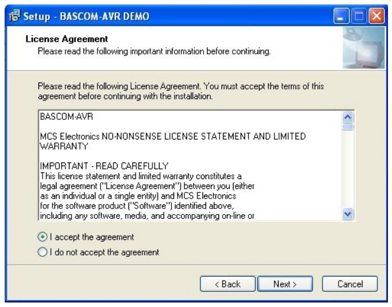
The publisher could not be verified. Are you sure you want to run this software?

Name: setupdemo.exe
Publisher: **Unknown Publisher**
Type: Application

Run | Cancel

This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust.
How can I decide what software to run?

The version of BASCOM-AVR that was current when this book was written was 2.0.7.5, and so that is what is shown in this book. While MCS Electronics does make changes to the program, the commands that we use in this book are so fundamental, that the changes should not affect the programs you will write. That said, there may very well be some changes to the screen shots, and other examples that I give you. You will need to adapt what you see in this book, to what you see on the screen. Read the screen carefully, and you should be able to figure out what to do. Chapter 2 of the BASCOM manual, covers installation, and gives far more detail than I can in this book.


Setup - BASCOM-AVR DEMO

License Agreement
Please read the following important information before continuing.

Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

BASCOM-AVR

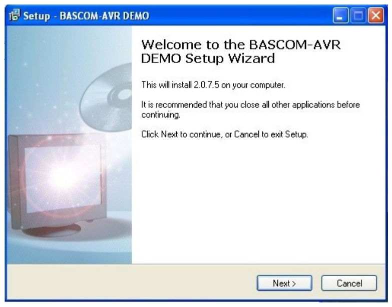
MCS Electronics NO-NONSENSE LICENSE STATEMENT AND LIMITED WARRANTY

IMPORTANT - READ CAREFULLY
This license statement and limited warranty constitutes a legal agreement ("License Agreement") between you (either as an individual or a single entity) and MCS Electronics for the software product ("Software") identified above, including any software, media, and accompanying on-line or

[✓ I accept the agreement]
I do not accept the agreement

< Back | Next > | Cancel

Open the zipped file that you downloaded (in Windows XP and higher, you can just double-click the icon on your desktop.) Now, launch the BASCOM Installer, by double-clicking on the new SetupDemo.exe icon on your desktop. If you get the Windows security warning about Windows not being able to verify the publisher, click on the Run button. This will begin the installation. The first screen that the installation gives you will simply be a welcome screen. Click on the Next> button to begin the installation.


Setup - BASCOM-AVR DEMO

Welcome to the BASCOM-AVR DEMO Setup Wizard

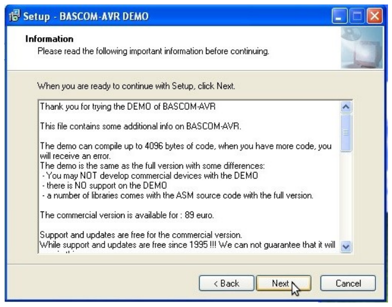
This will install 2.0.7.5 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Next > | Cancel

Now, the installation will display the License Agreement. Basically, this says that MCS Electronics put some real effort into creating and developing BASCOM-AVR, and that you agree not to try to steal from them. They do give us the right to use their programming system as hobbyists. However, this is in the hopes that you will find this a great program and purchase the full program for developing your own software. This is a great package (trust me, I've used over 100 different programming environments) and you will really want to purchase the full package if you continue on with microcontrollers – so start saving now. After you read and agree to the License Agreement, click Yes. If you click No, the program will not install and you will need to find your own way of programming your Tiny2313 Experimenter System (there are other ways, but they are a little more difficult and we will not cover them in this book.)



Information
Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

Thank you for trying the DEMO of BASCOM-AVR

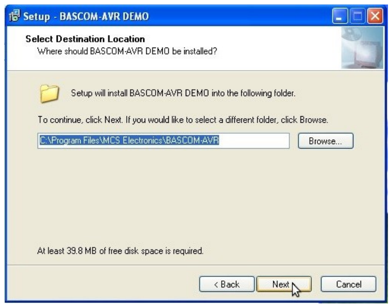
This file contains some additional info on BASCOM-AVR.

The demo can compile up to 4096 bytes of code, when you have more code, you will receive an error.
The demo is the same as the full version with some differences:
- You may NOT develop commercial devices with the DEMO
- there is NO support on the DEMO
- a number of libraries comes with the ASM source code with the full version.

The commercial version is available for : 89 euro.

Support and updates are free for the commercial version.
While support and updates are free since 1995 !!! We can not guarantee that it will

After clicking on the Yes radio button, click on the Next> button and the Information file will be displayed. Read over this note file for any information more recent than this book. When you are done with reading the notes, click on the Next> button again.



Select Destination Location
Where should BASCOM-AVR DEMO be installed?

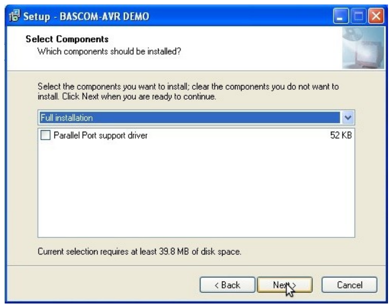
Setup will install BASCOM-AVR DEMO into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\MCS Electronics\BASCOM-AVR

At least 39.8 MB of free disk space is required.

When you get to the Destination Location screen, unless you have a reason not to, just click Next> to accept the default locations. After this, the install program will ask you if you would like to install additional components. Leave the selection to Full installation, and leave any options unchecked and click Next>.



Select Components
Which components should be installed?

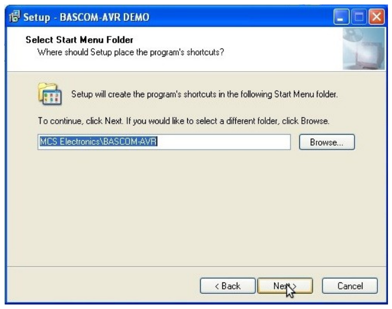
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

Full installation

☐ Parallel Port support driver 52 KB

Current selection requires at least 39.8 MB of disk space.

The last thing that the installer needs before it begins to install the files for BASCOM-AVR is which Program Manager group to put the installation into. Again, just click Next> to accept the default BASCOM-AVR group. At this point, the installation is ready to actually start copying files onto your hard drive.

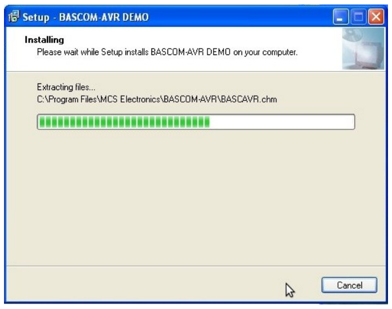


Select Start Menu Folder
Where should Setup place the program's shortcuts?

Setup will create the program's shortcuts in the following Start Menu folder.

To continue, click Next. If you would like to select a different folder, click Browse.

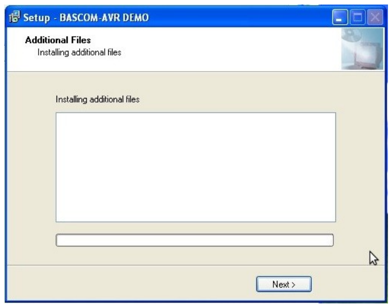
MCS Electronics\BASCOM-AVR



Installing
Please wait while Setup installs BASCOM-AVR DEMO on your computer.

Extracting files...
C:\Program Files\MCS Electronics\BASCOM-AVR\BASCAVR.chm

Go ahead and click on Next> again to begin the copying. The installation program will begin copying files and will keep you advised as to it's progress by two "progress bars." After the files have been copied onto your system, the install program will show a screen for installing additional files, click Next>. After BASCOM has been installed, your computer will need to be restarted, or BASCOM will not work properly. The install program will ask you to restart at the proper time. Again, leave the Yes radio button selected and click the Finish button. Your computer will restart automatically.

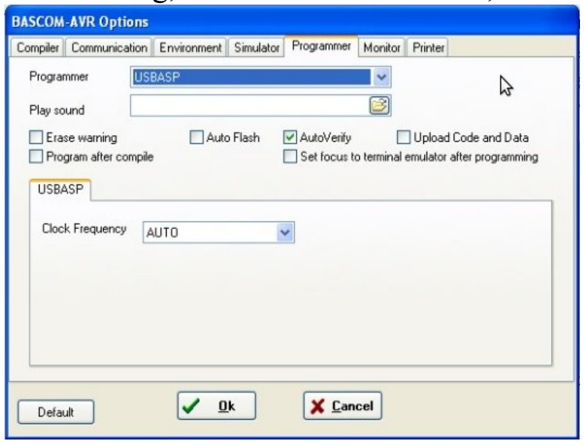


Additional Files
Installing additional files

Installing additional files

Setting up BASCOM for your Tiny2313 Experimenter System controller

After your computer has BASCOM-AVR installed, there is a setting, which we need to make, to prepare BASCOM for our specific setup. BASCOM has, built-in, the capability of working with several different programmers. In order for BASCOM to be able to use our programmer, we need to tell BASCOM which programmer cable we are using. The cable, which is included in your package, is called the USBASP Programmer. That programmer is built into BASCOM and so our Tiny2313 controller can be programmed directly from within BASCOM.

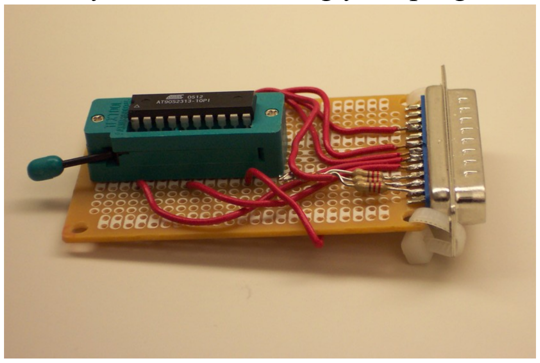


Programmer: USBASP
Play sound:
☐ Erase warning ☐ Auto Flash ☑ AutoVerify ☐ Upload Code and Data
☐ Program after compile ☐ Set focus to terminal emulator after programming
USBASP
Clock Frequency: AUTO

Launch BASCOM AVR. Now set the specific programmer that you will use by clicking on the Options menu and then clicking Programmer. In the Programmer field, use the drop-down box to select the USBASP programmer. Click OK to save your selection. That’s it for setting up BASCOM, simple, right?

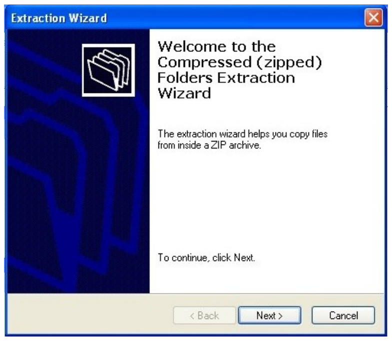
The Downloader Cable

While you will be writing your programs for the Tiny2313 on your PC, you will need to transfer that program to the Tiny2313 chip, itself, in order to let the controller run the program. To put the program into your microcontroller, you must have some way to get the program from your PC into the chip. The way engineers used to do this was by taking the chip and placing it into a special device called a programmer. After running the programming software, you would take the chip with the program in it and place it into your circuit. If, or rather *when,* you discovered an error in your program, you would take the chip out of the circuit, put it into a special device to erase the program (often taking hours), put it back into the programmer, and start the whole process again. This would happen over and over until either you got the program correct, or (often) you just got tired and quit.


An older device programmer for the Atmel ATtiny2313 chip. This is called the Fun Card programmer.

We will be using what is called an ISP Downloader Cable. The ISP abbreviation stands for In System Programmer; this is a way that engineers have developed to get around that old way of using a dedicated programmer. We have a special ISP connector on the microcontroller board, and will just connect the programming cable to the board in order to download the program. All of the programming activity is accomplished on the circuit board and the microcontroller chip itself. No need to pull the chip out and move it around.

The ISP that you received with your kit is called a USBASP Programmer. We mentioned that earlier, when talking about how to set up BASCOM-AVR. In order for your computer to be able to use a piece of hardware, you will need to tell it how. The way that you do this is to load a program, called a *driver*. A driver is just a program that tells your computer how to talk to an attached device. There are drivers, built in to Windows, to tell your computer how to use the mouse, how to read the keyboard, how to display pictures and text on the screen. The USBASP Programmer is not something that everyone uses, though and so, we need to install the driver for that Programmer.

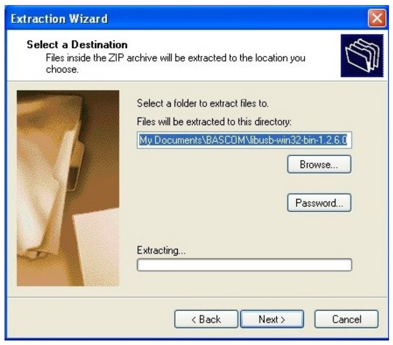


Welcome to the Compressed (zipped) Folders Extraction Wizard

The extraction wizard helps you copy files from inside a ZIP archive.

To continue, click Next.

As you did with the BASCOM compiler itself, you will need to download the driver for your USBASP Programmer. To do this, go to http://sourceforge.net/projects/libusb-win32, and download the driver, by clicking on the green download button. You will need to wait about five seconds for the download to start. Save this file to your desktop, just as you did with the BASCOM install file. When the download is done, close out, or minimize, your browser.


Extraction Wizard

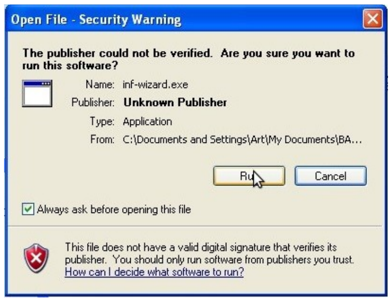
Select a Destination
Files inside the ZIP archive will be extracted to the location you choose.

Select a folder to extract files to.
Files will be extracted to this directory:
My Documents\BASCOM\libusb-win32-bin-1.2.6.0

Extract the zipped file, that you downloaded, by right-clicking on the zipped file and dragging it to a blank spot on your desktop. The Extraction wizard dialog box will open. Click on Next> to begin the extraction, and then Next> again to begin. After the install files are extracted, close the wizard by clicking the Finish button. The wizard will open the zipped file folder where you will see another folder with a name, which starts with “libsub-win32” and then includes version numbering information. Open this folder and then open the “bin” folder. In this folder, you will see an icon for inf-wizard.exe.

inf-wizard.exe
libusb-win32 Inf-Wizard
http://libusb-win32.sourcefor...

This is the install program. Launch the installer by clicking on it’s icon. Again, you may see a security warning; click the Run button. Next you will get an Information window; this asks you to make sure that your device is connected to the computer. The device is your programmer cable. Plug the USB end of the programmer into an empty USB port on your computer, and then click on Next>. *Important, do not continue without connecting your programmer to your computer. If your programmer is not connected to your computer, your computer will not be able to detect the device and so would not be able to select the correct files to create the installer package.

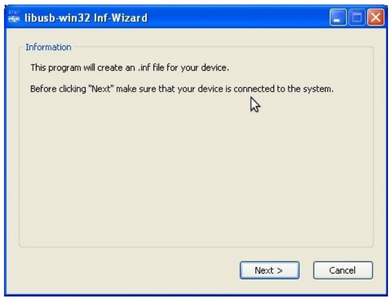

Open File - Security Warning

The publisher could not be verified. Are you sure you want to run this software?

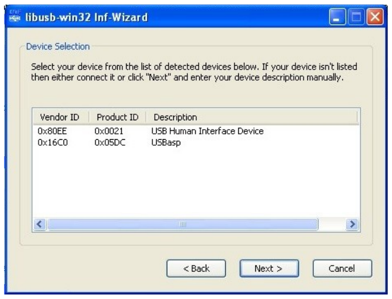
Name: inf-wizard.exe
Publisher: Unknown Publisher
Type: Application
From: C:\Documents and Settings\Art\My Documents\BA...

☑ Always ask before opening this file

This file does not have a valid digital signature that verifies its publisher. You should only run software from publishers you trust. How can I decide what software to run?


libusb-win32 Inf-Wizard

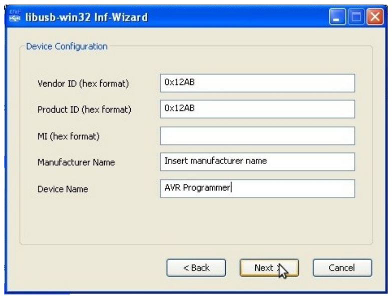
Information
This program will create an .inf file for your device.
Before clicking "Next" make sure that your device is connected to the system.


libusb-win32 Inf-Wizard

Device Selection
Select your device from the list of detected devices below. If your device isn't listed then either connect it or click "Next" and enter your device description manually.

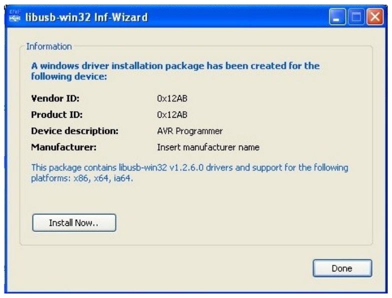
Vendor ID	Product ID	Description
0x80EE	0x0021	USB Human Interface Device
0x16C0	0x05DC	USBasp

A window will open which shows the USB devices that your computer can detect. Make sure that the USBasp device is showing in the window, and then click Next>. Next, a window will pop up giving some information about your device. In this Device Configuration window, you may enter a description (such as “AVR Programmer”) in the Device Name field, and then click Next>. Be careful not to change any other fields in this Configuration window.


libusb-win32 Inf-Wizard

Device Configuration

Vendor ID (hex format)	0x12AB
Product ID (hex format)	0x12AB
MI (hex format)	
Manufacturer Name	Insert manufacturer name
Device Name	AVR Programmer

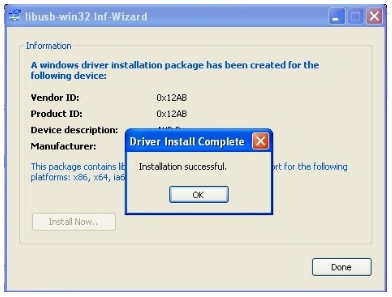

libusb-win32 Inf-Wizard

Information
A windows driver installation package has been created for the following device:

Vendor ID:	0x12AB
Product ID:	0x12AB
Device description:	AVR Programmer
Manufacturer:	Insert manufacturer name

This package contains libusb-win32 v1.2.6.0 drivers and support for the following platforms: x86, x64, ia64.

In the Save as dialog box, just click the Save button. Another information window will appear telling you that it has created an installer package, click the Install now button. A small Installing window will appear, followed by a Driver Install Complete window. Click on the OK, button to close the installer program.


libusb-win32 Inf-Wizard

Information
A windows driver installation package has been created for the following device:

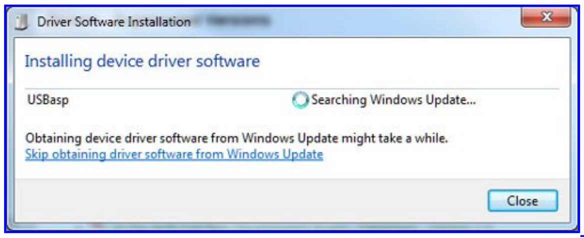
Vendor ID: 0x12AB
Product ID: 0x12AB
Device description:
Manufacturer:

This package contains lib... ...rt for the following platforms: x86, x64, ia6...

Driver Install Complete
Installation successful.

Alternate instructions for Windows 7:

You will need to download the driver, as we described above. You may download it from http://www.protostack.com/download/USBasp-win-driver-x86-x64-ia64-v1.2.4.zip. Save the file on your desktop and unzip it. Next, insert the USBASP into your USB port. The system will attempt to install a driver from Windows Update...



Driver Software Installation

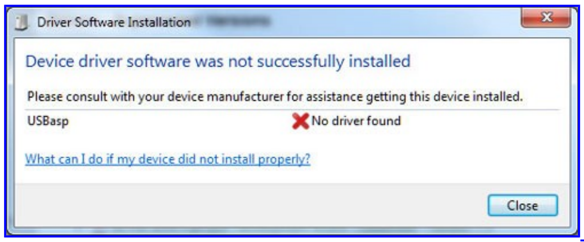
Installing device driver software

USBasp Searching Windows Update...

Obtaining device driver software from Windows Update might take a while.

Skip obtaining driver software from Windows Update

but, it will fail.



Driver Software Installation

Device driver software was not successfully installed

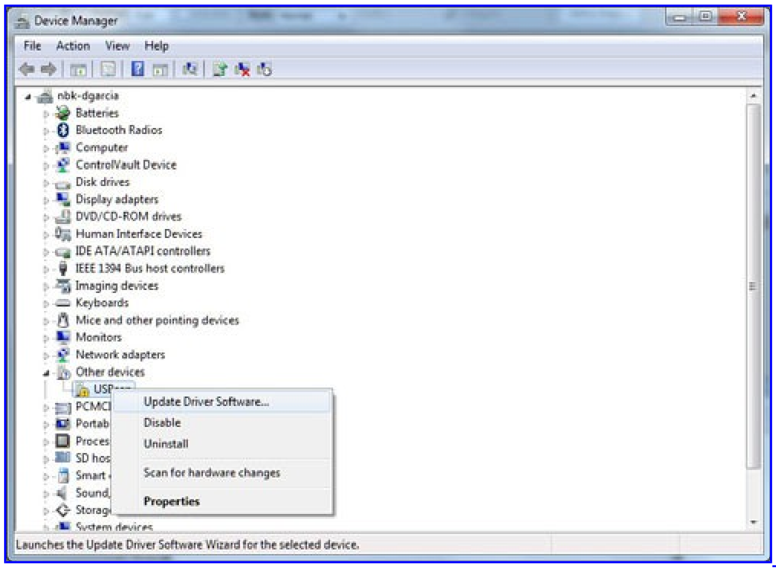
Please consult with your device manufacturer for assistance getting this device installed.

USBasp No driver found

What can I do if my device did not install properly?

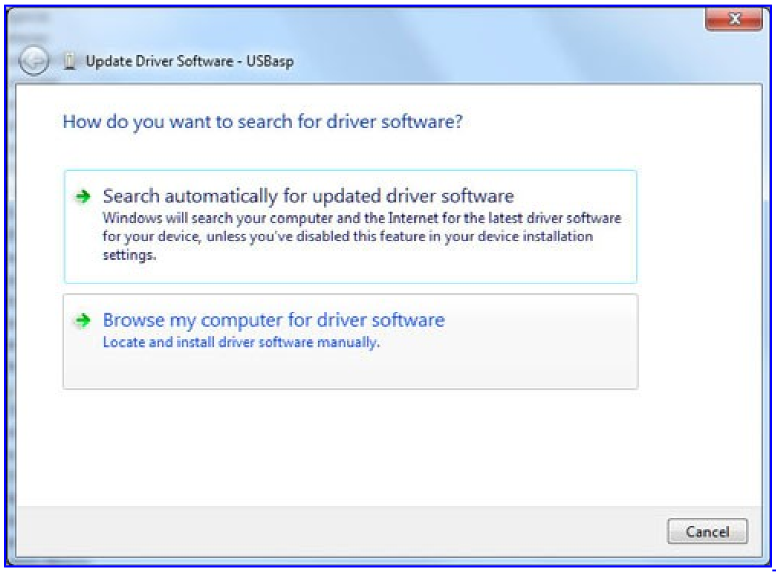
Click on the Start button, and then click on Control Panel. Click on System and Security, and then under System, click on Device Manager. If you're prompted for an administrator password or confirmation, type the password or provide confirmation. In the list under your computer's name, find the entry for the USBASP programmer. It should be displayed with a yellow alert icon next to it.

Right click on the device and select "Update Driver Software"



Launches the Update Driver Software Wizard for the selected device.

When prompted "How do you want to search for driver software", select "Browse my computer for driver software”



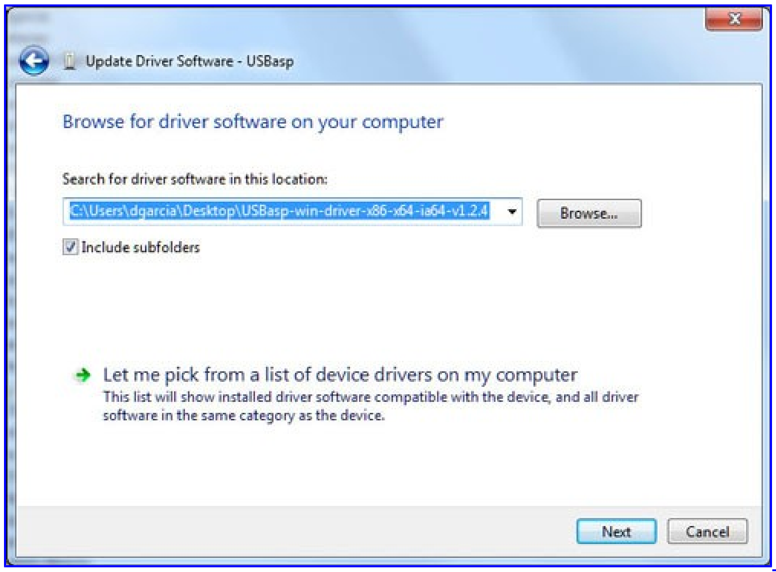
Update Driver Software - USBasp

How do you want to search for driver software?

Search automatically for updated driver software
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.

Browse my computer for driver software
Locate and install driver software manually.

Select the folder, on your desktop, which was created when you unzipped the driver files then click “Next.”



Update Driver Software - USBasp

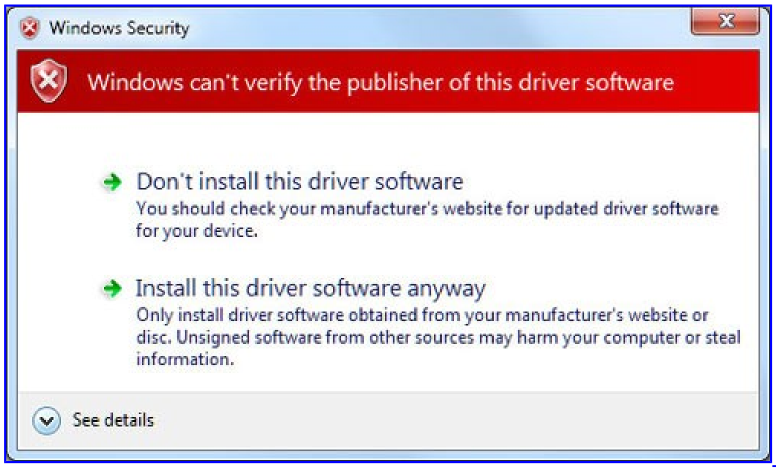
Browse for driver software on your computer

Search for driver software in this location:
C:\Users\dgarcia\Desktop\USBasp-win-driver-x86-x64-ia64-v1.2.4

☑ Include subfolders

Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.

At this point, windows will give you a nice red warning dialog with the message “Windows can’t verify the publisher of this driver software”.


Windows Security

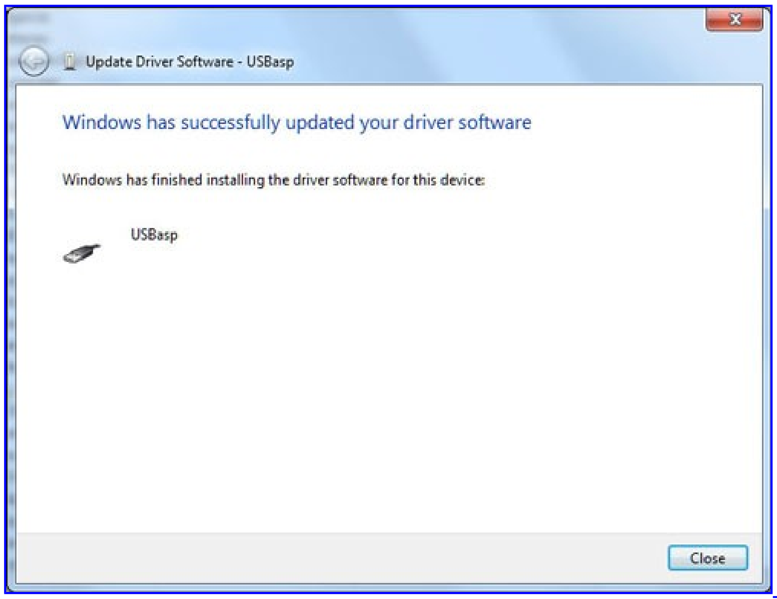
Windows can't verify the publisher of this driver software

Don't install this driver software
You should check your manufacturer's website for updated driver software for your device.

Install this driver software anyway
Only install driver software obtained from your manufacturer's website or disc. Unsigned software from other sources may harm your computer or steal information.

See details

Click "Install this driver software anyway" and the driver will install.


Update Driver Software - USBasp

Windows has successfully updated your driver software

Windows has finished installing the driver software for this device:

USBasp

Close

Congratulations, your computer is now ready for you to start writing programs for your Tiny2313 Experimenter System.

Quiz 2

1) What is the native language of computers? _____

   A) Electrical signals of “high” or “low”
   B) BASIC
   C) English
   D) Swahili

2) The process of translating a more human-like language into a computer’s language is called? _____

   A) Hacking
   B) Transliteration
   C) Translation
   D) Compiling

3) The BASIC programming language was designed to help computer science students with extremely complex programming tasks, and is not suitable for a beginner. (true or false) _____

4) What is a computer program? _____

   A) A list of instructions to tell you how to use the computer
   B) A device which translates the computer’s instructions from a human-like language
   C) A set of directions that you give the computer to tell it how to perform a task
   D) A combination of the Control Unit and the Arithmetic/Logic Unit

5) What is the main purpose of an ISP Downloader Cable? _____

   A) To provide power to the microcontroller
   B) To provide a way to transfer the program from the host computer to the microcontroller
   C) So that the microcontroller can verify that it is running a legal copy of the program
   D) To provide a physical connection between your development system and the microcontroller

Extra Credit:

The BASIC language was created by two: _____

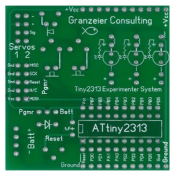
   A) Professors at Dartmouth
   B) Engineers at Cornell
   C) Accountants with the Federal Government
   D) Math students in a Doctoral program

Let’s build this thing!

Preparing the Tiny2313 Experimenter System for this tutorial

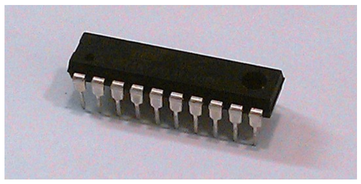
Identifying the Components
In order to assemble your controller board, you must be able to identify the correct parts in order to insert them in the proper place on the printed circuit board. There are eight different types of components that you need to know:

- the printed circuit board
- sockets and integrated circuits
- resistors
- capacitors
- diodes
- light emitting diodes (LEDs)
- speaker
- connectors and headers



The printed circuit board, or PCB, - contains the circuit traces printed right onto a fiberglass board. The Tiny2313 Experimenter System PCB is 2” by 2” and has a silkscreen printing on it showing where each component is supposed to go. If you look closely, you can see the copper traces running on the board from components to other components.

Sockets and ICs - are rectangular devices with several small, silver-colored pins along each longer side. There is only one rectangular IC on the controller and it has a socket for easier trouble-shooting and repair. In integrated circuits, the actual circuit is much smaller than the “chip” that you see. The IC itself is actually about the size of the fingernail on your pinky finger, and is packaged in a plastic or ceramic “carrier”. The carriers come in many different shapes and sizes, but your Tiny-2313 chip is packaged in what is called a PDIP-20 (or a Plastic Dual In-line Package with 20 pins). This means that there are two (dual) rows of pins, which are in-line (in parallel, on opposite sides of the rectangle) with each other. The two rows each contain 10 pins for a total of 20 pins. The socket for this chip is the same size and shape as the Tiny-2313, but has two rows of holes in the top for the legs, or pins, of the chip. IC’s have a mark on them to identify pin number 1. This is often a small circle or triangle in the corner of the carrier nearest pin 1, there may also be a notch in the end, which has pin 1. The pins are numbered so that if you look at the chip with the notch up or the circle in the upper left corner, pin number 2 is directly below pin 1. Going down, you have pins 3 then 4 and so on until you get to the bottom-left corner. Continue with the numbering by crossing over to the bottom-right pin and continue up the right side until you get to the highest number pin directly across from pin 1. Refer to the diagram of the Tiny-2313 on page 6.


A 20-pin DIP package.

Resistors - are small, cylinder-shaped devices with a wire lead coming out of each end of the cylinder. The resistors used in controller electronics are normally about 1/8" in diameter and about ¼" long. They are usually either brown or tan colored and have three to five colored bands around the resistor. These bands tell the value of the resistor (how much resistance the device has), and they start near one end of the resistor and are read band by band going towards the other end. Each color represents a number, and its position on the resistor tells the meaning.

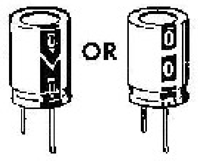
The colors, and their meanings are listed here:

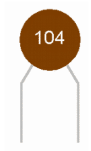
Color	***1st Band***	***2nd Band***	***3rd Band***
Black	N/A	0	X .1
Brown	1	1	X 10
Red	2	2	X 100
Orange	3	3	X 1,000
Yellow	4	4	X 10,000
Green	5	5	X 100,000
Blue	6	6	X 1,000,000
Violet	7	7	X 10,000,000
Gray	8	8	X 100,000,000
White	9	9	X 1000,000,000

The colors for the 4th and 5th bands, if present, represent precision tolerances and are not important for our purposes here.

So, using the above table; if we have a resistor with the first three bands being: yellow, violet and brown, we would take the first color meaning and list that as the first digit of the value (so yellow means 4). The meaning of the next color would be the second digit (violet means 7, so thus far it's 47). Finally we would use the 3rd color to find out by how much to multiply the 47 to find the resistance (brown means that we multiply the 47 by 10 to come up with a total resistance of 470Ω – the Greek letter Omega [Ω] stands for ohms, the unit of measure of resistance.) This is the value of the resistors used in your Experimenter System.

For further study: The American Radio Relay League (ARRL,) which is the association for Amateur Radio enthusiasts, has an article in the March 1984 issue of their QST Magazine with lots of information about resistors. You can get to that article through their web site at: http://www.arrl.org/files/file/Technology/tis/info/pdf/8403011.pdf.

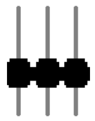




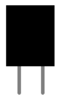
Capacitors – are sometimes shaped, like resistors, as small cylinders (only with both leads usually coming out of the same end of the cylinder), and sometimes shaped like small disks with the leads coming out of the edge of the disk. Occasionally, you will find capacitors that are like a small bead in the middle of two leads. The cylinder shaped capacitors are usually electrolytic capacitors and are, most often, used for power supplies and power circuits. The disk-shaped capacitors are usually used to condition signal lines and are called either disk capacitors or ceramic disk capacitors. The normal markings on a cylindrical electrolytic capacitor are just labels with the value of the capacitor. They are also labeled with a gray or silver stripe or arrow on one side of the cylinder pointing to one of the leads. This gray or silver stripe has negative signs (-) on it and denotes the negative lead of the capacitor. Disk capacitor labels are a little bit more difficult to read. They usually have a three-digit code on one side. The first two digits are simply the first two digits of the capacitance and the third digit is just the number of zeros to add after the first two digits, this number is the number of pico-farads. So a disk capacitor with a code of 104 would be 10 with four zeros after it, or 100000 pico-farads or 0.1 micro-farads. This is the capacitor that we will be using for your Experimenter System

Diodes – are usually small cylinders that look like they are made of ceramic with wire leads coming out of each end. They are labeled with a black, or gray, band circling around near one end. A diode allows electricity to travel in only one direction, going from the lead near the black band through to the lead on the other end. This is used to protect your Experimenter System if you accidentally connect the battery wires backwards.

Light Emitting Diodes (LEDs) – are most often small colored plastic cylinder-shaped devices. They are usually rounded on one end of the cylinder and flat on the other end. There are two wire leads coming out of the flat end and one of those leads is shorter than the other. The side of the cylinder near where the short lead enters into the body of the LED is usually flattened. These two indicators (the short lead and the flattened side of the LED body) show the cathode of the device and this is where the negative point of the circuit is connected. There are many other styles of LED packages, but this is the type that we use on your Experimenter System.



Connectors and Headers – are devices that allow other circuits to be connected to your controller board. There are several types of connectors on your PCB. First are the male headers, as pictured to the right. These black plastic components have silver-colored pins sticking through them. Notice that part of the metal pieces sticking through the plastic body, in these pictures, is shorter below the plastic body than the parts that stick out above. These shorter ends are the ends that will be inserted into the PCB, to be soldered in place. Two of these headers are used to connect optional devices for a more advanced book. The other header allows you to choose to power your Experimenter System from either the programming cable, or the battery pack.



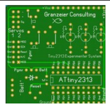
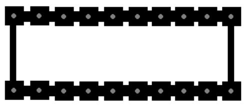
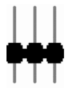
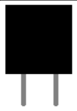
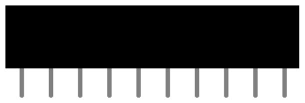
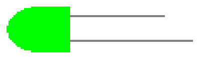
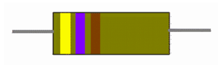
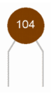
Your Experimenter System also has several female headers. These will allow you to insert wires to connect the various components to the Tiny2313 microcontroller for the experiments. The short metal legs will be soldered into the holes in the PCB, leaving the holes in the top for inserting hookup wires.

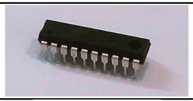
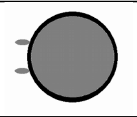
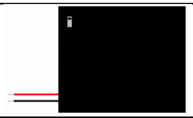
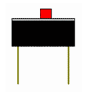
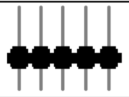
Next is the small green two-pin screw connector. This is used to connect the battery box to provide power to the board. This is located in the lower-left corner of the PCB (as you hold the board with the Tiny2313 chip in the lower-right corner of the board.)

The Parts List

Check each part against the following list. Make a check in the space provided (☐) as you identify each part. The parts may vary slightly from the picture. Any part that is individually packaged with a part number on it should be kept in its package after it is identified until you use it. Save the packaging material until all parts have been located.

To order a replacement part, contact Granzeier Consulting (support@granzeier.com) and indicate the model, value, and the description of the part to be replaced.

√	Quantity and Description	Drawing
	1 – Tiny2313 Experimenter System Printed Circuit Board – This is a piece of fiberglass material with copper traces to connect the components of your development system.	
	1 - 20-pin DIP Socket – This holds the Tiny-2313 microcontroller into the circuit. It protects the chip by allowing you to power up and test the board without the chip installed. Also, since the chip is heat-sensitive, you will solder in the socket and not insert the chip until the solder cools down.	
	3 - 3-pin Male Headers – One of these is used to select the power source for your system. The other two of these headers are for more advanced experiments involving servomotors and sensors. Those topics are beyond the scope of this book, but the headers are included for future study.	
	1 – 1N4001 Diode – This diode will only allow electrical current to flow in one direction, and is used to protect your Experimenter System from reversal of the battery leads.	
	7 - 2-pin Female Headers – These headers are for connecting wires from your controller to the different components on the Experimenter Board.	
	4 - 10-pin Female Headers – Two of these headers are for connecting wires to the I/O pins on your controller, and the other two are to provide convenient connections to power and ground.	
	4 - T-1 Green LED (5 mm) – Note that the shorter lead is the cathode, or the negative lead. The side of the green plastic casing of the LED, nearest the cathode lead, is also flattened out a tiny bit.	
	4 - 470Ω ¼ Watt Resistor (yellow, violet, brown) – There is no polarity on resistors. This means that a resistor can be placed either way in a circuit there is no backwards.	
	1 – 0.1µF Ceramic Disc Capacitor – There is no polarity on this type of capacitor. This means that the capacitor can be placed either way in a circuit; there is no backwards.	

√	Quantity and Description	Drawing
	1 – Shorting Block – This is a small piece of plastic, which has metal inside it to short out two posts on a 2-pin male header. It acts kind of like a switch.	
	1 – Atmel AT-Tiny2313 Microcontroller – There is a small circular indentation in the corner near pin one.	
	1 – Small Speaker – This speaker has two metal tabs coming out of the side, which will be soldered to the PCB.	
	1 – 2-pin Screw Terminal Blocks – These allow you to quickly connect wires to the circuit. These wires will come from the battery case.	
	1 – 4 AA-Cell Battery Case, with Power Switch – This will hold the battery to provide power for your Experimenter System.	
	3 – Miniature Push-Buttons – One of these is the reset button to clear the microcontroller and restart it's program. The other two are for conducting experiments.	
	1 – Double Sided Sticky Foam, 2" X 2" – This will allow you to mount your PCB on to the battery box, for easier transportation.	
	6 – About 12" length of hookup wires – These wires are for you to use to connect the pushbutton switches, LED lights, speaker and other devices to your Experimenter System.	
	1 2X5, 10-pin Male Header – This looks like 2 5-pin male headers joined side-to-side. It is used for the ISP port.	
	1 USBASP Programmer – This is how you will download your program to your ATtiny2313 controller chip.	

Required Tools

There are a few tools, which you will need to assemble your Experimenter System:

1) A diagonal wire-cutter (R/S cat. number: 64-061)
2) A small needle-nosed pliers (R/S cat. number: 55066670)
3) A wire stripper (R/S cat. number: 55066351)
4) A small low-wattage, pencil-style, soldering iron (use 25-30 W.) (R/S cat. number: 64-2070)
5) Solder (R/S cat. number: 55047969) – Note: do not use acid-core solder
6) A jeweler's flathead screwdriver (R/S cat. number: 64-188) – for the green battery terminals (you can also use the screwdriver from a cheap eyeglass repair kit)
7) Safety Glasses/Goggles (Not at Radio Shack, check Walmart, etc.)
8) A Roll of Masking Tape (Not at Radio Shack, check Walmart, etc.)

There are also a few optional tools, which, while not required, can be very helpful:

9) Soldering iron stand (R/S cat. number: 64-2078) – the soldering iron comes with a simple stand, but this is much safer and offers better stability
10) Forceps (R/S cat. number: 64-065) – this is useful for picking up/holding small parts
11) Multimeter (R/S cat. number: 22-182) – this will be useful for troubleshooting
12) Book, *Using Your Meter* (R/S cat. number: 62-116) – this is full of instructions and tips on how to use your multimeter.
13) Solder wick (R/S cat. number: 55048052) – this, or one of the two items below, are needed to unsolder, in case of a mistake
    - or -
14) Solder bulb (R/S cat. number: 64-2086) – while this will work, I have found that the vacuum works much better
    - or -
15) Solder vacuum (R/S cat. number: 64-210)

I have provided the Radio Shack catalog numbers for these items because Radio Shack is the most commonly available electronics store in the U.S. To get any of these items that you do not already have, you can go into your nearest Radio Shack store and ask the sales person to show you where the items are. If they do not carry the items in their store, they can order them for you.

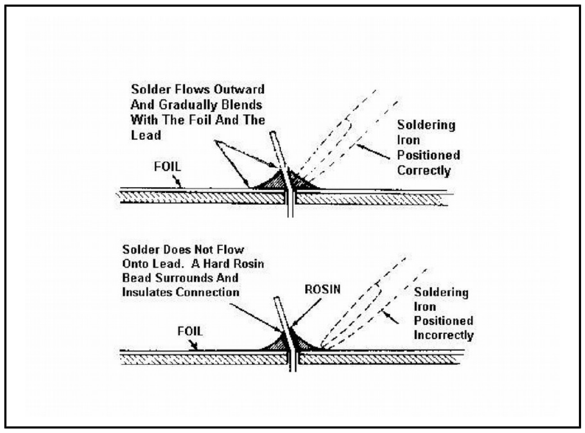
In addition to these tools, you will also need 4 AA-size cells for the battery (you can use either Alkaline non-rechargeable, or either the NiMH or NiCad rechargeable cells.) The system can actually be run off of power from the ISP device; you will only need the battery to run a program without the PC.

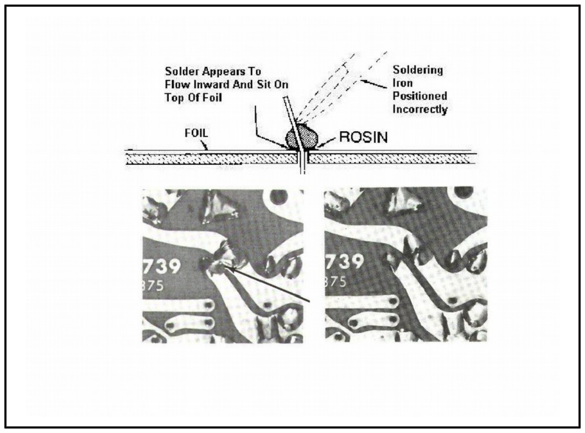
Soldering Introduction

In order to put your new Experimenter System together, you will need to do a little bit of soldering. This will electrically, and mechanically, connect all of the components of your kit to the Printed Circuit Board (PCB.)

Soldering is the process of connecting two or more electrical connectors or wires together and then melting a low-temperature metal, called solder, to hold the connection together. There are two very important things to remember when soldering: first you must have a good mechanical connection, and second you must have a good electrical connection. The first point means that you must make sure that the wires are physically connected before you begin soldering the joint. If you are working on a printed circuit board, the electrical connection of melting the solder will actually provide the mechanical connection.

The easiest way to ensure that you get a good solder joint is to start out by ensuring that the point of the soldering iron is touching both of the conductors that you wish to solder. Heat up both pieces of metal for about 2-3 seconds, and then place the solder against the conductors and let it melt onto both conductors. You will need to be careful that you do not leave the soldering iron on the PCB for too long, this can cause the circuit traces to come loose from the board, and can destroy the board.


Solder Flows Outward And Gradually Blends With The Foil And The Lead
FOIL
Soldering Iron Positioned Correctly
Solder Does Not Flow Onto Lead. A Hard Rosin Bead Surrounds And Insulates Connection
ROSIN
FOIL
Soldering Iron Positioned Incorrectly

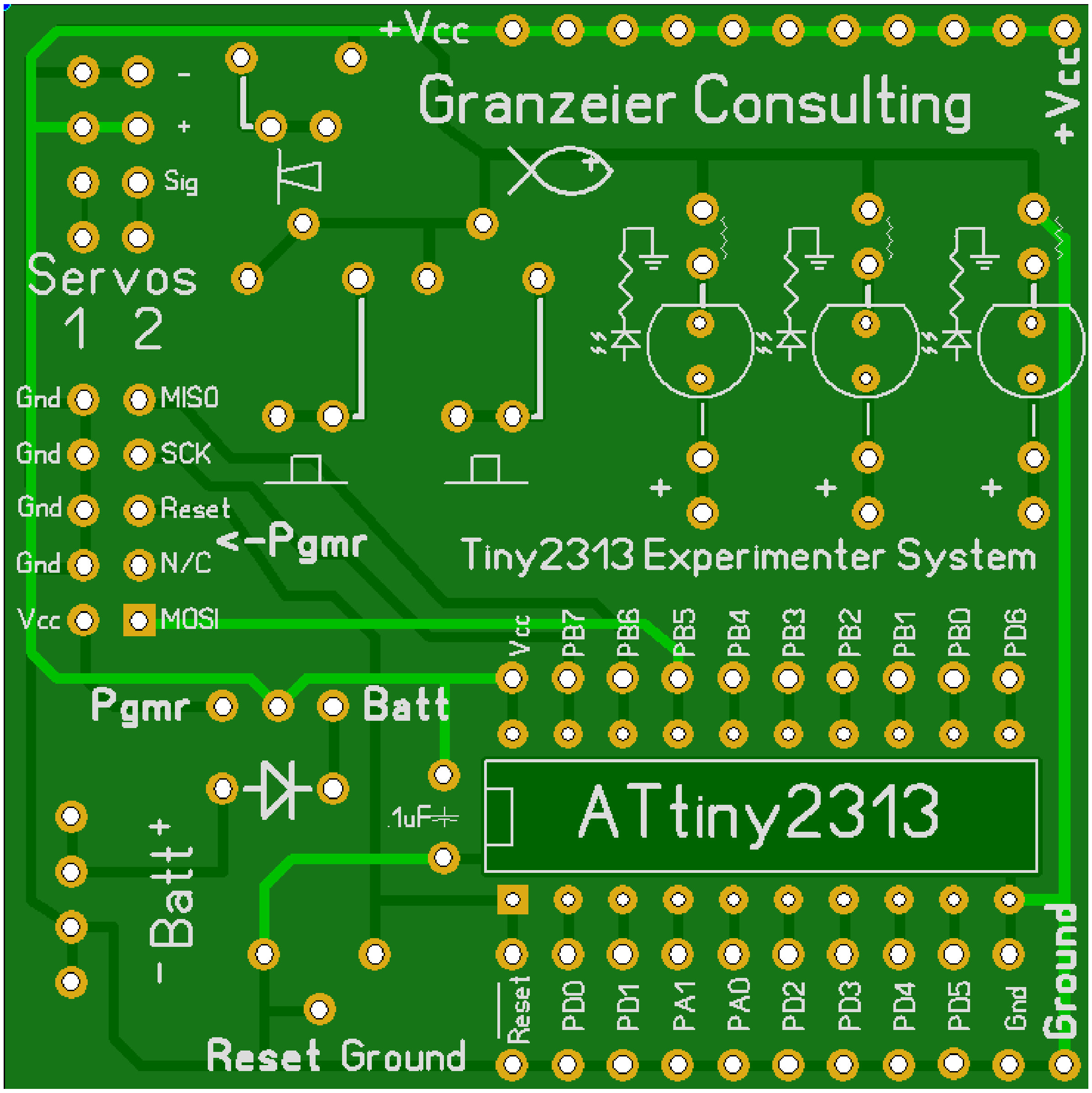

Solder Appears To Flow Inward And Sit On Top Of Foil
Soldering Iron Positioned Incorrectly
FOIL
ROSIN
739
375
739
375

While there are several good web sites that can show you how to get started in soldering; one of the best is NASA's introduction to PCB soldering. You can find it here: *http://radiojove.gsfc.nasa.gov/telescope/soldering.htm*. There are several videos on soldering to a printed circuit board (PCB.) Even though the videos are getting on in years, the information is still pertinent.

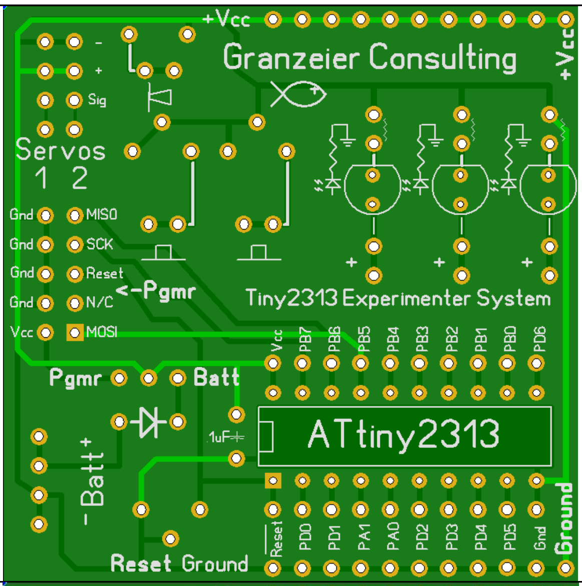
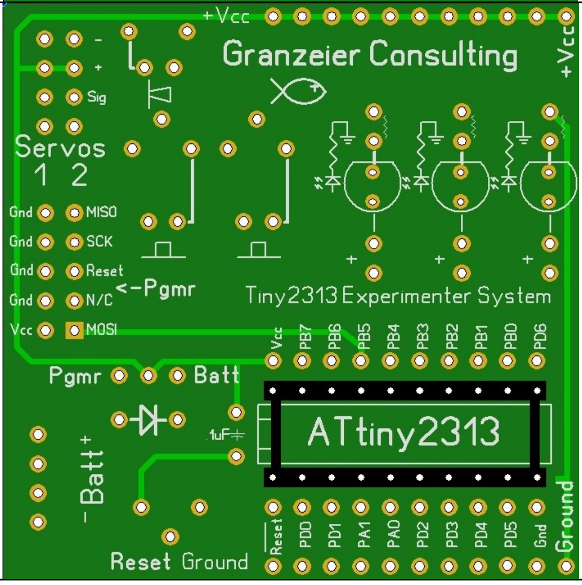
Assembly Instructions

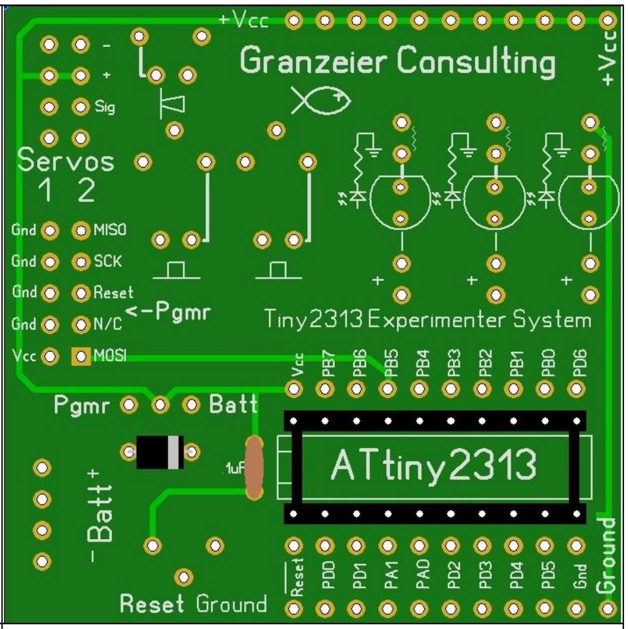
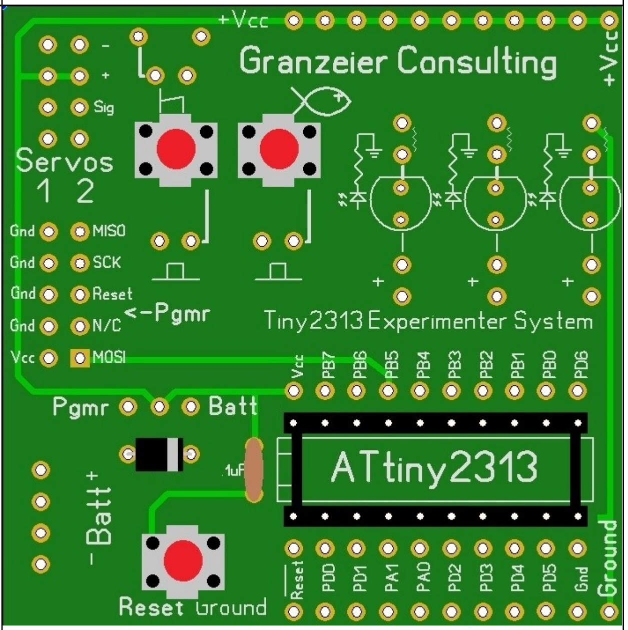
The Controller Board
To build the Tiny2313 Experimenter Board, you will need to make sure that you have the tools that you need. This will include a soldering iron with a fine point, some rosin-core solder, a wire cutter (side- or flush-cut), a small needle-nosed pliers and some masking tape. Refer to the list from page 17.

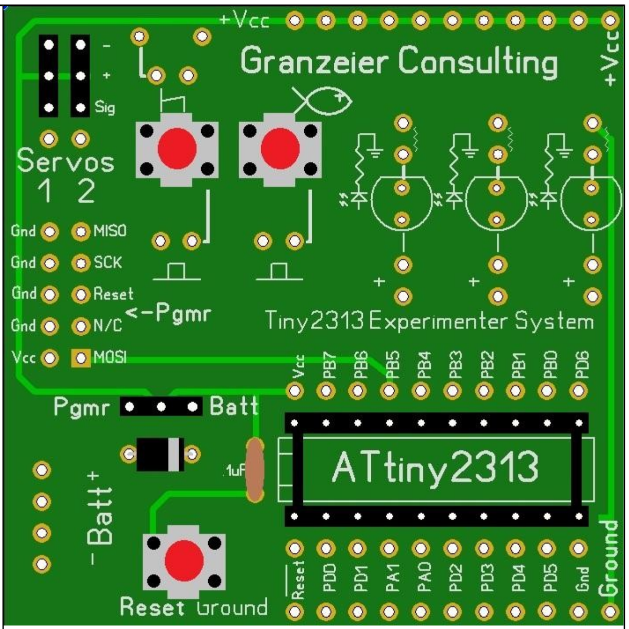
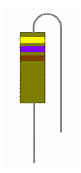
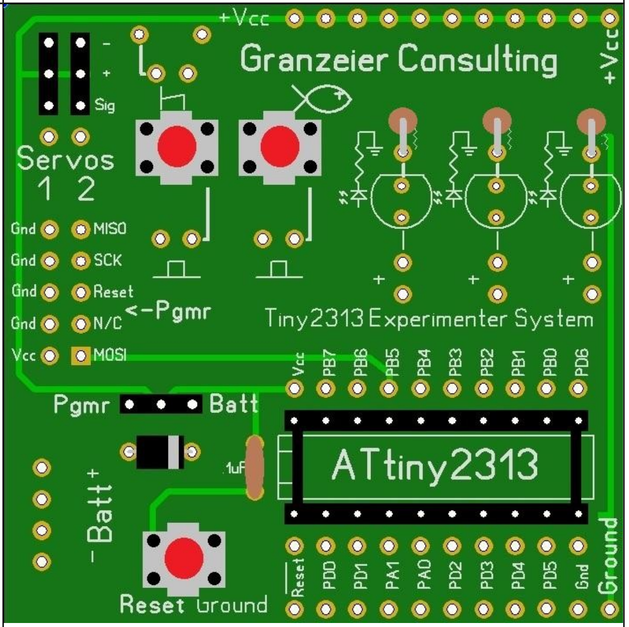
To get ready, take the small components and stick their leads into the black anti-static foam. This will keep the components from rolling around and going missing.

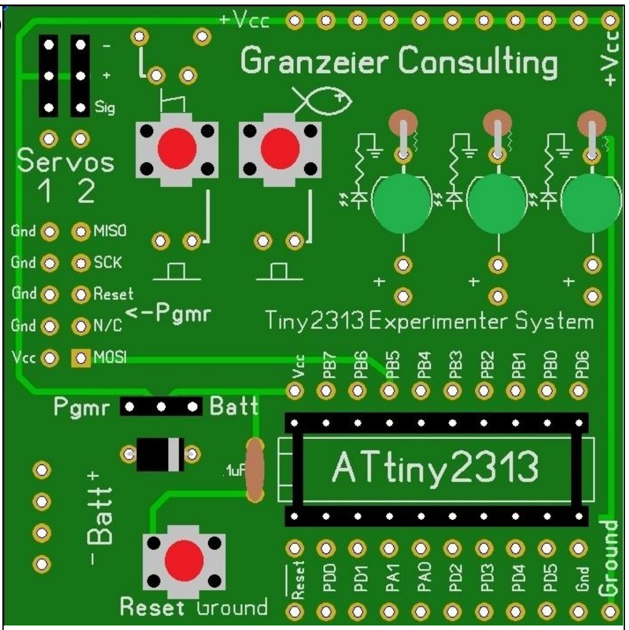
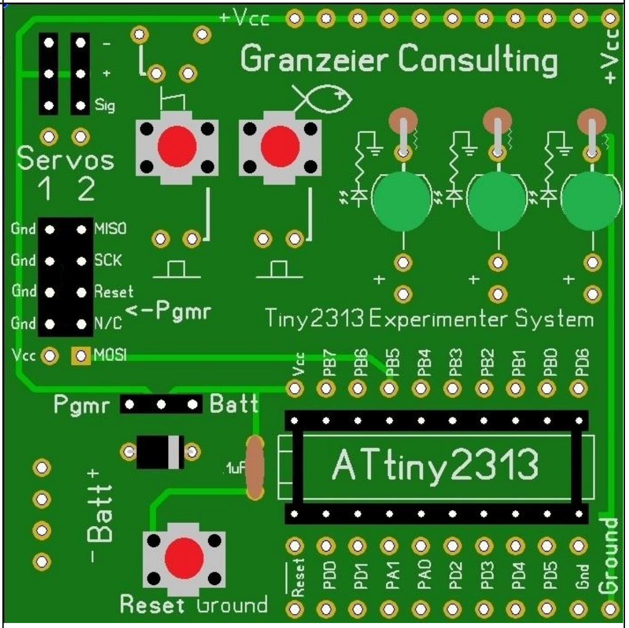


Study this diagram of your Experimenter System board. You may want to go through the instructions step-by-step with this diagram before starting to do any actual soldering.

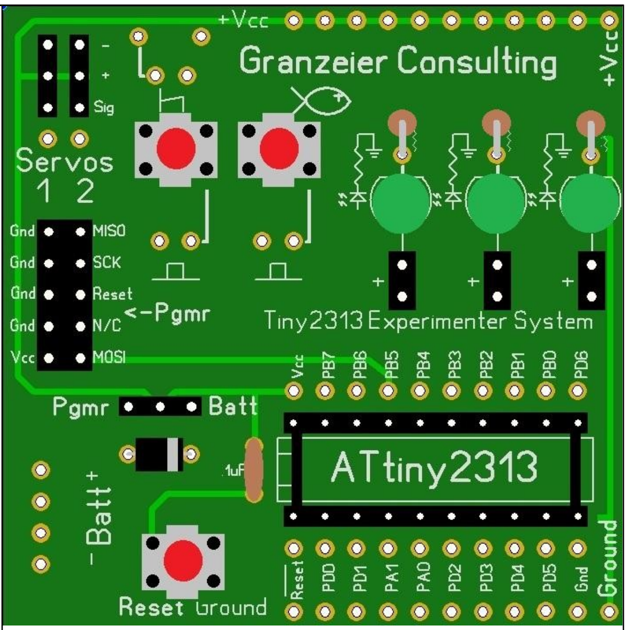
Orient the Printed Circuit Board (PCB) so that the printed side is up and the bottoms of the letters are towards you. Take one strip of masking tape about 1½ - 2 inches (5-6 cm) long and set it aside so that it is ready for use. Take the 20-pin IC socket and place it into the board in the position marked ATtiny2313. Make certain that the notched end of the socket goes to the left and lines up with the notch in the printout on the board. Go slowly and make sure that each pin lines up with and goes through it's corresponding hole in the PCB, and no pins are bent under the socket. When you are certain that you have inserted the socket properly, place a piece of tape over the socket and tape it tightly to the board.	
Turn the PCB over and solder two pins on opposite corners. After you are done, turn the PCB back over and recheck the socket. The entire length of the IC socket should be tight against the PCB. If not, take a piece of paper folded in half several times (to protect your finger from heat,) hold your finger against the paper, pushing the socket tightly against the PCB. While doing this, touch the solder joint on the bottom of the board with the soldering iron. This should loosen the joint and the socket should slide tight against the board. Once the IC socket is tight against the board, solder the rest of the pins to the board. Finally, remove the tape from the IC socket.	

Take the small disc capacitor and insert it into the two holes just to the left of the IC socket. This capacitor is not polarized, which means that there is no forward or backward; so just insert it into the holes and cover it with some more tape. Now, take the diode and bend the leads 90°, so that they are parallel to each other. Insert the diode into the holes with the diode symbol. The diode is polarized, and it does matter which way you insert it. Make sure that the silver band around the body of the diode is in the hole with the bar across the triangle on the schematic symbol; in other words, it should be in the diode hole that is closer to the IC socket. Cover the capacitor and diode with more masking tape, to hold them in place.	 Granzeier Consulting +Vcc Servos 1 2 Sig Gnd MISO Gnd SCK Gnd Reset Gnd N/C <-Pgmr Vcc MOSI Tiny2313 Experimenter System Pgmr Batt ATtiny2313 1uF - Batt + Reset Ground Vcc PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0 PD6 Reset PD0 PD1 PA1 PA0 PD2 PD3 PD4 PD5 Gnd Ground
Again, turn the PCB over and solder the four leads in place. Be careful with the diode – semiconductors (the material inside the diode) are heat sensitive. Do not leave the soldering iron on either leg of the diode for more than about five seconds, or you may destroy the diode. Also, allow several seconds between soldering the two legs for the diode to cool down. Then, use your wire cutters to trim the excess lead on each leg. Turn the PCB back over and remove the tape again.	
Next, place one of the pushbutton switches into the three holes for the reset switch, just to the left of the IC socket. Make sure that the switch is seated, all the way, against the circuit board. Cover that with tape to hold it into place. Next insert the other two pushbuttons into the positions for the other switches. Cover those with tape, and then turn the PCB over and solder all nine legs. After the pushbutton switches are mounted, clip off the excess lead from each leg of each switch with your wire cutter. Again, remove the tape from the pushbutton switches.	 Granzeier Consulting +Vcc Servos 1 2 Sig Gnd MISO Gnd SCK Gnd Reset Gnd N/C <-Pgmr Vcc MOSI Tiny2313 Experimenter System Pgmr Batt ATtiny2313 1uF - Batt + Reset Ground Vcc PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0 PD6 Reset PD0 PD1 PA1 PA0 PD2 PD3 PD4 PD5 Gnd Ground

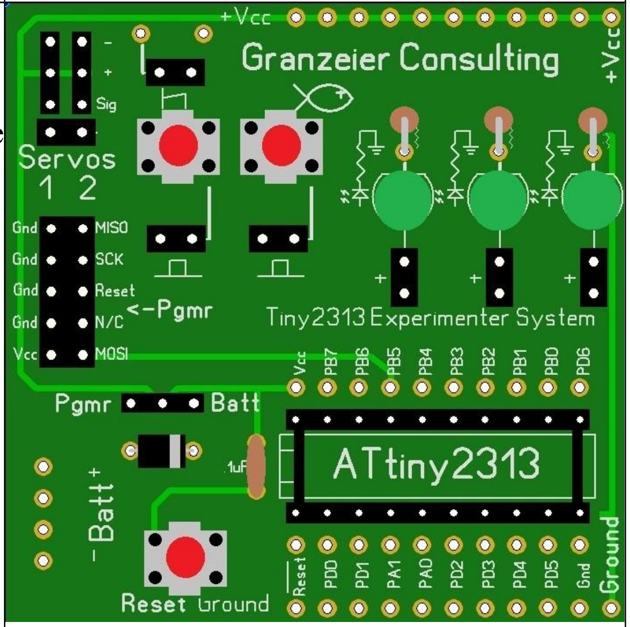
Next, we will be adding the three 3-pin male headers. Notice that these headers have a piece of plastic with three metal leads going through the plastic. The plastic is not in the center, but is offset so that there are three longer leads and three shorter leads on the opposite side of the plastic. Insert these headers into the PCB with the short leads into the board, and the longer leads sticking up from the board. One will go next to the diode that you inserted earlier, and two will go in the upper-left corner of the board. Cover these with tape to hold them in place and solder one pin of each on the bottom. Be careful to make sure that they are tight to the board, just like you did with the IC socket. Once the headers are tight, you can go ahead and solder the rest of the legs.	 +Vcc Granzeier Consulting Servos 1 2 Sig Gnd MISO SCK Reset N/C MOSI Vcc <-Pgmr Tiny2313 Experimenter System PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0 PD6 Pgmr Batt 1uF ATtiny2313 - Batt + Reset Ground PD0 PD1 PA1 PA0 PD2 PD3 PD4 PD5 Gnd
Now, for the LED/Resistor circuits: Take three of the resistors and bend one lead, of each, back along the body of the resistor so that it is parallel to the other leg (as in the picture to the right.)  Insert each resistor, so that the body of the resistor is closer to the name Granzeier Consulting on the PCB and the bent-over leg is closer to the LED symbol on the PCB. Tape the resistors into place and solder the bottom. Again, trim the excess lead from each leg, and then remove the tape. Notice that there is no polarity to resistors, and so you do not need to worry about placing them backwards.	 +Vcc Granzeier Consulting Servos 1 2 Sig Gnd MISO SCK Reset N/C MOSI Vcc <-Pgmr Tiny2313 Experimenter System PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0 PD6 Pgmr Batt 1uF ATtiny2313 - Batt + Reset Ground PD0 PD1 PA1 PA0 PD2 PD3 PD4 PD5 Gnd

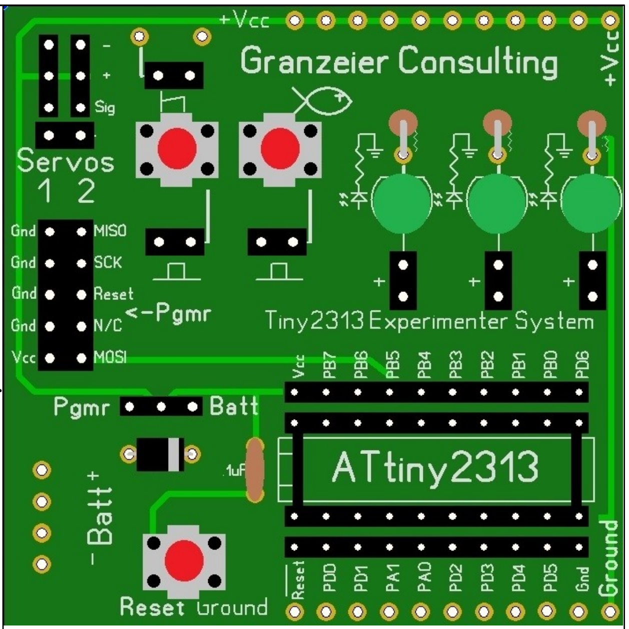
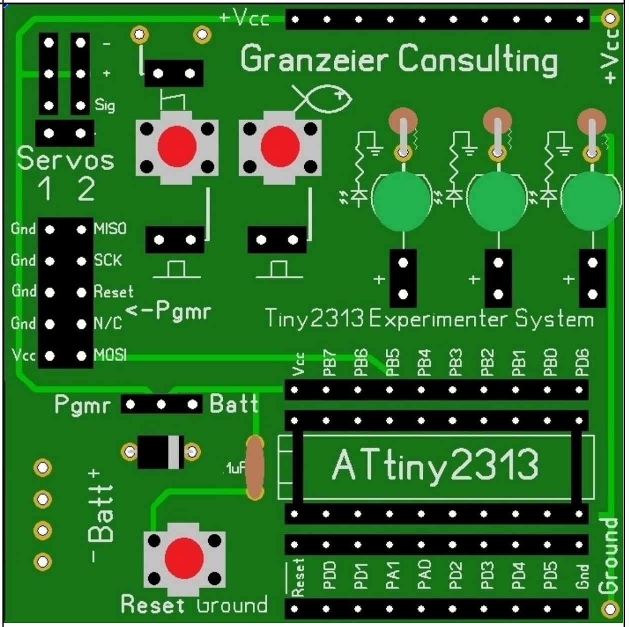
Next, place the three LEDs into the holes with the LED schematic symbols. The LEDs are polarized, so you will need to make sure that you insert them in the proper direction: the longer leg is the anode, or the positive leg. The anode must go into the hole closer to the IC socket. Also, if you look closely, you will notice that the side of the LED's body, closer to the shorter leg, is slightly flattened. That side of the LED should go into the hole where the circle of the schematic symbol is flattened – closer to the resistors. Again, cover with tape, solder and trim the excess leads. Also, LEDs are semiconductors, me careful not to overheat them; don't keep the iron tip on the lead for more than about five seconds.	
The next component is the 10-pin (2X5) male header for the in-system programming (ISP) port. This header is similar to the 3-pin male headers that you already soldered in to the PCB. You will place the header into the board (short ends of the legs into holes) on the left side of the board. As you did with the IC socket, just solder two opposite pins first and then make sure that the entire header is tight against the board. If not, use the same procedure of holding the header tight (don't forget something to protect your finger from heat) and touch the solder connection on the back. Once the header is tight against the PCB, go ahead and solder the rest of the pins.	

Next, we will be inserting the 2-pin female headers. There are seven of these, and so you will want to break this step down. Take three of the female headers and insert them into the board in the three spaces below each of the LEDs. Use the tape to hold them in place and solder one leg of each header. Check that the headers are tight against the board, as you did with the IC socket. If necessary, tighten the headers to the board and then solder the remaining leg for each header, and remove the tape.

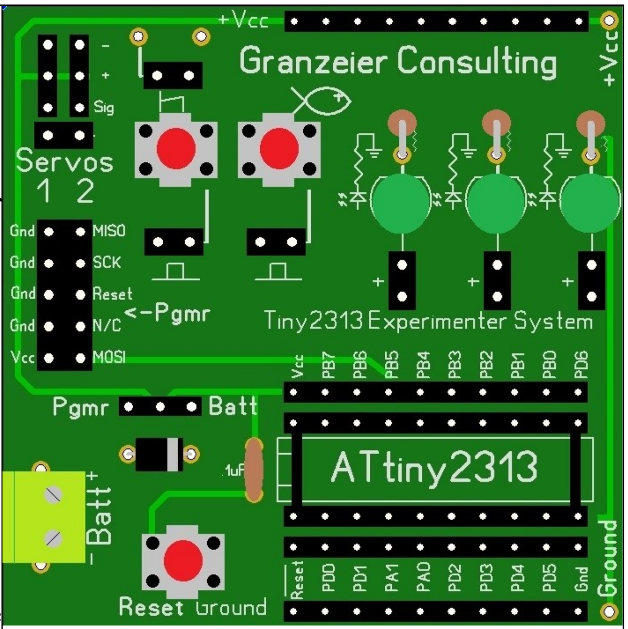


Take two more 2-pin female headers and insert them into the spaces just below the two pushbuttons. Tape these down and then solder the headers into the board following the procedure that you used for the first three 2-pin headers. Once those are soldered in place, insert the last two into the spaces below the 3-pin male headers and below the speaker schematic symbol on the PCB. Solder these down in the same way that you soldered the other 2-pin female headers. Remove the tape when you are done soldering.



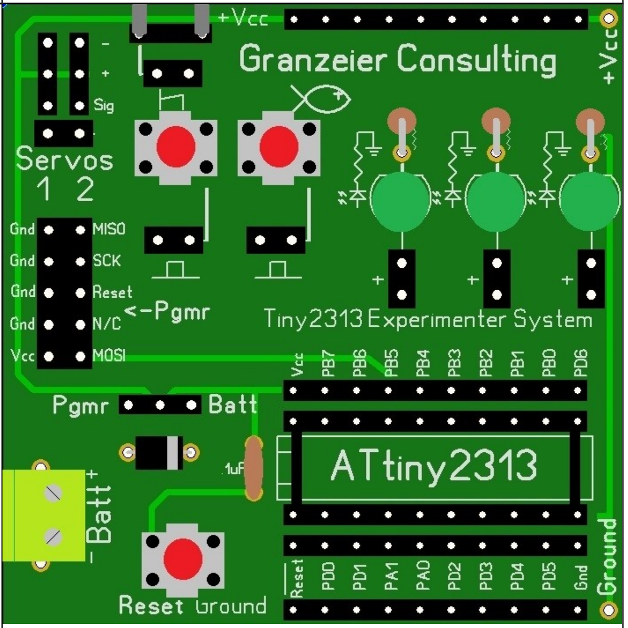
Next, we will be installing the 10-pin female headers. Two of these are for connecting the input and output (I/O pins of the Tiny2313 to the other components, and two are to provide a convenient place to access the +V and ground power terminals. We will start with the I/O headers; take two of the female headers and insert them alongside the IC socket, and tape them down. You will want to start with only the two end pins when you solder. Then you may check that the headers are tight against the board, as with the IC socket. Once the headers are tight against the board, go ahead and solder the rest of the pins, on each header.	 +Vcc Granzeier Consulting Servos 1 2 Sig Gnd MISO Gnd SCK Gnd Reset Gnd N/C Vcc MOSI <-Pgmr Tiny2313 Experimenter System Vcc PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0 PD6 Pgmr Batt .1uF ATtiny2313 - Batt + Reset PD0 PD1 PA1 PA0 PD2 PD3 PD4 PD5 Gnd Reset Ground Ground
After the I/O headers, you will want to insert the +V and ground power headers. These are the 10-pin female headers along the top and bottom edge of the PCB. These will go in almost the same as the I/O headers. The only change is that there are eleven holes for the 10 pins of the header. The right-most hole is an extra pin to be used to expand the Experimenter System. For now leave those holes empty and insert the headers flush to the left. Again, solder the two end pins, and make sure that the headers are flush before soldering the rest of the pins.	 +Vcc Granzeier Consulting Servos 1 2 Sig Gnd MISO Gnd SCK Gnd Reset Gnd N/C Vcc MOSI <-Pgmr Tiny2313 Experimenter System Vcc PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0 PD6 Pgmr Batt .1uF ATtiny2313 - Batt + Reset PD0 PD1 PA1 PA0 PD2 PD3 PD4 PD5 Gnd Reset Ground Ground

Next, take the small green screw terminal and insert the pins into the holes on the left side of the PCB. You will notice that there are four holes. That is because the screw terminal comes in two different sizes. You will just insert the pins into the holes with one pin in either of the top two holes and the other pin in either of the bottom two holes. Make sure that you do not have both pins in the top two or bottom two holes. The other thing that you will need to pay attention to is that one side of the terminal has a small ledge. This is the side from which the wires will be inserted. This side will need to be closer to the edge of the PCB. Double check the orientation by unscrewing one of the terminals – a small hole should open on the side of the plastic block – this is where the battery wires will be inserted. After you have gotten it aligned correctly, tape it down and solder the bottom – make sure that the terminal block is tight against the PCB.



Now, for soldering the final component. Take the small speaker and insert the legs into the PCB with the body of the speaker sticking out over the edge of the board. The silver face of the speaker should be facing up. Tape the speaker down and solder it in, just as you did with the 3-pin male headers.

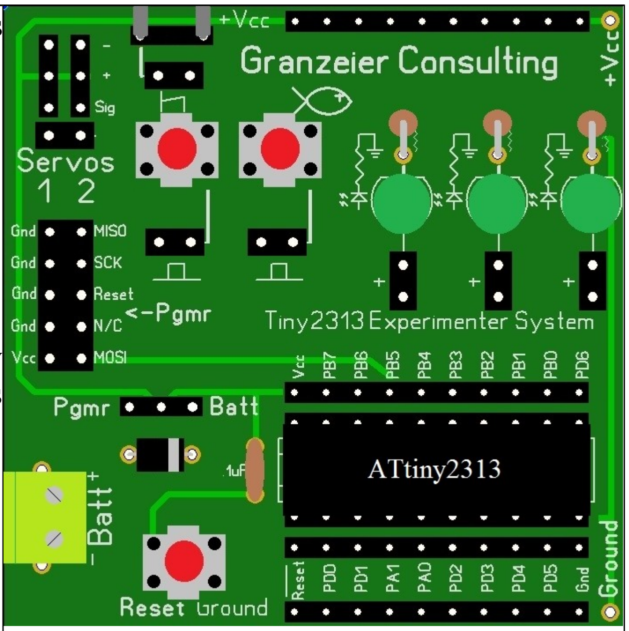
Take the shorting block and slide it down over the two pins of the Pgmr/Batt male header closest to the Pgmr side. There is a small piece of metal inside the shorting block. This acts as a switch, connecting the center pin (which is the positive point of the Tiny2313,) to either the positive pin of the programming cable, or the positive lead from the battery box. This allows you to select which will power your Tiny2313 Experimenter System.



Now, take the 2313 chip itself. Take the chip out of it's protective foam. You will notice that this foam is not ordinary plastic foam. This is called anti-static foam and it's purpose is to protect sensitive electronic devices from static electricity. (*See important note below.*)

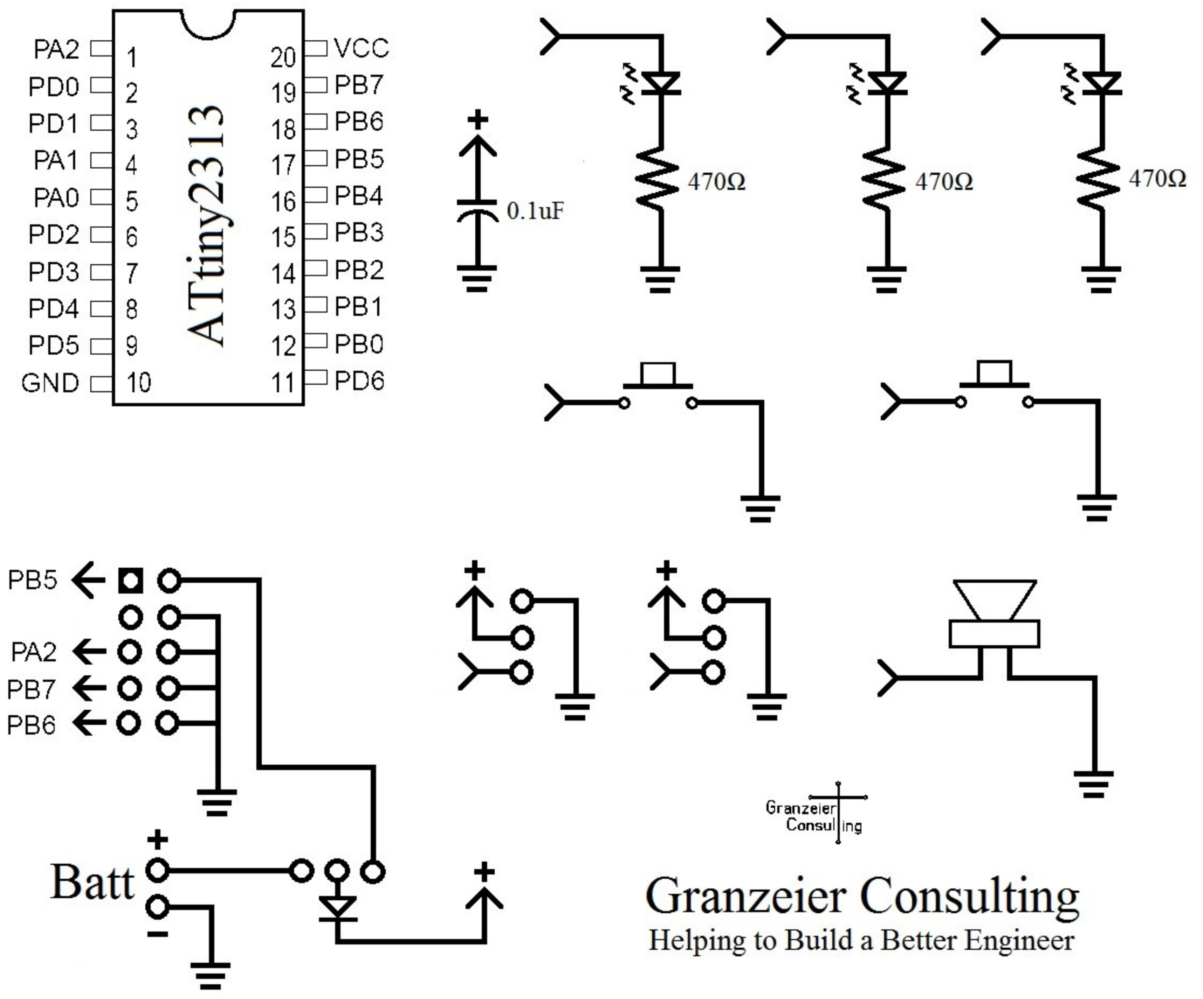
Most of the time, the pins on DIP chips are bent a little out and may require that you gently bend the two rows of pins back towards each other. Place the pins slightly into the socket and then carefully push the chip into it's socket.

Finally, take the red wire from the battery pack and, after making sure that the switch is turned off, insert it into the positive-side screw terminal and gently tighten the screw down (if the screw is already tightened, just unscrew it enough to insert the wire.) Do the same thing for the black wire into the negative terminal. Insert 4 AA-sized cells into the battery box – make sure that you follow the markings for correct orientation.



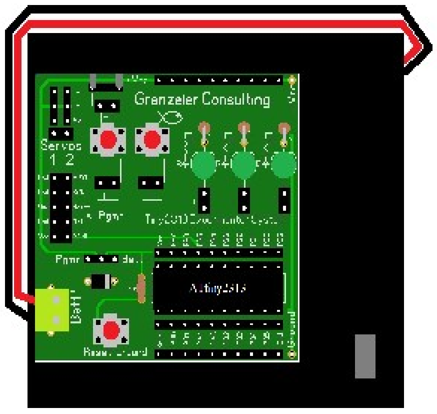
**Note* - Do you remember ever walking across a carpeted room, touching a metal object (maybe a door knob) and getting shocked as your finger nears the metal? This static electricity discharge only makes you jump a bit, but it can shock the bejeebers out of (destroy) your ATtiny2313A chip. Use a bit of caution here. Touch something grounded, like the metal case of a grounded (three-prong power plug) appliance, such as your refrigerator, before you touch the IC. This will ground you and eliminate any static charge on your body. Also, try to handle the printed circuit board only on it's edges like you would a CD.

Your Tiny-2313 Experimenter System is pretty flexible about it's input voltage. You may use standard carbon/zinc or Alkaline AA cells, or you may use rechargeable, Nickel-Cadmium (NiCd, or NiCad) or Nickel Metal-Hydride (NiMH) cells. The standard cells put out 1.5 volts, while the rechargeable cells only put out 1.2 volts. With the standard cells, your battery pack puts out 6.0 volts and with the diode in the circuit, your system runs on +5.4 volts (the diode drops 0.6 volts.) If you put rechargeable cells in the battery pack, the battery will put out 4.8volts and your system will run on 4.2 volts (again, due to the diode's voltage drop.) Both of these system voltages are within the limits of the AVR-ATtiny2313A chip. You could even connect up a 5+ V DC power supply, like the charger for your phone – just be careful that the power supply puts out direct current (DC) and that it does not exceed +6V. Also, make sure that you connect the positive lead from the power supply to the + screw terminal and the negative lead to the - screw terminal.



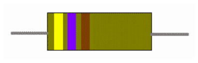
This is the schematic diagram for the Tiny2313 Experimenter System.

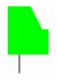
The last step is optional, included with your kit is a 2" X 2" piece of double-sided, foam tape. Peel the plastic paper off of one side and carefully stick it onto the top (the side with the power switch) of the battery pack. You will want to make sure that you keep it away from the power switch. Now, peel the plastic paper off of the other side of the foam tape and carefully stick the Tiny2313 PCB onto the tape. Note that you will want to twist the board around (see the picture) so that the wires are neatly against the side of the battery box. You may even want to glue the wires to the side of the battery box – if you do that, make sure that you take the cover off the bottom of the box, so that you do not glue it closed. The cover needs to be able to slide opened and closed freely so that you may change the battery.

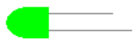


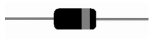
Quiz 3

1) What does a resistor look like? _____

   A) 

   B) 

   C) 

   D) 

2) What input voltage will work best for your Tiny2313 Experimenter System? _____
   - A) 1.0V DC
   - B) 1.5V DC
   - C) 5.0V DC
   - D) 9.0V DC

3) The diode on your Experimenter System is heat sensitive, and care must be taken to not overheat the device while soldering it to the circuit board. (true or false) _____

4) For protection from static electricity, the '2313 chip is stored in _____
   - A) Styrofoam
   - B) A plastic bag
   - C) Shaving foam
   - D) Anti-static foam

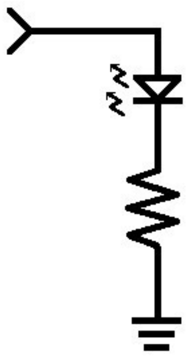
5) What are the two very important things to remember when soldering? _____ & _____
   - A) Heat up, Cool down
   - B) Mechanical connection, Electrical connection
   - C) Touch the conductors, Not too long
   - D) Component leads, PCB pads

Extra Credit:

Briefly describe the purpose and function of a printed circuit board (PCB) _____

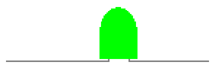
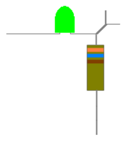
Your First Program

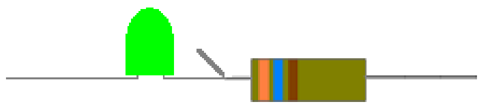
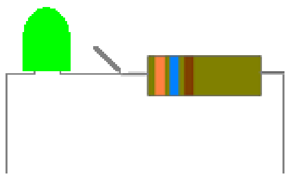
Most programmers, when they are learning a new programming language, will start with a program called the "*Hello World*" program. This is nearly the simplest program that may be written in a language for any given computer. All it does it to output the phrase: *Hello World* onto the output device. It's only purpose is to provide a very quick success and show the new programmer that the language does indeed work. In the controller world, it turns out that there are far simpler things to do than output some text. In fact, sometimes it can be downright complex to output text, so we do something much simpler. The *Hello World* equivalent in the microcontroller world is simply lighting up an LED. A Light Emitting Diode, or LED, is an electronic device that gives off light when electricity goes in one direction through it.



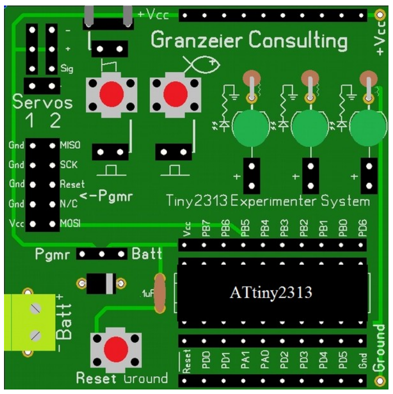
For this *Hello World* experiment, first study the schematic diagram to the right. A schematic diagram is simply a picture representing an electronic circuit. At the top of the schematic, we have an arrow pointing to the right. This symbol represents an input (to the circuit) from the Tiny2313's output pin. From there, we have a line going to the right a bit and then turning down. This simply represents a wire connecting that input to the next component. That component looks like a triangle with a bar across the bottom point, and a couple of small arrows to the left. This symbol represents a light emitting diode (LED); this device, as we mentioned above, gives off light when current flows through it from the bottom (with the bar) to the top. There is another wire from the bottom of the LED to the top of a zig-zag symbol. This symbol represents a resistor; the resistor protects the LED and the Tiny2313 from too much current. Finally, there is another wire going to the symbol of 3 horizontal lines; this represents *ground*, or the negative terminal of the battery.

Prepare your LED/Resistor pair for your first experiment.

Take the additional LED and resistor, which were included in your kit.	
Carefully bend both leads of the LED out 90° as in the picture.	
Cross the shorter lead with the resistor lead and twist them together like you see in the next picture.	

Carefully, solder the two leads together, trim off the excess and then, after the solder cools, bend the soldered twist back against the LED's lead like in the next picture.	
Finally, bend the longer lead from the LED and the resistor lead so that they are about parallel with each other (like the picture to the right.) This will give you a simple LED/Resistor pair for your experiments. Keep in mind that the lead with the resistor is the cathode, or the negative lead.	

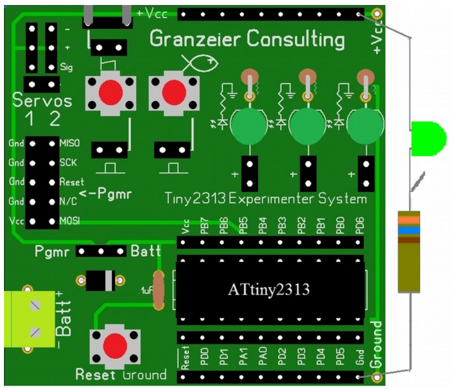
This gives you a simple piece of test equipment so that you can perform initial testing of the Tiny2313 system, and to verify that LEDs work the way we have described.



Make sure that the USB programmer is NOT plugged into your Experimenter System, and that the battery box is turned off. Place the tiny shorting jumper over the center pin and the Batt pin of the power select pins, just above the diode. Notice, that along the top and bottom of your Tiny2313 PCB, there are two 10-pin female headers. These are the +Vcc (positive voltage) and the Ground (or 0 volts) access busses. All of the socket holes in these headers are connected together and to +V or Ground. Their only purpose is to provide convenient access points for ground and plus voltage for your circuits. These are the same as the positive and negative leads from the battery pack.

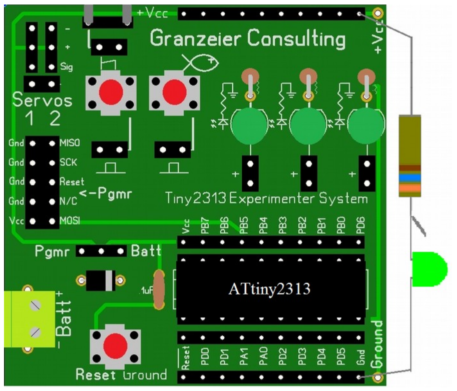
> ***Important Note:***
> *Changing the wiring to your Experimenter System while power is applied could damage, or even destroy, your Tiny2313 chip.*

Get your LED/Resistor pair. Note that the resistor is soldered to the shorter lead of the LED. This



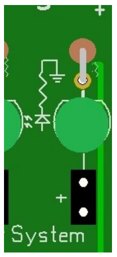
marks the cathode, or negative, lead on the LED. This lead will need to be connected to the ground (or 0 volts) connector on the Tiny2313 Experimenter System. The other lead of this LED/resistor pair will go to one of the +Vcc header pins on your Tiny2313 system. Take the free lead from the resistor and gently push it into the female header socket along the bottom of your Tiny2313 in the right-most hole (or pin.) Now, gently push the other lead from the LED into the right-most pin on the header across the top, labeled +Vcc. This pin is connected to the positive voltage on your Tiny2313 Experimenter System. Take a look at the diagram to the left for the locations where each pin is located on your board. If your LED/resistor pair is a bit too short, you may solder an additional couple of inches of the included jumper wire to the lead of either the resistor, or the LED.

When you have the leads from your LED/resistor pair firmly plugged into the Ground and +Vcc header sockets, turn the battery switch to the on position. You will notice that the LED immediately begins glowing. This shows two things: 1) that your Tiny2313 system is working, at least as far as the power goes; and 2) that when you connect the cathode of an LED through a resistor to ground, and the anode of the LED to a positive power source, the LED will emit light. Congratulations, you have verified that an LED works the way it should.



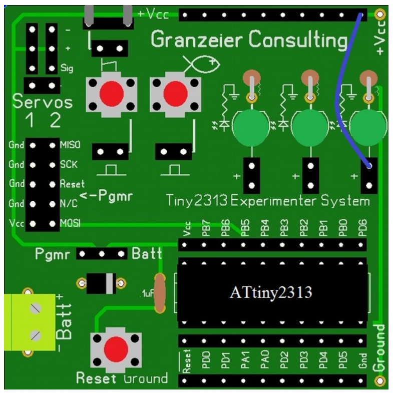
Turn the battery switch back to the off position.

Now, gently remove the LED/resistor pair and swap the leads. Insert the leads back into the power headers, only this time, place the resistor lead into one of the +Vcc header pins, and the LED lead into one of the Ground header pins. Turn the battery pack back on, and notice that the LED does not glow this time. Go ahead and turn the battery pack back off.



This shows that LEDs work by allowing current to flow in only one direction, and that only when current is flowing, they will give off (or emit) light. Once again, go ahead and remove the LED/resistor pair and set it aside. You will use your LED/resistor pair a bit later in this book.

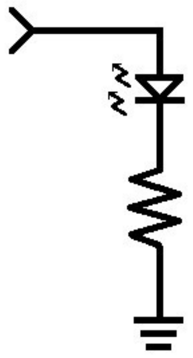
After this experiment, take another look at the white schematic drawing printed on the PCB, right next to each LED. This looks almost exactly like the schematic drawing that we saw a couple pages back (it's just upside down.) The anode of the LED is connected directly to the 2-pin header (notice the white line below the LED, going to the header.) All you will need to do to complete the circuit is to plug a hookup wire from the header to whatever you would like connected to your LED. In fact, that is the idea behind *development systems* like your Tiny2313 Experimenter System, most of each circuit is already present, all that you need to do is connect the parts together – simple, huh? Just like the dot-to-dot pictures when you were a little one.



You will need to cut one short (2"-3") piece off of one of your hookup wires, and strip about ¼" of the plastic insulation off each of the ends of that short piece of hookup wire. It is standard convention to use black wires only for ground, and red wires only for +V on a circuit. It would be best to use another color to hook up your Tiny2313 and the LED/resistor pair. This wire will need to be connected between the Tiny2313 +Vcc connector, and the LED connector. Take the hookup wire and gently push one end into the +Vcc female header socket along the top side of your PCB, where you had the LED/resistor pair connected. Now, gently push the other end of the hookup wire into one of the holes in the 2-pin female header socket below the right-most LED. This pin is connected to the positive pin of the LED, through the resistor and to the ground of your Experimenter System.

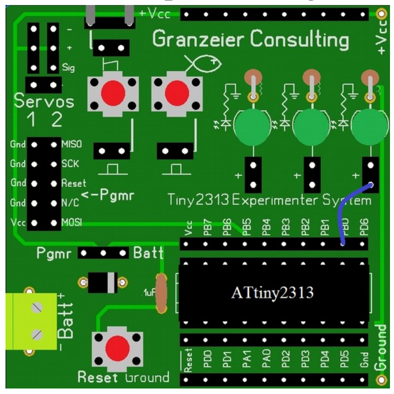
Once again, turn on the battery pack, you will notice that, just like your LED/resistor pair, the LED glows. This is because you duplicated the circuit that you had created with the outside LED/resistor pair, using the LED, resistor and wiring on-board your Tiny2313 system board. Electrically, these circuits are exactly the same.

Go ahead and move the jumper wire from the right LED, to the center LED (you did make sure to turn off the battery first, right?) and test that LED. When you turn on the battery box, the center LED will light. Turn the battery box off and move the jumper wire to the left LED. Test that circuit by turning the battery back on for a short time – the LED should light again. If one, or more of, your LEDs do not light, recheck your solder connections. If you missed soldering one of the connections on either the LEDs, the resistors or the female header, the LED will not light. Also check for the LED orientation – the small flat spot, on the side of the LED, should be towards the resistor. If one of the LEDs is inserted backwards, you will need to carefully unsolder the leads and remove the LED. Re-insert it properly and solder the leads down again.



Take another look the schematic diagram on the right. Notice that, following this diagram, we are going to take an output from our Tiny2313 and connect it to an LED, which is then connected through a resistor, and to ground. When we have the Tiny2313 apply positive voltage on this pin, the current will flow from ground, up through the resistor and then through the LED by being attracted to the positive charge on the output pin (remember that the only things moving in an electrical circuit are electrons, and with the electron's negative (-) charge, they will only move towards a more positive (+) charge – opposites attract.) When the current flows through the LED, it will give off light, saying "*Hello World*". Likewise, when we have the controller apply ground to the output pin, the electrons will not be attracted to any positive charge and will not flow from ground. Without any current flow, the LED will not light up. We now have the basis for turning a LED on and/or off under control of the Tiny2313.

For this experiment, you will need your Experimenter System board and the hookup wires. Again, make sure that the USB cord is NOT plugged into your Experimenter System, and that the battery box is turned off – remember: changing the wiring to your Experimenter System while power is applied could damage, or even destroy, your Tiny2313 chip. You will also need that short piece of hookup wire. This time the wire will need to be connected between the Tiny2313 I/O pin connector, and the LED connector. Take the hookup wire and remove the end that was plugged into the +Vcc header and gently push that end into the female header socket along the top side of your ATtiny2313 chip, in the second from the right-most hole (or pin), labeled PB0. Take a look at Appendix B in the back of this book for a diagram showing you where each pin is located on your '2313 chip. This single piece of short hookup wire, combined with the components on your Experimenter System PCB, completes the entire circuit shown in the above schematic diagram.


Granzeier Consulting
+Vcc
Servos
1 2
Sig
MISO
SCK
Reset
N/C
MOSI
<-Pgmr
Tiny2313 Experimenter System
Vcc PB7 PB6 PB5 PB4 PB3 PB2 PB1 PB0 PD6
Pgmr
Batt
ATtiny2313
Reset PD0 PD1 PA1 PA0 PD2 PD3 PD4 PD5 Gnd
Reset Ground
Ground

Now that we have the components wired up, we will work on the program. Make sure that the shorting block, for the power select jumper is connected to the two left-most pins, this will make your Experimenter System obtain it's power from the ISP plug. Plug the USB programmer in to the USB jack on your computer and the cable end into the 10-pin male header on your Tiny2313 board.

The cable should be plugged into the male programming port with the colored edge of the ribbon cable towards the two three-pin male headers near the top of the board. Launch BASCOM, by double-clicking on it's icon. When the program comes up type the following lines into the editing (largest) pane of your BASCOM screen (change my name in the Author line, to yours):

```
' Title:
' Author: Art Granzeier, Granzeier Consulting
' Date:
' Description:

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

' Main Program

End
```

This is called a template; it is simply a skeleton, upon which we can build our real program. We will start each new program with this template, and then we don't have to worry about the basic "housekeeping." Save your template by clicking on File and then clicking on Save as. In the Save as dialog box, give it the name "template". From now on, whenever you want to create a new program for your Tiny2313 Experimenter System, you will start by loading the template and then changing the name. If you ever, accidentally, delete or overwrite your template, you may recreate it, just by following the above steps again.

A computer, like the Tiny2313A will normally, execute a program starting at the top, and working it's way down through the program. This is called "top down" program execution. It's really pretty easy, it's just like when you follow someone's written directions; you start at the top, and work you way through the directions, just like normal reading.

Let's take a closer look at what is going on here. Notice that the first four lines all begin with the apostrophe (single quote, or '), this tells the BASCOM compiler that everything on that line is a comment, and to ignore it. Comments are added to programs so that you, the programmer, can tell what the program is doing. At the beginning, we are just giving identifying information about the program. In fact, this is called the *identification section* of the program.

The next section, which begins with another comment line, contains what are called directives. These lines, which each start with the dollar sign ($) are directing the BASCOM compiler how to set itself up. As the comment line indicates, this is the beginning of the *configuration section*. These first five lines are really beyond the scope of this introduction, but if you look at the lines you just entered, you can already tell a little bit about these directives. The first directive, sort of contains the name of the microcontroller on your Tiny2313 Experimenter System. Since BASCOM can convert your BASIC program for many different microcontrollers in Atmel's AVR family, it needs to know exactly which chip to target; this command directs BASCOM to convert your program to the ATtiny2313. For now, you can just keep the directives lines of the configuration section as they are above.

The final line in our template is the `End` statement. Every program must end with an `End` statement to prevent the program from starting over by built-in resets.

Now that we have this framework, the template, set up, we can get on with writing our own programs – and, with the framework done, it will be a pretty simple job. With the template (as shown above – remember, use your name as the author, not mine,) in your editing screen, start a *new* program by clicking on the File menu and then clicking on the Save As... entry. When the Save As dialog box appears, give the program a name by typing **`LED Test 01`** in the file name field, and then click the Save button at the bottom of the dialog box. Notice, at the top of the BASCOM window – in the blue title bar, that BASCOM is showing the name of your new program. You are now ready to start entering your new program.

Begin by updating the identification section of the program like this:

```
' Title: LED Test 01
' Author: Art Granzeier, Granzeier Consulting (again, use your name)
' Date: 13 Oct 13 (use today's date)
' Description: Turning an LED on
```

Next, go to the bottom of the configuration section (right below the `$framesize` directive) and add the following line:

```
Config PortB = Output
```

And after the Main Program comment line, add the line:

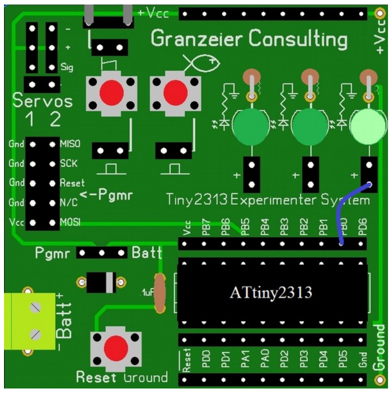
```
      set PortB.0
```

So, your whole program should look like this:

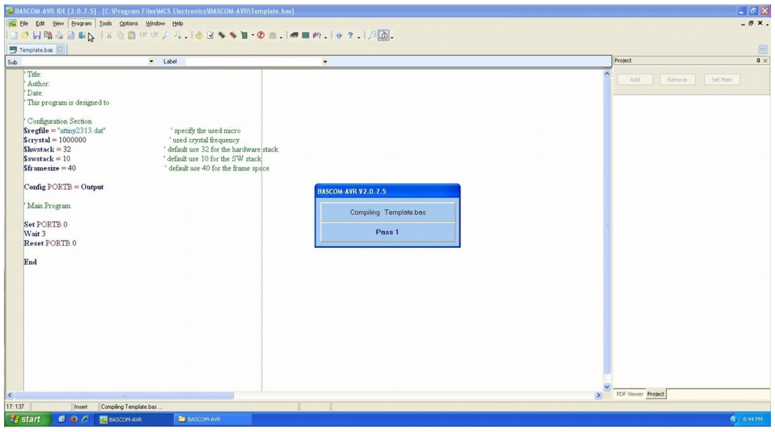
```
' Title: LED Test
' Author: Art Granzeier, Granzeier Consulting (again, use your name)
' Date: 13 Oct 13 (again, use today's date)
' Description: Turning an LED on

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space
```

```
Config PortB = Output

' Main Program
set PortB.0                               ` Turn LED on

End
```

Before we download this into the Tiny2313, let's take a closer look at these lines. The identification section is just as we discussed. It is just a series of comments, to you, to let you know what the program is supposed to be doing. The beginning of the configuration section, at least the directive statements, is the same as we described earlier. The only thing new in the configuration section is a statement that is new to us. That line (`Config PortB = Output`) tells the controller to set Port B up as an output port. Port pins can be set to either read the voltage level on the pin (0V, or 5V,) called input, or to place a voltage (0V or 5V) on the pin, this is called output. For this program, we want to have the pin be an output; so we set the port to output. We can have control over individual pins of a port, but for this example, it is easier to just set the entire port to output.



Finally, we get to the meat of the issue. The last line of this program (`set PortB.0`) is actually the part that does the work. The `set` statement tells the controller to make a pin high, or set it to plus 5 volts. The remainder of this statement tells the Tiny2313 controller which pin to set to high, PortB.0. Take a look at the diagram of the Tiny2313 chip; notice that pin number 12 on the chip is labeled as PB0. The P part of the tells us that it is a Port – that is a series of pins, each of which can be set up to either sense a voltage (0V or 5V) and react to it (the pin is called an *input* when it does this) or it can output a voltage (0V or 5V) which can control lights, relays, motors or other things (the pin is then called an *output*.) As the comment, on that line, mentions, this will light the LED on that pin. Port B is one of the four ports available on the Tiny2313.

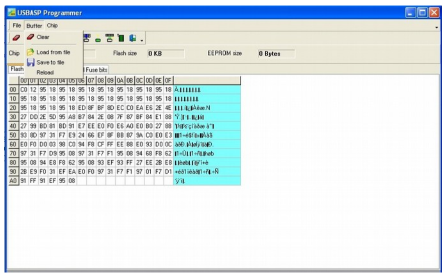

BASCOM-AVR IDE [2.0.7.5] - [C:\Program Files\MCS Electronics\BASCOM-AVR\Template.bas]
File Edit View Program Tools Options Window Help
Template.bas
Sub
Label
Project
BASCOM-AVR V2.0.7.5
Compiling Template.bas
Pass 1

```
' Title:
' Author:
' Date:
' This program is designed to

' Configuration Section
$regfile = "attiny2313.dat"              ' specify the used micro
$crystal = 1000000                       ' used crystal frequency
$hwstack = 32                           ' default use 32 for the hardware stack
$swstack = 10                           ' default use 10 for the SW stack
$framesize = 40                          ' default use 40 for the frame space

Config PORTB = Output

' Main Program

Set PORTB.0
Wait 3
Reset PORTB.0

End
```

Now that we have written our program, we need to convert it into machine code, and then send the program to the controller (or download it.) Remember from earlier, that the process of converting a program from a human-like language, such as BASIC, into a machine language is called compiling. In order to compile the program, you will need to click on the Program menu, and then click Compile. This will start the compiler and that will convert your program into something that the '2313 chip can understand, and the machine program is stored in BASCOM's buffer area, ready to send to the Tiny2313 chip.

After your program is compiled, make sure that there are no errors. If there are any errors, a new window will open below your main program window showing a list of all errors. If there are any errors, carefully compare your program to the program in this book, correct the errors and re-compile by clicking on the Program menu and then clicking on Compile again.

Now that your machine language program has been stored in the buffer, you need to send the buffer to the microcontroller chip. Click on the Program menu again, and then click on Send to chip, near the bottom of the menu. This will open a new Programmer window. You can see the buffer in the main window of the Programmer window. To send the buffer to the microcontroller, click on the Chip menu and then click on Autoprogram. Your program will be sent to the Tiny2313 and then the Programmer window will disappear.



Chip
Flash size 0 KB
EEPROM size 0 Bytes

```
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
00 C0 12 95 18 95 18 95 18 95 18 95 18 95 18 95 18
10 95 18 95 18 95 18 95 18 95 18 95 18 95 18 95 18
20 95 18 95 18 95 18 ED 8F BF 8D EC C0 EA E6 2E 4E
30 27 DD 2E 5D 95 A8 B7 84 2E 08 7F 87 BF 84 E1 88
40 27 99 BD 81 BD 91 E7 EE E0 F0 E6 A0 E0 B0 27 88
50 93 8D 97 31 F7 E9 24 66 EF 8F BB 87 9A C0 E0 E3
60 E0 F0 D0 03 98 C0 94 F8 CF FF EE 88 E0 93 D0 0C
70 97 31 F7 D9 95 08 97 31 F7 F1 95 08 94 68 F8 62
80 95 08 94 E8 F8 62 95 08 93 EF 93 FF 27 EE 2B E8
90 2B E9 F0 31 EF EA E0 F0 97 31 F7 F1 97 01 F7 D1
A0 91 FF 91 EF 95 08
```

Take a look at your Tiny2313 Experimenter System; notice that the right-most LED is now lit. Congratulations are in order again. Things are going nicely, huh?

Now, I want you to take another look at the compiling process. Click on the Program menu, again and take another look at those menu options. Notice that just to the left of the menu entry for the Compile command, there is a small icon, which looks like an IC, and just to the right of the Compile is an F7. If you look at the tool bar, just a bit to the right of the Program menu, you will see another IC icon, exactly like the one to the left of the Compile command on the Program menu. This icon is a shortcut for the Compile command; it does the same thing. The F7, listed to the right of the Compile command is another shortcut. You can use either one of these to do the same thing.

You will also notice shortcuts for the Send to chip menu entry. You can use either method to compile, and download, that you wish. For this book, we will use the Function keys (F7 and F4 keys) because they are simpler.

Now, we have gotten the LED to light, but so what? My flashlight, when I was a kid, could do that! Well, we need to learn to crawl before we can learn to run, these are our baby steps. We need to have more control over that LED, so we should be able to have the Tiny2313 turn the light out. Let's revisit that set statement… If the statement, as you typed it in earlier, set the port PB0 to +5 volts (or high, or 1) then how would you reset the port to back to low (or 0)?

Before you make any changes, save the working program and start a new one. Click on the File menu in BASCOM, and then click on Save. This will save your LED Test 01 program. Now, Click on the File menu again, and this time click on Save as… Change the name of your program to LED Test 02, and then click Save. Also, update the first line, in the identification, with the new title for your program.

Change the set statement in your program to reset, as so:

```
reset PortB.0                          ` Turn LED off
```

Now, compile your new program by pressing the F7 function key on your keyboard. Make sure that there are no errors shown at the bottom of your BASCOM window, and then open the Programmer window by pressing the F4 function key.

Send your compiled program to the '2313 by clicking on the Autoprogram option on the Chip menu. The LED on your Experimenter Board will go out. Yep, that `reset` at the beginning of the line told the computer to put port PB0 to ground, or low voltage. There is no positive point in the circuit, so electrons are not attracted to anything and current does not flow anywhere. Thus the LED does not light up.

(I'll let you in on a small secret: when BASCOM sends the program to the controller, it resets the '2313. So, the LED would have gone out, even without the reset statement. That said, the reset statement does work as advertised, and we will be using it more properly pretty soon.)

Now that we have gotten the Tiny2313 to be able to turn the LED on and off on command, how about more advanced things? Since we have not really entered a real program here (that secret that I mentioned,) we do not need to give BASCOM a new program name.

Reenter that set statement above the reset statement in your program:

```
' Main Program
set PortB.0                          ‘ Turn LED on
reset PortB.0                        ‘ Turn LED off
```

This seems to be right, correct? The '2313 chip will put a high (+5V) signal on the PB0 pin and then immediately bring that pin back to ground. The +5V will cause electrons to flow through the LED, making it light, and then the low, or ground, on PB0 will cause the LED to go out, right?

Compile your new program by pressing the F7 function key on your keyboard. Make sure that there are no errors shown at the bottom of your BASCOM window, and then open the Programmer window by pressing the F4 function key. Again, send your compiled program to the '2313 by clicking on the Autoprogram option on the Chip menu.

Your Tiny2313 chip will immediately run the new program, lighting the LED and then immediately turning it back off. Right?

What? Wait a minute – What happened here? It appears that there is something wrong with the program, doesn't it? The LED didn't light at all. If you want to see the Tiny2313 run the program again, just press the reset button to the left of the Tiny2313 chip itself. The program is stored in the '2313 chip and the reset will cause the chip to run the program each time that it is pressed. Well, actually, the LED did light. It's just that computers work extremely quickly, and the Tiny2313 turned the LED on and then immediately turned it back off. This happened in only a few *millionths* of a second, far too fast for your eye to notice the flash.

If you look through Appendix A, you will run across a statement called `wait`. This statement tells the computer to do exactly what it says. The parameter, or number, that you type after the `wait` statement is the number of seconds that you want the Tiny2313 to pause, before continuing with the program.

Let’s fix up our program. In between the `set` and the `reset` statement, enter a `wait` statement with a delay of 1, like this:

```
' Main Program
set PortB.0                         ‘ Turn LED on
wait 1                              ‘ Pause for 1 second
reset PortB.0                       ‘ Turn LED off
```

Compile your new program by pressing the F7 function key on your keyboard. Make sure that there are no errors shown at the bottom of your BASCOM window, and then open the Programmer window by pressing the F4 function key. Again, send your compiled program to the ‘2313 by clicking on the Autoprogram option on the Chip menu.

Your Tiny2313 chip will immediately run the new program, lighting the LED and then after about one second, turning it back off. This time the program works, right? Remember, the computer does EXACTLY what you tell it to. If you tell it to do something too quickly, then that is exactly what it will do. When you told the Tiny2313 to turn the LED on and then off, it did just that; only it was so quick that it was pretty much useless to you. Again, if you want to see the Tiny2313 run your program again, just press the reset button on your Tiny2313 Experimenter System.

Now that your program works correctly, save your program, before going on. (One more reminder, click on File and then click on Save.)

Begin a new program by clicking on File and then clicking on Save As… and name the program LED Test 03. Change the wait line to: `wait 2`, what do you think this will do? Try compiling this program and downloading it to your Experimenter Board (press the F7 key and then the F4 key and then click on Autoprogram) – did it do what you thought? The number, after the `wait` statement, tells the controller how long (in seconds) to wait before continuing with the next statement (the `reset` statement.) This new program should leave the LED lit for about two seconds before turning it back off.

A tiny diversion is needed here: I keep saying “about” when referring to time. The reason for that is that the ‘2313 controller on your board uses what is called an RC tank for it’s timing circuitry. This is much cheaper than the more expensive crystal-controlled timing circuitry.

You do have a bit of control over the timing of your wait statement. Notice, back in the configuration section of your program, there is a `$crystal` command. This command tells the compiler how to set up timing for things like the wait statement. An AVR-ATtiny2313A comes from the factory set to run at an internal 1 MHz, or 1,000,000 clock cycles per second. When you entered the `$crystal` command and gave it the number 1000000, you were telling it to assume that the clock is at, exactly, 1000000 Hz (or 1 MHz.) As I mentioned before, the timing on these things (without extra circuitry) is close, but not perfect. It could be off by several percent. And, while it is good enough for beginning stuff, the timing is not perfect – and it will change slightly, with the temperature and humidity. Just don’t try to do anything requiring very precise timing.

You can experiment with the timing by changing the number after the wait statement to something like 5 or 10 (or higher,) and then using a good stopwatch. Once you can time the LED flashing time, play around with the $crystal statement in the configuration section. Try a few percent higher and lower, until you get it so that it seems to flash for exactly the correct time. Each time that you change it, you will need to recompile and then download the new program. If you would like to just re-run the same program, you can just push the reset button to the left of the ‘2313 on your board. Be careful to not change it by very much at a time – you could get it so that some things do not work.

While it is useful to have your controller wait for full seconds, computers really shine at doing things quickly. Sometimes you need to have your controller pause for a shorter period of time. A statement, which is related to the `wait` statement, that allows you much finer control over the delay times, is: `waitms`. You can find a description of the `waitms` statement in Appendix A, but this stands for “wait for a specified number of milliseconds (or 1/1000ths of a second.”) You would use it exactly as you would the `wait` statement, except that the time is given in thousandths of a second (`waitms` 1000 is the same as wait 1.)

Play around with the program, substituting the `waitms` statement for the `wait` statement. See what happens with your LED flashing. Try lowering the delay time; try to find the shortest delay that you can actually notice. The average human eye can notice things happening at about 30 times per second (or Hertz, which means “per second”). If a light blinks faster than that, your eye will not notice the blinking, and will tell your brain that the light is steadily off, or on. This 30 Hertz (or 30 Hz) flashing equals out to be about 33 ms, or about 16 ms on and then off. See how close you can get to 16 ms before the LED appears to remain off. Your eye will notice and respond better if the room light is lowered or the LED light is raised; try turning the room lights out while performing this experiment.

Try lengthening the pause statement. Enter numbers around 2000 or 4000, and try using a stopwatch to see how accurate you can get in starting the timer and then stopping it. You will not be able to get perfectly, exactly accurate, because human response time is measured in milliseconds. In other words, from the time that your eye sees the LED turn on, until your brain tells your finger to press start on the stopwatch, until your finger actually does the work, will take several hundreds of milliseconds. It can be kind of entertaining to see how close you can get to the time you entered in the pause statement.

Having the Tiny2313 flash the LED can be kind of fun, but still, a flashlight can do that – again, “baby steps.”

Quiz 4

1) How does an LED light? _____

   A) Electrical current flows from the cathode through to the anode and on to a positive source.

   B) Electrical current flows from the cathode through to the anode and on to a negative source.

   C) Electrical current flows from the anode through to the cathode and on to a positive source.

   D) Electrical current flows from the anode through to the cathode and on to a negative source.

2) What statement will cause an LED, with the anode connected to an output pin, to light? _____

   A) `output`
   B) `set`
   C) `reset`
   D) `high`

3) For safety, you need to make sure that power is applied to your Experimenter System before making any changes to the circuitry. (true or false) _____

4) How long will the program delay with a `waitms 10000` statement? _____

   A) 10000 seconds
   B) 10 milliseconds
   C) 10 seconds
   D) 100 milliseconds

5) Why were you not able to see the LED flash in your first attempt to flash the LED? _____

   A) The statements were not placed in a loop
   B) The `set` statement was used incorrectly
   C) The `reset` statement was used incorrectly
   D) The LED was turned off too quickly to see

Extra Credit:

What is the function of the two connectors labeled Servo1 and Servo2? _____

Let’s step it up a notch

One thing that computers can do well is repetition. Take your program (save it and rename it as LED Flash Test 01), and again, update the identification section of your program. We are going to add a new statement: `goto`. This tells the controller to stop executing the program instructions one after another, and to go to a new location. In order for your controller to know where it is supposed to go, we will add a new type of statement. At the beginning of the main program section, add a line like this:

```
' Main Program
Start:
```

The colon, at the end of the word `start`, tells BASCOM that the word “`start`” is a label. This means that the compiler is going to need to know where that label is, in the program.

Next, change the delay time in the `waitms` line back to the 1000ms that we had originally:

```
waitms 1000                        ‘ Pause for 1000 milliseconds
```

At the end of your program, add the new `goto` statement, like this:

```
goto Start
```

The “Start” after the `goto` statement tells the compiler to have the ‘2313 go back to the beginning (where the `Start` label is located) and continue running from that point.

Now, your program looks like:

```
' Main Program
Start:
set PortB.0                        ‘ Turn LED on
waitms 1000                        ‘ Pause for 1 second
reset PortB.0                      ‘ Turn LED off
goto Start
```

Does this look right? What we are telling the controller, is to skip the label in the first line (labels are for locations, they do not do anything themselves,) and then turn the LED on, pause for one second and then turn the LED back off. After this, the `goto` statement tells the controller to go back to the beginning of the Main Program (at the `Start:` label) and continue running from there.

After you compile this, you should see the LED flash on for one second, and then go back off, and then repeat this, continuously. This programming tool is called a loop. In other words, the program runs through turning the LED on and then back off and then loops back to the beginning to loop through that task again and again.

Compile your program again, and if there are no errors, go ahead and download it to your Tiny2313 Experimenter System. Does your LED flash, like you expected?

Oops, not again! This time it looks like the LED never goes out. What is happening here? Doesn't the `reset` statement turn the LED back off each time through the loop? Well, remember when you first tried to have your Tiny2313 blink the LED on and off? We now have a pause while the LED is on, but there is no pause while the LED is turned off. The Tiny2313 turns the LED off and then immediately back on. Again, this happens in microseconds (millionths of a second.) You need to tell the Tiny2313 to delay while the LED is turned off also. Don't forget that the controller does exactly what you tell it to, even if it is not what you want it to do. You need another `waitms` statement in your program. Place this `waitms` statement in your program after the reset statement and before the `goto` statement.

```
' Main Program
Start:
set PortB.0                        ‘ Turn LED on
waitms 1000                        ‘ Pause for 1 second
reset PortB.0                      ‘ Turn LED off
waitms 1000                        ‘ Pause for 1 second
goto Start
```

We need the controller to pause after it turns the LED on, and also after it turns the LED back off. After you compile and download the program to the controller, you should see the LED blink on and off, repeatedly. Now, sit back and watch the pretty blinking light. This kind of reminds me of the lazy flashing yellow traffic light in Radiator Springs, in the movie Cars – except that, every third flash is not actually longer than the rest. Congratulations, you have just written your first commercial-type program. Remember that Bill Gates got his start writing programs for traffic light systems.

Now, change the delay times in the `waitms` statements to 500 each. 500 milliseconds is one half a second, and so when you compile and download your program, you should see the LED flashing on and then off, about once per second. If you have one of those old mechanical clocks in your room, you will notice that the LED is flashing, pretty much, in time with the sound of the clock: tick… tock… tick… tock… Cool, right? You are getting sleepy… sleepy… sleeee…

How would you get the LED to stay lit every third flash, like the traffic light in Radiator Springs? Well, the simple way, would be to actually write out the program steps to flash the LED twice for the same amount of time, followed by a slightly longer flash for the third time (remember, that you could not tell that every third flash of the traffic light was longer unless you timed the flashes – make the third flash only slightly longer.)

You can do this in several different ways, but here is one way:

```
' Main Program
Start:
‘First flash
```

```
set PortB.0
waitms 1000
reset PortB.0
waitms 1000
` Second flash
set PortB.0
waitms 1000
reset PortB.0
waitms 1000
` Third flash - 10% longer
set PortB.0
waitms 1100
reset PortB.0
waitms 1000
goto Start
```

Also, notice that we are not commenting each and every line – for the simple program lines, which only do a single simple thing, you do not need to comment them. You should, however, comment the groups of lines that work together on a small task, like we have above. In addition, you should comment new single lines with what you think they should do – at least until you are familiar with the new statement.

Again, give this program a new name by updating the identification section and using the File•Save as…, and then compile it and download the program to your Tiny2313 Experimenter System. Now, you can start looking for a job as a traffic control programmer for the Radiator Springs Public Works department.

Notice that each of the `set` statement lines includes an apostrophe, followed by a comment. Remember, that any time BASCOM sees an apostrophe, in a program line, it ignores everything to the right of the apostrophe. This way, we can place comments inside our program, so that we (or other programmers) can tell what the program is supposed to be doing.

Play around with the delay times; change the `waitms` statements to see what happens when you choose different pause times. Try using different times for the on time and the off time. Can you make the LED wait patiently and then suddenly flash on once every several seconds? How about turning the LED on and then have it suddenly blink? Can you win a stare-down contest with your Tiny2313? Who will blink first?

Ok, that got boring pretty quickly, but it is a bit more advanced than just turning the LED on and off. Still, with some careful timing, the flashlight, that I had as a little kid, had a flash button on the side that would do similar things. More baby steps…

Ok, time to step it up a bit. Type in the following Main Program section:

```
      ' Main Program
      ` First
```

```
	set PortB.0
	waitms 100
	reset PortB.0
	waitms 100
	set PortB.0
	waitms 100
	reset PortB.0
	waitms 100
	set PortB.0
	waitms 100
	reset PortB.0
	waitms 300
	' Second
	set PortB.0
	waitms 300
	reset PortB.0
	waitms 100
	set PortB.0
	waitms 300
	reset PortB.0
	waitms 100
	set PortB.0
	waitms 300
	reset PortB.0
	waitms 300
	' Third
	set PortB.0
	waitms 100
	reset PortB.0
	waitms 100
	set PortB.0
	waitms 100
	reset PortB.0
	waitms 100
	set PortB.0
	waitms 100
	reset PortB.0
	waitms 300
```

Whew! You will definitely want to save this program before you lose it.

So, what does this program do? Just by looking at the program, you can already tell that it flashes the LED on and off. You can also tell that it is not a steady blinking, the LED turns on and off for different amounts of time. Compile and download your program to run this new program and watch the LED blinking on and off. Did you recognize the pattern? Press the reset button on your Tiny2313 Experimenter system to run the program again and try to recognize it.

The times that I chose are not random, the International Morse Code standard calls for a dot to be approximately one tenth of a second and a dash to be three tenths of a second. The time between dots and dashes of an individual letter are to be the same length as a dot, and the time between letters is to be as long as a dash. Run the program again. Do you see it yet? The program that you just typed in displays the International Distress Signal of S-O-S. My son downloaded an app for his $250 phone that uses the camera flash to do the same thing. Just think, your Tiny2313 has saved you over $200; you don't need that phone for this now.

Take a look at the Morse code in Appendix C and write a program to flash your name on the LED. Remember, for a dot, turn the LED on for 100ms and for a dash, turn it on for 300ms. Between elements of an individual letter, turn the LED off for 100ms, and between words, turn it off for 300ms. Experiment with having your Tiny2313 flash different messages in Morse Code; how about that *Hello World* program now? Remember what I said about how it can, sometimes, be downright difficult to output text on a microcontroller? This is one example of how that can be.

One of the contests, in which ham radio operators engage, is called Fox and Hounds. A radio transmitter (which is called the fox) is left in a hidden place, and just broadcasts a Morse Code message, while other radio operators (the hounds) listen in, and try to find the fox. This program that you just wrote is the same as the program that would run the fox in this type of contest, the only real difference is that instead of lighting up an LED, the controller sends a Morse Code signal. Similar programs would be used for emergency transmitter beacons used on the ocean.

Yes, with a lot of work, my flashlight can do Morse Code; but, boy, those baby steps are getting bigger now, aren't they?

Including the sink?

I want you to remember that we set up our LED circuit so that when you output a 1 to the output port, the LED will light up. When you output a 0 to the output port, the LED will go out. Since we often use 1 to mean on and 0 to mean off, this seems to make the most sense. However, that is not always the case.

In computer engineering terminology, an output can either sink or source current for the output device. With regards to our LED circuit, one end of the circuit is tied to our output port (PortB.0 so far,) while the other end is tied to ground (through the LED and resistor.) This configuration is called current sourcing. If the LED were turned around and tied to the positive power source, then that would be called current sinking.

When your output port is in current sink mode, you will need to output a low (or 0) to turn the LED on (current flows from the output port through the resistor and LED and to the positive power source.) You can experiment with this by turning off and unplugging the Tiny2313 Experimenter System ISP cable and retrieving the LED/resistor pair that you used earlier. Plug the resistor lead into the female header for the left-most LED, and reinserting it with the LED lead plugged into the I/O header socket for PB1 (just to the left of the jumper wire in port PB0.) Rename your program and enter these commands:

```
' Main Program
‘ Flash LED 2
     set PortB.1
     wait 2
     reset PortB.1
     wait 2
```

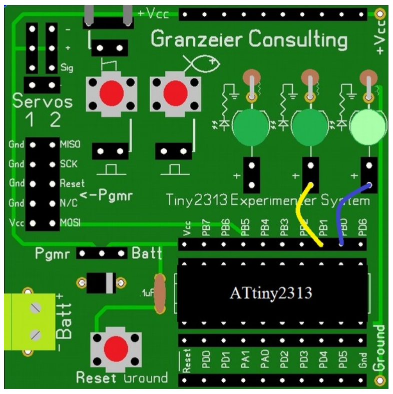
Notice that, since we are now trying to control an LED connected to PortB.1, we need to change the statements in our program. Earlier, when you had the program output a 1 (with the `set` command,) it would turn the LED on. With the above program, the LED would light for two seconds and then go out. This time the LED starts off not lit, but the after two seconds it lights. That’s strange, huh? This seems to go against all logic, why would you use a 0 to indicate on and a 1 to indicate off? Well, the truth is that sometimes you need to think in a reverse type of logic. In some circuits, an electronics device can source much more current that it can sink. In some older chips, the source capabilities can be many times greater than the sink capabilities. When you are working with those other types of microcontrollers or other circuitry, be aware that you may need to use this “reverse” logic.

Like most other modern microcontrollers, the Tiny2313 is not subject to this restriction; your chip can sink or source the same amount – 40 milliamps. We do not need to worry about reverse logic here, just keep it in mind in case you run across it later. For now, turn off the Experimenter System and remove the LED/resistor pair, so that it is back to the way you had it before. You are done with your LED/resistor pair for this book, but you may want to play with it to test other circuits. Just remember that you do not want to exceed about 5 volts.

Look Ma, two lights!

For our next set of experiments, take another short jumper wire and strip the insulation off the ends. Don't forget to turn off the Tiny2313 Experimenter System before making any wiring changes. Carefully, place one end into the socket for PB1 (like you had with the LED/resistor pair, and the other end of the jumper wire into the female header connection of the middle LED above the Tiny2313 chip. The picture to the right shows the new jumper wire in yellow. Leave the original jumper wire where it was.

> ***Important Note:***
> *You want to make sure that you do not connect two different I/O pins from your 2313 together. This may damage your controller.*



Now, enter the following program (don't forget the Identification and Configuration sections):

```
' Main Program
     Do                                 ‘ Introducing the Do Loop
       set PortB.0
       reset PortB.1
       waitms 500
       reset PortB.0
       set PortB.1
       waitms 500
     Loop
```

Change the name with File•Save as… to save this, and then compile and download it. Also, before you run it, take a close look at the program. First, you will see that the `Start:` label and the `goto` statement have been removed. They have been replaced with, what is called a `Do Loop`. This is a more acceptable way of creating a main loop for a program that repeats itself forever. At first, we used the `goto` statement, because that is simpler and easier for a beginner to understand. Unfortunately, many beginning programmers tend to over-use the `goto` statement and their programs grow to be unreadable. More experienced programmers tend to use the `Do Loop`, over the `goto` loop, to make their programs more readable. Also, you can see that we have indented (or put spaces at the beginning of) the lines inside the loop. This is also to make the program easier to read – everything that is indented is repeated each time the computer runs through the loop. The BASCOM compiler does not care if you indent these lines, but it makes your program easier to follow. Because this format is easier to read (and is also more standard,) we will be following this convention from now on; after all, you are not really a beginner any more, are you? Not with you having already successfully programmed two traffic warning systems (including one for Radiator Springs,) a contest "fox" transmitter and an emergency beacon - so far.

Now, notice, how, in the first two lines inside the loop (the `set` and `reset` statements,) the LED for PB0 is turned on, while the LED for PB1 is turned off. Then, after about a half-a-second, they are switched so that the LED for PB0 is off, while the LED for PB1 is turned back on. After another short delay, the program repeats. This is kind of like the warning lights of a railroad crossing. In fact, you could use this exact setup to create a warning light for a model railroad setup. All that you would need would be a way for the Tiny2313 to tell when a train is approaching. A very simple way for your 2313 to do this, is to have the train turn the controller on and then, after crossing the road, turn the controller back off. With the above program loaded into your Tiny2313, turn the battery power switch off. Now, when you turn the Tiny2313 power back on, the LEDs will begin alternately flashing, until you turn it back off.

Again, experiment with changing the times on your program. Make them shorter and longer and watch what happens with the LEDs flashing. Change the times so that they are not the same. Make one LED stay on longer than the other.

See if you can get a light version of the old “Indian War Drums”, where the beats are the same, except that every fourth beat is harder. For this, you would only have one LED on at a time. You could set it up so that one LED will flash three times, followed by the other LED flashing once. Another way to do that may be to have the one LED flash for short durations three times, followed by the other (strong beat) LED flashing for a little longer time. Try both ways, and see what the differences are. Play with the pause between “beats”, and see how you can get it looking better.

There, let’s see my old flashlight do that! Those steps are not so much baby steps any more, right?

Quiz 5

1) What is the programming tool that allows the controller to repeat program steps? _____
   - A) A do loop
   - B) A loop
   - C) A `for…next` statement pair
   - D) A while…wend loop

2) What is the purpose of the `goto` statement? _____
   - A) To create a program flow
   - B) To count iterations of a loop
   - C) To serve as a program comment
   - D) To change the normal top-to-bottom flow of a running program

3) In the microcontroller world, a *Hello World* program outputs a line of text saying "Hello World." (true or false) _____

4) What does a colon at the end of a statement mean? _____
   - A) It marks a label
   - B) It is an immediate command
   - C) It marks the end of a statement line
   - D) It is used to separate statements on a multi-statement line

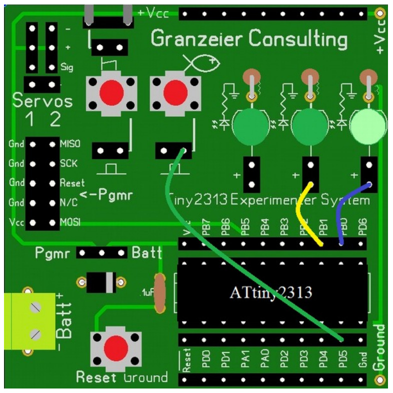
5) What is the shortcut key to compile a program? _____
   - A) F7
   - B) F6
   - C) F5
   - D) F4

Extra Credit:

In the normal configuration, does an output pin, on the Tiny2313 Experimenter System sink, or source, current when connected to one of the on-board LEDs? _____

To reach the 2313, press 1

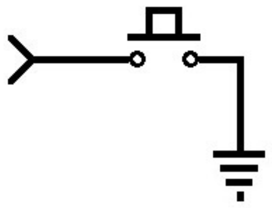
For our next experiment, we will be using one of the small pushbutton switches. You will need to cut one more piece of jumper wire and strip the ends to about ¼". Don't forget to remove the power from your Experimenter System. Carefully plug one end of the new jumper wire into the 2-pin header for the right-most push button, and the other end of the wire into the socket of the Tiny2313 chip, labelled PD5. This socket pin is on the bottom of the Tiny2313 chip, in the second from the right-most hole.



A switch is basically like two pieces of wire which, when the switch is off (or the button is not pressed, with these buttons – that is what the "normally open" means) then the wires are held apart from each other. This prevents electricity from flowing through the switch. When the switch is turned on (or the button is pressed), the wires inside the switch are connected together and electricity is able to flow.

There is another type of pushbutton switch called a normally-closed switch; this switch normally has the leads connected together inside and breaks them apart when you push the button. We won't be using any of those here, but keep this in the back of your mind.

Tiny2313 ports, when they have been configured as inputs, have a small resistor tied between the port pin and +V. This is called a "pull-up" resistor and "pulls up" the pin to +5V. This avoids what is called a floating input, or an input, which is neither high nor low, and could be seen as either (or even worse, keep changing.)



Notice, in the schematic diagram to the left, that there is a break in the line from ground up to the arrow representing the input port of the '2313. The switch (looks like the top hat in the schematic) has a piece of metal, that when you press the button, touches the two electrical contacts, making the path complete. The switch between ground and port PD5 will allow electricity to flow from ground through the internal pull-up resistor ONLY when it is pressed. When the button is not pressed, PortD.5 will "see" a high signal (through the pull-up resistor) and when you press the button, PD5 will be connected to (or "see") the low from the ground. Another important purpose of the pull-up resistor is to prevent a short circuit between the +5V of high and the 0V of ground when you connect a port pin to ground.

Now that we have a way for the Tiny2313 to detect a button being pressed, how do we get the program to respond? Enter this modification to our simple railroad-crossing program:

```
' Title: Button Test
' Author: Art Granzeier, Granzeier Consulting (again, use your name)
' Date: 13 Oct 13 (again, use today's date)
' Description: Experiment with detecting a button press
' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output
Config PortD = Input
Set PinD.5                          ‛ Activate Pull-up Resistor

' Main Program

Do                                  ‛ Wait for button
  If Pind.5 = 0 Then Goto Start
Loop

Start:

Do                                  ‛ Flash LEDs
       set PortB.0
       reset PortB.1
       waitms 500
       reset PortB.0
       set PortB.1
       waitms 500
Loop

End
```

There are a few differences: first is the addition of two additional `Config` statements and the simple 3-line loop just after the Main Program comment, and the Start: label. The new `Config` statements are the addition of the input configuration for Port D, which is just like the output configuration statement, except it sets the pins for input to the Tiny2313, rather than output from the '2313. The other statement is to set the input pin for D.5. This seems strange, trying to output on an input pin; but setting the pin for an input just activates the pull-up resistor.

You can figure out that the first loop continues looping until something happens; but it isn't all that clear what that "something" is, right? The Pind.5 function is easier if you break it down to "pin d.5." In other words, check to see if the voltage on Pin 5 of port D is set to 0, or a low level. If the level of Pin D.5 is set to low, then the `if…then` statement will perform the part after the then clause, which is: `goto start`. If Pin D.5 is set to 1, or high, then everything after the `then` clause (the `goto` statement) will be ignored, and the program will continue down to the `Loop` statement on the next line.

So, these four lines (`Do`, `If…`, `Loop` and `Start:`) will form a delay, making the '2313 chip pause until the voltage on Pin D.5 goes low. With the Tiny2313 Experimenter System wired up as we currently have it, Pin D.5 will "see" a high voltage (logic 1) through the pull-up resistor, until you push the button.

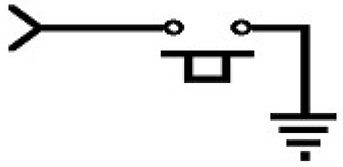
Compile the above program, and download it to the '2313 on your Experimenter System. After the download, the program will start running, and the system will pause until you push the button (excuse me, drive the train over the sensor. Once you push the button, the LEDs will begin their normal flashing.

We will make one small improvement before continuing. In the pause loop above, we check for the input inside the loop. A Do loop can actually check for a condition in the structure of the loop itself. The loop statement can check for the input on Pin D.5 and loop based on that input. Replace the first loop in the above program with:

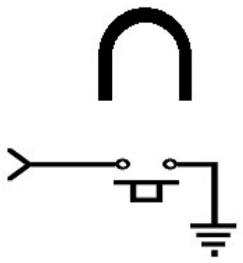
```
Do                                  ' Also waits for button
Loop Until Pind.5 = 0
```

Compile your new program, with the tighter loop, and download it to your '2313. You will notice that it runs just like the previous version. The Experimenter System will wait until you press the button and then the lights will flash until you reset the system. After the reset, it will start all over again, waiting for you to push the button.

This makes for a tighter loop (less program space, and faster execution – these don't seem all that important right now, but when you start dealing with much bigger, more complex systems, then size and speed will become more important.) It is also a bit easier to read, so we will use these conditional loops from now on.



Let's consider the pushbutton switch for a minute. Take a look at the picture to the left. Notice that it is, almost, exactly like the schematic diagram for our pushbutton above. The only real difference is that the switch has been placed, in the drawing, upside down. Remember that the mechanics of a switch is just a piece of metal (represented by the, now upside-down, top hat.) If that metal were to be made of some magnetic-sensitive material (like steel) and were placed on a railroad track so that a magnet on the bottom of a train would pull the metal up, thus completing the circuit, we would have something like this:

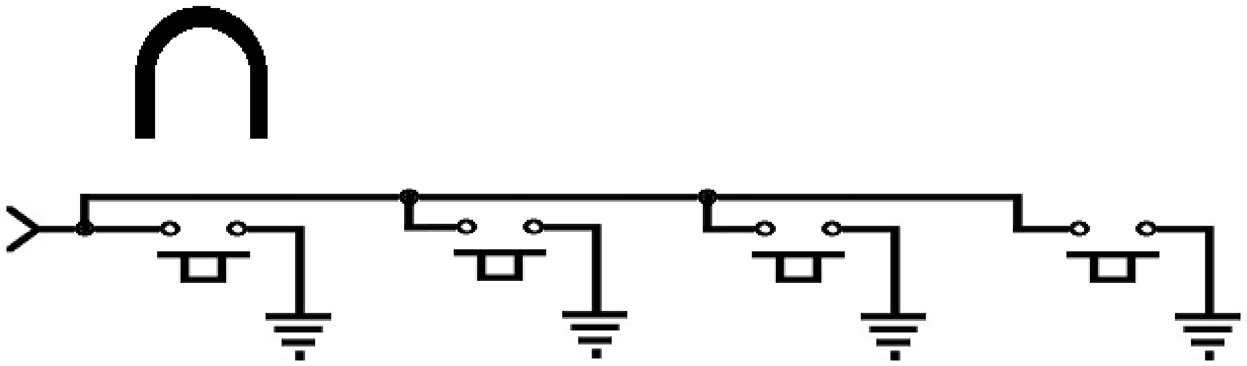


A real-world switch that can meet this need is called a magnetic reed switch. The switch remains open until it comes near a magnetic field, then the thin metal reeds come in contact with each other. Electrically, this is exactly the same circuit that you have connected to Pin D.5 of your Experimenter System. The only real difference is that the magnet under the train is pushing the button for you, rather than your finger.

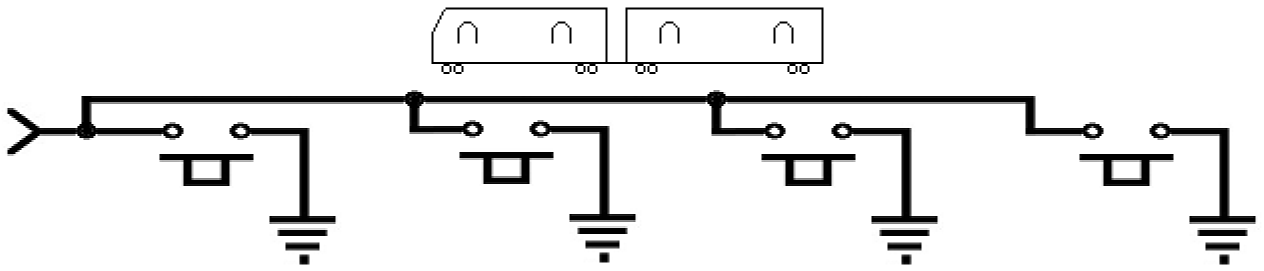
This is the beauty of using a development system. In addition to our purpose of learning about microcontrollers, you can also do low-level simulation of real-world systems. In this case, we are using the on-board pushbutton to simulate a magnetic sensor for a train/railroad-crossing system. By using your imagination, a little bit, you can start work on developing the program for the system. After you get it working the way you think that it should, you can then take the program and install it into the actual railroad-crossing system and you will only have a small bit of debugging (or fixing up) the program left. This saves you time and money – it is faster to work on a development system in your office, and your Tiny2313 Experimenter System costs much less than a real train crossing system. So long as your development system uses the same microcontroller as your real-world system, you can do much of your work in a nice, comfortable office.

Ok, this is pretty simple, but you do have a very good simulation of a real-world system: it is entirely reasonable to set up a controller to do one thing, one time, and then have the system reset the controller to begin all over again – some times. This would work with the "start" button on one side of the street, and the reset button on the opposite side of the street. However, it would only work for a one-way track – the train must pass over the "start" button first, and then, after clearing the street, pass over the reset button. The trolley tracks near my house operate like this; there is one track for trolleys headed into the downtown area, and a separate track for trolleys headed out to the suburbs. The trolleys would, normally, never use the opposite tracks. However, when there is a trouble with one of the tracks, the trolleys will need to switch over to the opposite tracks to continue running. A railroad-crossing system like this one would fail when the trolleys are running "backwards", on the "wrong" track.

Another possibility is to have the LEDs flashing, as long as the button is pressed. Consider this diagram:



You will notice that the magnet could be over any of the magnetic sensors. Also, notice that if any one (or more) of the pushbutton switches (sensors) are pressed, it will provide a path for current to flow from the ground, up through the switch and then through the internal pull-up resistor allowing the input pin to see a ground (or 0.) This arrangement could be expanded out to many more switches. If the sensors were to be placed so that the magnet under the train was always over at least one sensor (perhaps with multiple magnets,) then this could be used for trains going either direction.



In this diagram (ok, so I am not an artist, work with me here…) you can see that there are multiple magnets on the train, and that no matter where the train is over the sensors, at least, one of the sensors will be activated, allowing the input to read a 0. If the street crossing is in the middle of these sensors, then our '2313 could sense when the train is within range of the street, and then it could activate the warning lights. In addition, it would work, no matter which direction the train was headed.

So, how would we handle this in the program? First of all, we can start with the program that we already have. We want the program to wait for the button to be pressed (err, the train to arrive over the sensor.) While waiting, we want to make sure that both LEDs are off – the first time through, that will be the case from the reset, but after the train passes, we want to turn both LEDs back off. We also want the basic LED flashing, as we have currently. The only difference would be that we need to have the controller detect when the input is back to a high (or the train has passed, and all of the switches are opened again, pulling the input to high through the pull-up resistor.

Let's add an `if…then` statement inside the loop – actually, since there are delays inside the loop, let's add one `if…then` statement before each delay. These `if…then` statements will check to see if the switch is not pressed (or the sensor is not reporting a train,) and then exit the flashing loop, and go back to wait for another train to approach.

```
' Main Program

Wait:
     reset PortB.0                     ' Ensure both LEDs are off
     reset PortB.1
Do                                ' Wait for button
Loop Until Pind.5 = 0

Start:

Do                                ' Flash LEDs while button
       set PortB.0
       reset PortB.1
       if Pind.5 = 1 then Goto Wait
  waitms 500
       reset PortB.0
       set PortB.1
       if Pind.5 = 1 then Goto Wait
       waitms 500
Loop
```

Well, this has now gone well beyond anything that my old flashlight could do. I would say that you are well beyond the baby steps level at this point. Try to think of a few other ways that you could use a switch, or sensor, in a control system.

Quiz 6

1) What is the purpose of the empty do loop (`Do…Loop Until Pind.5 = 0`)? _____
   - A) Causes the '2313 to count up rapidly
   - B) Pauses program execution until the voltage on Port D.5 goes low
   - C) Pauses program execution until the voltage on Port D.5 goes high
   - D) It is a simple delay loop

2) How does one of our pushbutton switches work? _____
   - A) Two pieces of metal are brought into contact when the button is pressed
   - B) FM (Foul Magic)
   - C) Two pieces of metal are in contact with each other until the button is pressed
   - D) The button measures change in capacitance when a human's finger touches it

3) With the Tiny2313 Experimenter System wired up as we currently have it, Pin D.5 will "see" a low voltage (logic 0,) when you push the button. (true or false) _____

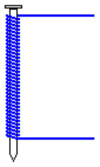
4) How does the input pin read a logic 1 when the pin is not connected to ground? _____
   - A) A floating input is always read as a high
   - B) Current flows out of the input pin and goes to the positive terminal of the battery
   - C) Through the pull-down resistor network
   - D) With no connection to ground, the input pin senses a high through the pull-up resistor

5) In our railroad-crossing scenario, how many sensors would you need? _____
   - A) Enough that the magnets on the train are continuously activating at least one
   - B) One, at the street
   - C) At least 2, one for each direction
   - D) Four

Extra Credit:

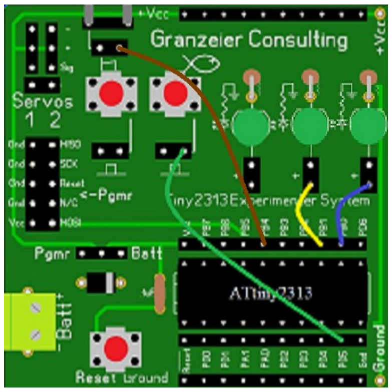
Describe a different, real-world situation, and how you would set up a simulation of that on your Tiny2313 Experimenter System. Write a rough, first draft of the main part of your control program for your system. _____

Now listen up!



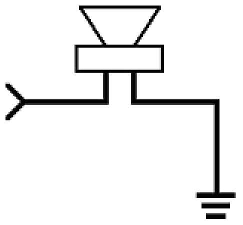
A speaker is, actually, a pretty simple device. The electronic element of the speaker is nothing more than an electro-magnet. Many of you built your own electro-magnet when you were child; you just coiled a wire around a nail, and then connected the ends of the wire to a battery to turn the magnet on. Once you turned the magnet on, you were able to use the nail to pick up small pieces of metal.

A speaker uses an electro-magnet to attract, and repel, a small magnet, which is attached to a thin membrane made of paper or plastic. When the electro-magnet is turned on and off rapidly, it will cause the permanent magnet to vibrate, thus vibrating the thin membrane. It is this membrane vibrating which causes the air to vibrate, which is interpreted by our ears as sound.



Take another short piece of hookup wire and strip the insulation off the ends. Plug one end into the female header under the speaker and the other end into the female header above the Attiny2313 chip. This should go into Port B.4. This is shown in the picture to the left, with the new hookup wire displayed in brown. You should leave the other wires, which were installed in previous steps, as we will be using them later.

Now, keeping with our precedent of using Port B for output and Port D for input, Port B.4 will control the speaker. Take a look at the schematic, to the right. When Port B.4 is set to high, current will flow from ground through the speaker and to the output pin. This will act the same as when you connected the wire ends in your electro-magnet to a battery, and will activate the electro-magnet inside the speaker. When you output a low on Pin B.4, there will be no positive point in the circuit, so no current will flow, and the electro-magnet will be turned off.



In the same way that we used loops to turn the LEDs on and off repeatedly, we will again use loops to magnetize and de-magnetize the electro-magnet inside the speaker. The main difference will be that we will use much shorter delay times within our loops. Using your template, add the following program and then save it with a name like Speaker Test.

```
Config PortB = Output

' Main Program

Do                                  ` Toggle speaker line
       set PortB.4
       waitms 1
       reset PortB.4
       waitms 1
Loop
```

Again, take a look at this program: we see a standard loop, nothing new here. Also, we see that, inside the loop, we set port B.4 (the speaker pin) to high, wait 1millisecond and then reset it back to low and wait for another 1 millisecond. Because these statements are within a loop, this will repeat, turning the magnet in the speaker on and off at 500 times per second – two milliseconds on and off, and repeated.

Compile your program and download it to the '2313. What happens? With the speaker activating at 500 Hz (Hertz, or times per second) you should hear a tone coming from the speaker. Try changing the delay times, what happens? See how large you can get the delay time before it no longer sounds like a tone, but rather a series of clicks.

There is one more statement set that I want to introduce; the `for…next` loop. This loop includes the ability for the controller to keep count of how many times it executes the loop. In it's basic format, the loop starts with a `for` statement, and then it ends with a `next` statement. You can best understand this loop with an example:

```
For cntr = 1 to 100      ' Toggle speaker line 100 times
  set PortB.4
  waitms 1
  reset PortB.4
  waitms 1
Next cntr
```

Notice the similarity with other loops that we have already covered; there is a beginning to the loop (the `for` statement) and an end (the `next` statement) and everything inside gets repeated a certain number of times. At the beginning of the loop, the `for` statement uses a special area of memory, called a variable, to hold a number. The variable is given the name "cntr", and it is first set equal to the first number in the `for` statement (in this case 1). The commands inside the `for…next` loop are then executed. At the end of the loop, with the `next` statement, two things happen: first the variable is compared to the highest number in the `for` statement (in this case 100), if the number is equal (or higher) then the `next` statement exits the loop, and the program continues after the loop. Secondly, if the variable, "cntr", is less than the limit number (100,) then "cntr" is incremented, and program flow loops back to the `for` statement.

In order to use a variable, you must set up the special area of memory. To do this you will need to add a new configuration statement. After the output config statement, add the following line:

```
Dim cntr as byte
```

This will set aside one byte of memory (one memory location) to hold the number that we call cntr. One byte can hold any integer between 0 and 255. We use this strange spelling of counter because the word counter itself is reserved for BASIC to use. So we just change it's name and everything works fine. If, during your experimenting, you wish to use more than 255 in your `for…next` loop, remember to change the `byte` parameter – try using `long`.

So, the first time through the loop, the variable "cntr" will hold the number one. At the end of the loop, the `next` statement will check and see that "cntr" is less than 100 (the limit, as set up by the `for` statement,) will increment “cntr” to 2, and then have the program loop back to the `for` statement.

The next time through the loop, everything is repeated, except that “cntr” contains the number 2. This is repeated, with the variable “cntr” being incremented each time, until it reaches 100. When “cntr” is equal to 100, then the `next` statement will say “that’s enough times through the loop, go on to the next statement.”

As you can see, this is a way to repeat some steps a certain number of times. How does that help us here? How about if we write a loop to toggle the speaker magnet on and off a specific number of times? Since we know how long we leave the magnet on and how long we leave the magnet off, we can calculate how long the tone will last. Take another look at the above loop; how long will that last? If we consider that the magnet is on for 1 millisecond and then turns off for another 1 millisecond, and then it loops for 100 times, we find that the tone should last for 200 milliseconds, or 1/5th of a second. Actually, it will last for just over 200 milliseconds because each of the other statements, including the loop statements themselves, will take a small amount of time. However, for the purposes of this class, we can ignore the overhead time – just keep it in the back of your mind, in case you need better timing later on.

How about if we embed this short beep routine into our railroad-crossing program? Take a look at this:

```
' Title: Railroad Crossing w/ Sound Test
' Author: Art Granzeier, Granzeier Consulting (again, use your name)
' Date: 13 Oct 13 (again, use today's date)
' Description: Simulate a railroad crossing warning system

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output
Config PortD = Input
     Dim Cntr as byte

' Main Program

Wait:                                ` Wait until train arrives
Do
Loop Until Pind.5 = 0

Start:
```

```
Do
       set PortB.0                       ‘ Turn right light on
       reset PortB.1                     ‘ Turn left light off
       if Pind.5 = 1 then Goto Wait      ‘ If train is gone, restart
       For cntr = 1 to 100               ‘ Play tone for 200 ms
         set PortB.4
         waitms 1
         reset PortB.4
         waitms 1
       Next cntr
  waitms 300                        ‘ Pause for 300 ms
       reset PortB.0                     ‘ Turn right LED off
       set PortB.1                       ‘ Turn left LED on
       if Pind.5 = 1 then Goto Wait      ‘ If train is gone, restart
       For cntr = 1 to 100               ‘ Play tone for 200 ms
         set PortB.4
         waitms 1
         reset PortB.4
         waitms 1
       Next cntr
  waitms 300                        ‘ Pause for 300 ms
Loop

End
```

When you compile and download this program, you will need to hold the pushbutton down for a minimum of several seconds; remember that the button is simulating the sensor over which the train travels. Trains should not be going several hundreds of miles per hour – the train will be over the sensor array from the time that it nears the crossing, until it leaves the area. So, your sensor will activate for a minimum of several seconds, which will allow the lights time to flash.

Hint: for higher-pitched frequencies, take a look at the BASCOM-AVR manual under the `waitus` statement.

I think that my old flashlight just ran away with embarrassment!

Welcome to the real world

By the way, in the same way that you can use the output port to control an electro-magnet in a speaker, you can also use this basic setup to control an electro-magnet directly. This could be useful if you were to need to build an electro-magnetic crane device into your system. What you will need to do is to make sure that your electro-magnet circuit does not draw more current than your Tiny2313 (or whatever control system you use) can provide. If the magnet does draw more current than the controller can provide, you could use a driver circuit, or a relay, to boost the signal strength.

In the same way, you can control motors, electric heater elements, pumps, coffee makers, and just about any other type device. Now that I think about it, one way that you could switch a track for a train (model railroading here, this would be too small for a real train) would be to use an electro-magnet to switch the track to redirect the train. Just use a signal booster circuit, of some kind, and write programs similar to what you have done here.

For input devices, you already know about having the computer read a simple switch. As you have seen in this chapter, the switch could be a regular pushbutton switch, but it could also be a magnetic train-sensor. Standard toggle switches (the kind that you use to turn your bedroom light on and off) work in a similar way, but the contacts remain opened or closed, depending on the position of your switch. Push button switches are often used as limit switches, to let a control computer know when a motor has moved something to it's farthest position. A thermostat, like the one used to control your furnace and/or air conditioner, is nothing more than a simple switch (that changes the point at which is opened or closed based on the temperature) – you can connect a thermostat to an input pin, and ground, and your Tiny2313 Experimenter System can tell when the temperature is above, or below, the set temperature.

One other type of input (among millions of types in the world of control systems) would be an operator input. For example, if you were to set up five switches, going to five different input pins, you could use those pushbuttons to have your program know what options the human operator selects. Perhaps you could use one button for each of several different songs, or light pattern displays. The possibilities are, literally, endless.

You would need to do a bit more research on basic electronics before undertaking jobs like these – but you already have an understanding of the controller, and programming, end of the tasks.

Quiz 7

1) What is a main advantage of a `for…next` loop? _____
   - A) It is faster
   - B) It takes less program steps
   - C) It uses less memory
   - D) It maintains a count of each loop

2) In what mode would you place the Port pin for a speaker? _____
   - A) Input
   - B) Output
   - C) Bi-directional
   - D) Tri-state

3) A `waitms` parameter of 10000 is equal to a 1 second delay. (true or false) _____

4) Which statement would you change to make an LED attached to Port B.7 stay on longer? _____
   - A) `Set PortB.7`
   - C) `Waitms 250`
   - C) `Reset PortB.7`
   - D) `Loop`

5) If you use a standard Do loop for the speaker, how long will the speaker sound? _____
   - A) One second
   - B) Ten seconds
   - C) 100 milliseconds
   - D) For ever

Extra Credit:

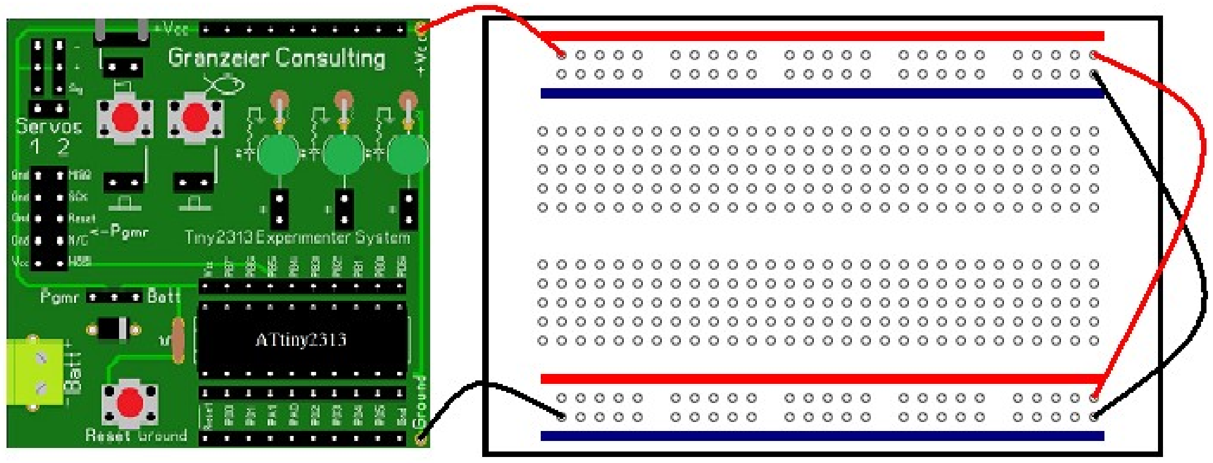
Scenario: You have a computer-controlled incubator for baby chicks. The heater is connected to a relay connected between an AVR Tiny2313A pin B.0 and ground. The thermostat is connected between pin D.0 and ground, and is set to 101.5 °F (this is not a forced air incubator.) Write a program statement that will turn on the heater after your Tiny2313 detects that the temperature is below the set temperature.

Where do I go from here?

Congratulations, you now have a basic understanding of what a microcontroller is, and what it can do. You have built your own development system, and set up several circuits. You have started learning a beginners programming language, although it is a real language, and many professionally developed products are written in BASCOM-AVR. You learned to teach your Atmel AVR ATtiny2313A microcontroller many new tricks, and hopefully, you had fun doing so.

If you are interested in continuing with this new hobby, there are a few things that you will want to do. First, spend some time going through the BASCOM manual that you downloaded earlier in this course. Next, play! Playing is the best way to learn about new things. Take the experiments that were presented in this book, and follow the suggestions for changes. Then make more changes to the program and watch what happens. Connect the third LED and then see if you can make the LEDs flash in sequence, from left to right, then try to make them flash from right to left. Connect the other button and try to have the LEDs flash in sequence on command – if you press the left button, have the LEDs flash from right to left, and vice-a-verse. How about making a miniature KITT scanner (from the TV series *Knight Rider*,) or a mini Cylon scanner (from the TV series *Battlestar Galactica*)?

In addition, there are lots of projects on the Internet; do a search for “simple tiny2313 projects.” You will be able to duplicate many of these, right on your Tiny2313 Experimenter System. For some of these projects, you may need to expand the board. Just to the right of the ground and +V female headers, you will find an additional hole in the Printed Circuit Board – as I mentioned in the assembly instructions, these are for expanding your Experimenter System. Solder in an additional, short piece of red wire to the +V hole, and a black wire to the ground hole. These wires will provide power to a solderless breadboard so that you can add additional circuits, beyond what your System provides.



You can find out more about breadboards, including a bit of history, and how to use them, on my “Breadboard” page, at: http://projects.granzeier.com/what-is-a-breadboard/. This breadboard expansion will allow you to add any device to your Experimenter System, not just the three LEDs, two pushbutton switches and the speaker that is on-board.

You can get many more project ideas by trying to duplicate projects for other microcontrollers. Parallax produces a tiny microcontroller called the BASIC Stamp. This controller is similar, in speed and capabilities, to your Attiny2313A. Many years ago, Scott Edwards wrote a column on the BASIC Stamp in Nuts & Volts magazine. You can find a couple of compilations of his columns in his Stamp Apps books. These can be found at: http://www.hth.com/filelibrary/PDFFILES/ST_APP1.PDF, and http://www.hth.com/filelibrary/PDFFILES/ST_APP2.PDF. While these books are for a different microcontroller, your Tiny2313 can do everything that the BASIC Stamp can do. You will need to look through the BASCOM manual for the BASCOM equivalents to the BASIC Stamp statements – they should be similar.

Once again, congratulations on completing this course, and I hope that you have learned a lot, and have had some fun in your learning.

“The heart of the prudent acquires knowledge, and the ear of the wise seeks knowledge.”

Proverbs 18:15 - New King James Version (NKJV)

APPENDIX A – Selected BASCOM-AVR Statements:

Bitwait
Syntax: Bitwait <var>**, set/reset**
Description: This statement causes the program to pause until the specified bit changes to either set or reset.
Example:
```
        Bitwait PinD.5, reset  ‘ A pushbutton is connected between Pin D.5 and ground
        …
```
This program segment will cause the program to wait until the pushbutton has been pressed, grounding Pin D.5, and then it will continue with the next statement in the program.

FOR … NEXT
Syntax: FOR <var> = <expr> TO <expr> {STEP <expr>}
Description: This sets up a standard Basic FOR/NEXT loop. The variable is set to the value of the first expression and tested against the limit given by the value of the second expression (which is evaluated only once). The optional step size may be positive or negative. If negative, the limit test is appropriately changed. Improperly nested FOR/NEXT statement pairs may cause incorrect execution without an error message. Default STEP value is +1.
Example:
```
for x = 1 to 10
  set PortB.0
  wait 1
  reset PortB.0
  wait 1
next x
```
This program segment will flash an LED, attached to PortB.0, 10 times.

GOTO
Syntax: GOTO <expr>
Description: Go to the label given by the expression (Go to the line with <expr> as a label and continue running the program from that line number.
Example:
```
        let a = 0
        Beginning:
if a = 10 then goto Ending
        let a = a + 1
        set PortB.3
set PortB.3
        goto Beginning
        Ending:
```

This program segment will toggle port PortB.3 10 times and then execute the goto statement to end.

IF… THEN
Syntax: IF <expr> <condition> <expr> THEN statement

Where: <condition> is = (equal), < (less than), > (greater than) or <> (not equal)
Description: This statement allows a program to change it's behavior based upon some criteria. If the condition is met, then the statement will be executed, otherwise the program will continue without executing the conditional statement.
Example:

```
        let a = 0
        Begin:
if a = 10 then goto Ending
        let a = a + 1
        set PortB.3
set PortB.3
        goto Begin
        Ending:
```

This program segment will toggle port PortB.3 10 times and then end.

LET
Syntax: {LET} <var> = <expr>
Description: Set the variable <var> to the value of the expression <expr>.
Example:

```
let a = 23
b = 35
let c = a + b
```

This program segment will take the number 23 and store it in the memory set aside for the variable A and then take the number 35 and store it in the memory set aside for the variable B. It will then take the two numbers from those memory locations, add them together and store that sum in the memory set aside for variable C. Note that the actual command "LET" is not actually needed, however it does serve to make the program a little bit easier to read.

NEXT
Syntax: NEXT <var>
Description: See the For…Next entry for this description.

REM (')
Syntax: REM <comment>
Description: The rest of the line in the program is considered part of the comment and is otherwise ignored. This is useful for documenting what you are doing for each line in your program.
Example:

```
         set PortB.0                              ' Turn right light on
```

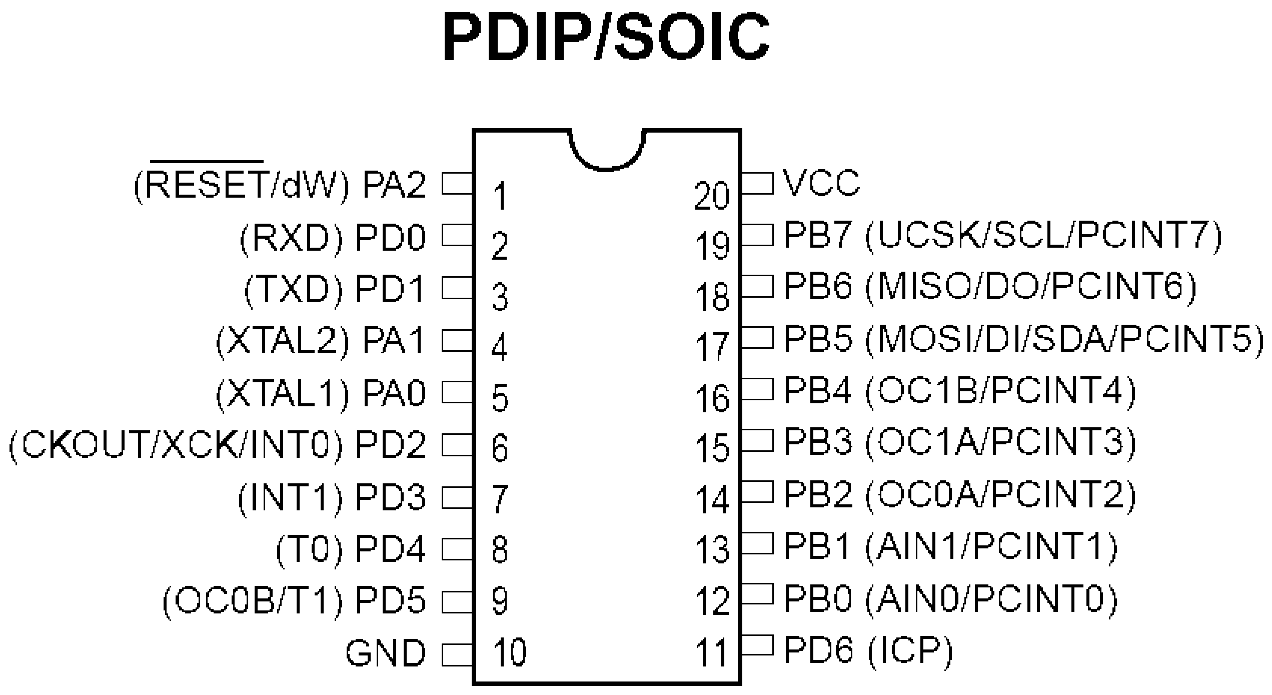
RESET
Syntax: RESET <expr>
Description: This sets the specified output pin to the ground (0V) level.
Example:
```
Reset PinB.0
```
This statement will set the output voltage on Pin B.0 to Ground.

SET
Syntax: SET <expr>
Description: This sets the specified output pin to a +V level.
Example:
```
Set PinB.0
```
This statement will set the output voltage on Pin B.0 to +5V.

WAIT
Syntax: Wait <expr>
Description: The program is paused for the number of seconds (sec) specified by the value given.
Example:
```
for x = 1 to 10
  Set PortB.0
  Wait 2
  Reset PortB.0
  Wait 2
Next x
```
This program will set Pin PB0 on your Tiny2313 Experimenter System to a high level (5V) for two seconds and then set that pin to a low level (0V) for another two seconds. It will then repeat this ten times. This will flash an LED attached to that pin (and then through a current-limiting resistor and to ground) on and off about once per four second for about 40 seconds.

WAITMS
Syntax: Waitms <expr>
Description: The program is paused for the number of milliseconds (ms) specified by the value given. The maximum pause time is 65535 ms.
Example:
```
for x = 1 to 10
  Set PortB.0
  Waitms 500
  Reset PortB.0
  Waitms 500
Next x
```
This program will set Pin PB0 on your Tiny2313 Experimenter System to a high level (5V) for 500 milliseconds (or half a second) and then set that pin to a low level (0V) for another 500 milliseconds. It will then repeat this ten times. This will flash an LED attached to that pin (and then through a current-limiting resistor and to ground) on and off about once per second for about ten seconds.

Appendix B - Tiny2313 Pinout and Information



Appendix C – International Morse Code

A	.-
B	-...
C	-.-.
D	-..
E	.
F	..-.
G	--.
H	....
I	..
J	.---
K	-.-
L	.-..
M	--
N	-.
O	---
P	.--.
Q	--.-
R	.-.
S	...
T	-
U	..-
V	...-
W	.--
X	-..-
Y	-.--
Z	--..
1	.----
2	..---
3	...--
4	....-
5	.....
6	-....
7	--...
8	---..
9	----.
0	-----

Appendix D – Answers to Quizzes

Quiz 1

1) A microprocessor combines which parts, of a computer, onto a single IC? ***C***
   - A) Input and Output
   - B) RAM and ROM
   - C) Control and ALU
   - D) CPU and Memory

2) A control computer is a computer which is designed to do just about any general task, such as balancing your checkbook, writing letters or browsing the internet. (true or false) ***False***

3) The microcontroller on your Tiny2313 Experimenter System is: ***B***
   - A) The Atmel AVR-ATtiny13
   - B) The Atmel AVR-ATtiny2313A
   - C) The Motorola MC68HC11
   - D) The Texas Instruments MSP-430

4) A fan would be connected to which major unit of a computer? ***D***
   - A) Input
   - B) Control
   - C) Memory
   - D) Output

5) By connecting switches to the _____ lines on a control computer, the computer can tell which switch has been closed and which has not. ***B***
   - A) Memory
   - B) Input
   - C) Output
   - D) Control

Extra Credit:

How many pins on the Tiny2313 chip can be used for Input/Output, without disabling the reset? ***B***
   - A) 10
   - B) 17
   - C) 23
   - D) All of them

Quiz 2

1) What is the native language of computers? ***A***
   - A) Electrical signals of "high" or "low"
   - B) BASIC
   - C) English
   - D) Swahili

2) The process of translating a more human-like language into a computer's language is called? ***D***
   - A) Hacking
   - B) Transliteration
   - C) Translation
   - D) Compiling

3) The BASIC programming language was designed to help computer science students with extremely complex programming tasks, and is not suitable for a beginner. (true or false) ***False***

4) What is a computer program? ***C***
   - A) A list of instructions to tell you how to use the computer
   - B) A device which translates the computer's instructions from a human-like language
   - C) A set of directions that you give the computer to tell it how to perform a task
   - D) A combination of the Control Unit and the Arithmetic/Logic Unit

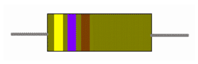
5) What is the main purpose of an ISP Downloader Cable? ***B***
   - A) To provide power to the microcontroller
   - B) To provide a way to transfer the program from the host computer to the microcontroller
   - C) So that the microcontroller can verify that it is running a legal copy of the program
   - D) To provide a physical connection between your development system and the microcontroller

Extra Credit:

The BASIC language was created by two: ***A***
   - A) Professors at Dartmouth
   - B) Engineers at Cornell
   - C) Accountants with the Federal Government
   - D) Math students in a Doctoral program

Quiz 3

1) What does a resistor look like? ***A***

A) 

B)

C)

D)

2) What input voltage will work best for your Tiny2313 Experimenter System? ***C***
   - A) 1.0V DC
   - B) 1.5V DC
   - C) 5.0V DC
   - D) 9.0V DC

3) The diode on your Experimenter System is heat sensitive, and care must be taken to not overheat the device while soldering it to the circuit board. (true or false) ***True***

4) For protection from static electricity, the '2313 chip is stored in ***D***
   - A) Styrofoam
   - B) A plastic bag
   - C) Shaving foam
   - D) Anti-static foam

5) What are the two very important things to remember when soldering? ***B***
   - A) Heat up, Cool down
   - B) Mechanical connection, Electrical connection
   - C) Touch the conductors, Not too long
   - D) Component leads, PCB pads

Extra Credit:

Briefly describe the purpose and function of a printed circuit board (PCB) ***Answer must contain the following two points: To hold the components in place; and to provide a path for current flow.***

Quiz 4

1) How does an LED light? ***A***

A) Electrical current flows from the cathode through to the anode and on to a positive source.

B) Electrical current flows from the cathode through to the anode and on to a negative source.

C) Electrical current flows from the anode through to the cathode and on to a positive source.

D) Electrical current flows from the anode through to the cathode and on to a negative source.

2) What statement will cause an LED, with the anode connected to an output pin, to light? ***B***

A) `output`

B) `set`

C) `reset`

D) `high`

3) For safety, you need to make sure that power is applied to your Experimenter System before making any changes to the circuitry. (true or false) ***True***

4) How long will the program delay with a `waitms 10000` statement? ***C***

A) 10000 seconds

B) 10 milliseconds

C) 10 seconds

D) 100 milliseconds

5) Why were you not able to see the LED flash in your first attempt to flash the LED? ***D***

A) The statements were not placed in a loop

B) The `set` statement was used incorrectly

C) The `reset` statement was used incorrectly

D) The LED was turned off too quickly to see

Extra Credit:

What is the function of the two connectors labeled Servo1 and Servo2? ***To connect servo motors, or external sensor devices.***

Quiz 5

1) What is the programming tool that allows the controller to repeat program steps? ***B***
   - A) A do loop
   - B) A loop
   - C) A `for…next` statement pair
   - D) A while…wend loop

2) What is the purpose of the `goto` statement? ***D***
   - A) To create a program flow
   - B) To count iterations of a loop
   - C) To serve as a program comment
   - D) To change the normal top-to-bottom flow of a running program

3) In the microcontroller world, a *Hello World* program outputs a line of text saying "Hello World." (true or false) ***False***

4) What does a colon at the end of a statement mean? ***A***
   - A) It marks a label
   - D) It is an immediate command
   - C) It marks the end of a statement line
   - D) It is used to separate statements on a multi-statement line

5) What is the shortcut key to compile a program? ***A***
   - A) F7
   - B) F6
   - C) F5
   - D) F4

Extra Credit:

In the normal configuration, does an output pin, on the Tiny2313 Experimenter System sink, or source, current when connected to one of the on-board LEDs? ***Source***

Quiz 6

1) What is the purpose of the empty do loop (`Do…Loop Until Pind.5 = 0`)? ***<u>B</u>***
    - A) Causes the '2313 to count up rapidly
    - B) Pauses program execution until the voltage on Port D.5 goes low
    - C) Pauses program execution until the voltage on Port D.5 goes high
    - D) It is a simple delay loop

2) How does one of our pushbutton switches work? ***<u>A</u>***
    - A) Two pieces of metal are brought into contact when the button is pressed
    - B) FM (Foul Magic)
    - C) Two pieces of metal are in contact with each other until the button is pressed
    - D) The button measures change in capacitance when a human's finger touches it

3) With the Tiny2313 Experimenter System wired up as we currently have it, Pin D.5 will "see" a low voltage (logic 0,) when you push the button. (true or false) ***<u>True</u>***

4) How does the input pin read a logic 1 when the pin is not connected to ground? ***<u>D</u>***
    - A) A floating input is always read as a high
    - B) Current flows out of the input pin and goes to the positive terminal of the battery
    - C) Through the pull-down resistor network
    - D) With no connection to ground, the input pin senses a high through the pull-up resistor

5) In our railroad-crossing scenario, how many sensors would you need? ***<u>A</u>***
    - A) Enough that the magnets on the train are continuously activating at least one
    - B) One, at the street
    - C) At least 2, one for each direction
    - D) Four

Extra Credit:

Describe a different, real-world situation, and how you would set up a simulation of that on your Tiny2313 Experimenter System. Write a rough, first draft of the main part of your control program for your system.

<u>There are an infinite number of correct answers here. The important part is that the program contain a loop to continuously control the system, that the program inputs and outputs correspond to the description, that the program read the inputs and activate the outputs.</u>

Quiz 7

1) What is a main advantage of a `for…next` loop? ***D***
    - A) It is faster
    - B) It takes less program steps
    - C) It uses less memory
    - D) It maintains a count of each loop

2) In what mode would you place the Port pin for a speaker? ***B***
    - A) Input
    - B) Output
    - C) Bi-directional
    - D) Tri-state

3) A `waitms` parameter of 10000 is equal to a 1 second delay. (true or false) ***False***

4) Which statement would you change to make an LED attached to Port B.7 stay on longer? ***B***
    - A) `Set PortB.7`
    - B) `Waitms 250`
    - C) `Reset PortB.7`
    - D) `Loop`

5) If you use a standard Do loop for the speaker, how long will the speaker sound? ***D***
    - A) One second
    - B) Ten seconds
    - C) 100 milliseconds
    - D) For ever

Extra Credit:

Scenario: You have a computer-controlled incubator for baby chicks. The heater is connected to a relay connected between an AVR Tiny2313A pin B.0 and ground. The thermostat is connected between pin D.0 and ground, and is set to 101.5 °F (this is not a forced air incubator.) Write a program statement that will turn on the heater after your Tiny2313 detects that the temperature is below the set temperature.

Set PortB.0

Appendix E – Example Programs

These programs have been copied and pasted directly into the BASCOM compiler and have compiled with no errors. If you have troubles with your program, compare it to the example, correct your program and try to compile again.

Example 1: Program Template:

```
' Title:
' Author: Art Granzeier, Granzeier Consulting
' Date:
' Description:

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

' Main Program

End
```

Example 2: LED Test 01

```
' Title: LED Test 01
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Turning an LED on

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output

' Main Program

set PortB.0

End
```

Example 3: LED Test 02

```
' Title: LED Test 02
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Turning an LED off

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output

' Main Program

reset PortB.0                      ‘ Turn LED off

End
```

Example 4: LED Flash 01

```
' Title: LED Flash 01
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Turning an LED on and then off

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output

' Main Program

set PortB.0                        ` Turn LED off
reset PortB.0                      ` Turn LED off

End
```

Example 5: LED Flash 02

```
' Title: LED Flash 02
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Turning an LED on and then off

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output

' Main Program

set PortB.0                        ‘ Turn LED off
wait 1                             ‘ Pause for 1 second
reset PortB.0                      ‘ Turn LED off

End
```

Example 6: LED Flash 03

```
' Title: LED Flash 03
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Turning an LED on and off repeating

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output

' Main Program
Start:
set PortB.0                          ` Turn LED on
waitms 1000                          ` Pause for 1 second
reset PortB.0                        ` Turn LED off
goto Start

End
```

Example 7: LED Flash 04

```
' Title: LED Flash 04
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Turning an LED on and off repeating

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output

' Main Program
Start:
set PortB.0                       ‘ Turn LED on
waitms 1000                       ‘ Pause for 1 second
reset PortB.0                     ‘ Turn LED off
waitms 1000                       ‘ Pause for 1 second
goto Start

End
```

Example 8: LED Flash 05

```
' Title: LED Flash 05
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Radiator Springs Traffic Light

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output

' Main Program
Start:
‘First flash
set PortB.0
waitms 1000
reset PortB.0
waitms 1000
‘ Second flash
set PortB.0
waitms 1000
reset PortB.0
waitms 1000
‘ Third flash – 10% longer
set PortB.0
waitms 1100
reset PortB.0
waitms 1000
goto Start

End
```

Example 9: LED Flash 06

```
' Title: LED Flash 06
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Emergency SOS Beacon

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output
```

```
' Main
Program
` First
     set
PortB.0
     waitms
100
     reset
PortB.0
     waitms
100
     set
PortB.0
     waitms
100
     reset
PortB.0
     waitms
100
     set
PortB.0
     waitms
100
     reset
PortB.0
     waitms
300
     ` Second
     set
PortB.0
     waitms
300
     reset
PortB.0
     waitms
100
     set
PortB.0
```
(continues on next column ▯)

```
     waitms
300
     reset
PortB.0
     waitms
100
     set
PortB.0
     waitms
300
     reset
PortB.0
     waitms
300
     ` Third
     set
PortB.0
     waitms
100
     reset
PortB.0
     waitms
100
     set
PortB.0
     waitms
100
     reset
PortB.0
     waitms
100
     set
PortB.0
     waitms
100
     reset
PortB.0
     waitms
300

     End
```

Example 10: LED Flash 07

```
' Title: LED Flash 07
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Test LED in current sink mode

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output

' Main Program
‘ Flash LED 2
     set PortB.1
     wait 2
     reset PortB.1
     wait 2

     End
```

Example 11: LED Flash 08

```
' Title: LED Flash 08
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Test two LEDs

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output

' Main Program
     Do                                ` Introducing the Do Loop
       set PortB.0
       reset PortB.1
       waitms 500
       reset PortB.0
       set PortB.1
       waitms 500
     Loop

     End
```

Example 12: Button Test

```
' Title: Button Test 01
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Experiment with detecting a button press

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output
Config PortD = Input
     Set PinD.5                        ‘ Activate Pull-up Resistor

' Main Program

Do                                ‘ Wait for button
  If Pind.5 = 0 Then Goto Start
Loop

Start:

Do                                ‘ Flash LEDs
       set PortB.0
       reset PortB.1
       waitms 500
       reset PortB.0
       set PortB.1
       waitms 500
Loop

End
```

Example 13: Button Test 01

```
' Title: Button Test 01
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Experiment with detecting a button press

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output
Config PortD = Input
Set PinD.5                        ‛ Activate Pull-up Resistor

' Main Program

Do                                ‛ Wait for button
  If Pind.5 = 0 Then Goto Start
Loop

Start:

Do                                ‛ Flash LEDs
       set PortB.0
       reset PortB.1
       waitms 500
       reset PortB.0
       set PortB.1
       waitms 500
Loop

End
```

Example 14: Button Test 02

```
' Title: Button Test 02
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Experiment with detecting a button press

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output
Config PortD = Input
     Set PinD.5                         ‘ Activate Pull-up Resistor

' Main Program

Do                                 ‘ Also waits for button
Loop Until Pind.5 = 0

Start:

Do                                 ‘ Flash LEDs
       set PortB.0
       reset PortB.1
       waitms 500
       reset PortB.0
       set PortB.1
       waitms 500
Loop

End
```

Example 15: Railroad Crossing Test 01

```
' Title: Railroad Crossing Test 01
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Railroad Crossing Test

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output
Config PortD = Input
     Set PinD.5                         ` Activate Pull-up Resistor

' Main Program

Wait:
     reset PortB.0                      ` Ensure both LEDs are off
     reset PortB.1

Do                                 ` Wait for button
Loop Until Pind.5 = 0

Start:
Do                                 ` Flash LEDs while button
       set PortB.0
       reset PortB.1
       if Pind.5 = 1 then Goto Wait
  waitms 500
       reset PortB.0
       set PortB.1
       if Pind.5 = 1 then Goto Wait
       waitms 500
Loop

End
```

Example 16: Sound Test 01

```
' Title: Sound Test 01
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Experimenting with sound output

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output

' Main Program

Do                                  ` Toggle speaker line
       set PortB.4
       waitms 1
       reset PortB.4
       waitms 1
Loop

End
```

Example 17: Sound Test 02

```
' Title: Sound Test 02
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Experimenting with timed sound output

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output
     Dim cntr as byte

' Main Program

     For cntr = 1 to 100          ' Toggle speaker line 100 times
       set PortB.4
       waitms 1
       reset PortB.4
       waitms 1
     Next cntr

     End
```

Example 18: Railroad Crossing 02

```
' Title: Railroad Crossing 02
' Author: Art Granzeier, Granzeier Consulting
' Date: 13 Oct 13
' Description: Testing railroad crossing with sound

' Configuration Section
$regfile = "ATtiny2313a.dat" ' Specify the micro
$crystal = 1000000 ' Frequency for internal RC clock
$hwstack = 32 ' Default - Use 32 for the hardware stack
$swstack = 10 ' Default - Use 10 for the SW stack
$framesize = 40 ' Default - Use 40 for the frame space

Config PortB = Output
Config PortD = Input
      Set PinD.5                              ‘ Activate Pull-up Resistor
      Dim cntr as byte

' Main Program

Wait:                                   ‘ Wait until train arrives
      reset PortB.0                           ‘ Ensure both LEDs are off
      reset PortB.1
Do
Loop Until Pind.5 = 0

Start:
Do
        set PortB.0                           ‘ Turn right light on
        reset PortB.1                         ‘ Turn left light off
        if Pind.5 = 1 then Goto Wait          ‘ If train is gone, restart
        For cntr = 1 to 100             ‘ Play tone for 200 ms
          set PortB.4
          waitms 1
          reset PortB.4
          waitms 1
        Next cntr
  waitms 300                            ‘ Pause for 300 ms
        reset PortB.0                         ‘ Turn right LED off
        set PortB.1                           ‘ Turn left LED on
        if Pind.5 = 1 then Goto Wait          ‘ If train is gone, restart
        For cntr = 1 to 100             ‘ Play tone for 200 ms
          set PortB.4
          waitms 1
          reset PortB.4
          waitms 1
        Next cntr
  waitms 300                            ‘ Pause for 300 ms
Loop

End
```