

Very Tiny Development

Dale Wheat · 13 January 2011

Introduction

The Very Tiny Development class is an introduction to *embedded development* using Atmel's ATtiny13A. It's a modest device with modest capabilities, all for a modest price. The development tools are modestly priced. The software tools used in this class are free.

The goal of this class is to introduce you to some of the tools and methods that can be used to do limited-scaled embedded applications. Both hardware and software design and development topics will be discussed.

The class is expected to span about two (2) hours, but can be expanded or condensed as circumstances dictate. Required lab apparatus is listed in the Appendix.

Two Kinds of People

There are two kinds of people in the world: Those who divide the world into two kinds of people, and those who don't.

We will be dividing the embedded development world into two distinct and unequal halves: *tools* and *targets*. Tools are the devices and stratagems we use to make or alter our little embedded systems. Targets are the actual embedded systems that we will be creating. As you can imagine, there can be a little or a lot of overlap between these two half-worlds.

Within each half-world (tools or targets) we further divide things more distinctly into Hardware and Software. Again, there is a certain amount of blur and overlap between these areas.

The Hardware

Target Hardware

This one is easy and has already been identified: the Atmel ATtiny13A, a member of the Atmel ATtiny family of microcontrollers. More information can be obtained from the Atmel web site, <http://www.atmel.com>. Specifically, the device datasheet and related application notes are required for any work on the ATtiny13A.

The ATtiny13A and whatever other circuitry you want to add to it comprise our *target hardware*. The ATtiny13A is a completely self-contained microcontroller IC, incorporating an 8-bit RISC processor, program memories, input & output devices, clocks and interconnect logic.

Hardware Tools

For this class, we demonstrate a method of quickly building a functional microcontroller circuit on a solderless breadboard. Using a combination of adapters, the microcontroller can remain in the test circuit and be programmed and re-programmed at will. This eliminates the tedious process of "burning" a chip with candidate firmware and then inserting it into a test circuit, only to be removed, erased, and re-programmed after code changes are indicated. This *rapid prototyping* technique has been employed by the author for several years with much success.

The hardware tools also include a host PC for application and firmware development, along with the appropriate *device programmer* adapter and required software. Additional laboratory test equipment is nice to have but not strictly required for the limited goals of this class.

The Software

Target Software

The target software will consist of the code you write or adapt that is actually programmed into the target hardware. Elementary examples will be provided as a starting point for future projects.

The C language will be used in the examples in this class. This allows a fairly high-level of abstraction to be used in describing your bidding for your newly minted minion.

Software Tools

A comprehensive development environment is available for the AVR family of microcontrollers. The officially supported tool is **AVR Studio** and is freely downloadable from the Atmel web site. Registration is required. The source code is proprietary and works with Microsoft Windows only.

Atmel supplies only an assembler with AVR Studio. For C programming, **WinAVR**, the Windows port of the AVR port of the GNU Compiler Collection will be used. WinAVR also contains a collection of utilities that may come in handy for AVR development.

Software development tools are available that work with other operating systems, such as Linux. Ubuntu 10.10 is used as an example in the class.

Laboratory Preparation

The development environment must be set up before the development cycle begins. Install the required software on your host PC in the order indicated.

Windows Setup

Download **WinAVR** from SourceForge:

http://sourceforge.net/project/showfiles.php?group_id=68108

At the time of this writing, the current version is "20100110", aka 10 January 2010. A self-extracting executable installation program is provided:

WinAVR-20100110-install.exe.

Launch this program to install WinAVR on your Windows computer.

Note: You do not need to install "Programmer's Notepad", but you can if you like.

Note: You need to allow the installation program to modify your system PATH.

A system restart is required at this point.

Once WinAVR has been successfully installed, download and install **AVR Studio** from the Atmel

http://www.atmel.com/dyn/products/tools_card_v2.asp?tool_id=2725

At the time of this writing, the current version of AVR Studio is 4.18 (build 684). AVR Studio is provided as a self-extracting installation program:

AvrStudio4Setup.exe

Start the installation by executing the AvrStudio4Setup.exe program. You will also need the latest "service pack". Download AVR Studio SP3 (build 716). Apply the service pack by running the installation application (AVRStudio4.18SP3.exe).

Note: These are large downloads. Allow plenty of time to get them downloaded and installed.

The AVR software development environment for Microsoft Windows is now complete.

Ubuntu Setup

Using the Synaptic Package Manager, install the following packages:

```
avrdude
avrdude-doc (optional, recommended)
binutils-avr
avr-libc
gcc-avr
```

An easy way to find all these packages is to do a "Quick search" on the term "avr". This shortens the list considerably.

Target Hardware Setup

Note: The basic target hardware setup is independent of operating system choice.

Using jumper wires, connect the left and right power rails of the breadboard together.

Install the breadboard ISP adapter into your solderless breadboard, with the ISP connector extending past the end of the breadboard.

Install a jumper from ATtiny13A pin 4 to the blue ground rail.

Install a jumper from ATtiny13A pin 8 to the red power rail.

Connect a "Power On" LED indicator across the power rails. Include a current-limiting resistor in series.

Connect the 10 pin, flat ribbon cable from the breadboard ISP adapter to the USB ISP device programmer.

Connect the USB ISP device programmer to your PC. Check that the "Power On" indicator lights.

Note: On Windows systems, ensure that the system recognizes an "AVR ISP mkII".

Disconnect the USB cable to remove power to the breadboard. This tends to be easier than removing the 10 pin, flat ribbon cable.

With the power disconnected, insert the ATtiny13A device into the socket on top of the breadboard adapter. Note the correct orientation of the device in the socket.

Install another LED from one of the available IO pins on the ATtiny13A through a current-limiting resistor to ground. Pin 1 is reserved for RESET. Pin 4 is ground and pin 8 is power. All the other pins are available as inputs or outputs.

Re-attach the USB cable. Now you are ready to program your ATtiny13A.

Tiny Development Under Windows

Launch AVR Studio. When asked, create a new project.

The New Project Wizard

For "Project type:", select "AVR GCC". If this is not an option, you most likely did not properly install WinAVR in a previous step, or there was a problem encountered during its installation. This must be corrected before proceeding.

Enter a "Project name:". Do not use spaces in the project name. Elect to have AVR Studio create the initial file and the project folder. Choose a convenient location (i.e., parent directory) for the project.

On the next page of the Project Wizard, you select a debug platform and specify a device. We will be using debugging or emulation hardware in this class. Select the "AVR Simulator 2" as the "Debug platform:". On the list on the right, scroll down and select the "ATtiny13A" device.

Click on the "Finish" button to complete the Project Wizard. This will open your new project in the "integrated development environment", or IDE of AVR Studio. At the top are menus and icons.

left is a tree-view of your project and its associated files. On the right and occupying the majority of the screen is the text-editing area. Your new source code file, <project_name>.c, should be open in the text-editing area. Click the "maximize" button in the upper right-hand corner of the text-editing box. At the bottom of the screen is the area for build results and other messages.

The Source Code

The first project is to blink an LED. You need the following code in your source file:

```
#include <avr/io.h>

int main(void) {

    int i;

    DDRB = 0x1F; // all pins are outputs

    while(1) {
        PORTB = 0x1F; // all LEDs on
        for(i = 30000; i > 0; i--); // short delay
        PORTB = 0x00; // all LEDs off
        for(i = 30000; i > 0; i--); // another short delay
    }

    return 0; // this never happens
}
```

Listing 1. Blink an LED in C.

Once you've entered all the source code into the file, be sure to save your work. You can use the "File/Save" menu item, press Ctrl-S or click on the "Save" icon, which looks like a floppy disk. Then save your source code file using the project name you supplied with the ".c" extension, in the project directory.

Project Configuration

Click on the "Edit Current Configuration Options". This should be the last icon on the toolbar and looks like a gear. This opens the "<project_name> Project Options" dialog box. We need to make one change at this time. "Optimization:" defaults to "-Os", which is good, and tells the compiler to optimize primarily for space and secondarily for speed. Unfortunately, it will "optimize" our naive delay loop completely out of the program, which is not so good, at least in this simple example. Change the "Optimization:" setting to "-O0" [dash, capital letter "O", numeral zero]. This disables compiler optimization. Click the "OK" button to save this change.

Build the Project

Now we build the project. Select the "Build/Build" menu item, press F7 or click on the "Build Action" icon. If all goes well, you should get the message:

"Build succeeded with 0 Warnings..."

If not, scroll back up in the build results area to find the specific error message describing the problem. Correct the problem and try building the project until it completes successfully.

"Building" the project consists of compiling and linking the source code and any libraries, as well as other processes you specify in the "Project Options" dialog box. Once complete, you should have a binary that is ready to be programmed into the chip and executed.

Device Programming

Make sure your device programmer is attached to your PC via the USB cable. Also make sure the device programmer is attached to the target board, and that the "Power On" LED indicator is on.

Now either select the "Tools/Program AVR/Auto Connect" menu item or click the "Connect to the Selected AVR Programmer" icon, which looks like a small IC with the letters "AVR" on it.

If the "Select AVR Programmer" dialog box appears, you must indicate which device programmer you have and what port it is using. Select "AVRISP mkII" and "USB", then click on the "Connect..." button.

You should now see the "AVR ISP mkII in ISP mode with <device>" dialog box. The <device> in the last one used. If not, the "Select AVR Programmer" dialog will re-appear. This means that AVR Studio was unable to communicate with your device programmer. This error must be corrected before you can proceed.

The "AVRISP mkII in ISP mode with <device>" dialog box is a busy one. It has eight tabs at the top. The first one is labeled "Main" and contains information about the target device and available programming modes.

In the "Device and Signature Bytes" group, use the drop-down combo box to select the "ATtiny1617". Now click on the "Read Signature" button. The text box below the device selector should change from "Signature not read" to "0x1E 0x90 0x07" and a message below that should interpret the device signature and report, "Signature matches selected device". This confirms that AVR Studio was able to talk to the device programmer, query the "device signature", a three byte code stored in all Atmel AVR devices, and that the device programmer was able to read that information from the target chip. This confirms the proper operation of your device programmer and its connection to your target device.

Click on the "Program" tab of the dialog box. Here you can program the device memory. In the "Program File" group, use the browse button "..." to find your project folder. Within your project folder is a folder created by AVR Studio called "default". Within that folder is the file with which we want to program your chip. It will be named <project_name>.hex, bearing the name you selected for the project and the ".hex" file extension. The file is in Intel HEX format. Select the file and click on the "Open" button.

Once the program file has been selected, click the "Program" button. This starts the process of downloading the program to the chip. It should only take a second or two to complete the process.

process includes erasing the device beforehand and verifying the contents afterwards, if those options are checked.

Minimize the "AVRISP mkII in ISP mode with ATtiny13A" dialog box. This enables the "Write Flash Memory Using Current Settings" icon, which looks like a chip with a red arrow in it. Clicking this icon is the same as pressing the "Program" button on the previous dialog box.

This completes one iteration of the development cycle. Go back and modify the code to change the delay times. Clicking the "Build Active Configuration" icon saves the current file and performs a build. If no errors occur, you can then click the "Write Flash Memory Using Current Settings" icon to send the new code to the chip. Test and repeat.

Tiny Development Under Ubuntu

Create a new folder in your home directory. Name it anything you want. If you are using the Nautilus file browser, navigate to your new project folder and right click in the file area and select "Create Document/Empty File". Replace the text "new file" with your source code filename, e.g., "blink.c" and press [Enter]. Right click the newly created file icon and select "Open With Text Editor" and enter the source code:

```
#include <avr/io.h>

int main(void) {

    int i;

    DDRB = 0x1F; // all pins are outputs

    while(1) {
        PORTB = 0x1F; // all LEDs on
        for(i = 30000; i > 0; i--); // short delay
        PORTB = 0x00; // all LEDs off
        for(i = 30000; i > 0; i--); // another short delay
    }

    return 0; // this never happens
}
```

Listing 2. Blink an LED in C. It's the same as for the Windows version.

Note: The default text editor in Ubuntu 10.10 is gedit, which does a fine job as a program editor. You'll notice that it knows your file is a C language source file (from the file extension) and highlights the syntax accordingly. One suggestion is to change the default tab stop from 8 spaces to 3 or 4. Select the "Edit/Preferences" menu item and select the "Editor" tab. Adjust

the tab width to your liking. "Automatic indentation" is another nice feature that you might want to enable.

Once you've entered all the source code, save your work by clicking the "Save" icon or selecting "File/Save" menu item or pressing Ctrl-S.

Open a terminal window ("Applications/Accessories/Terminal" menu item) and navigate to your project folder. Now we compile your program from the command line:

```
avr-gcc -mmcu=attiny13a blink.c -o blink.o
```

This compiles the program in blink.c for the AVR device "attiny13a" (all lower case) and puts the resulting output into an "object file" called blink.o. You can omit the "-o blink.o" and the compiler default to an object file called a.out.

Now we convert the object file to a HEX file that can be sent to the chip, using this command:

```
avr-objcopy -j .text -j .data -O ihex blink.o blink.hex
```

Lastly, we program the chip with the following command:

```
avrdude -p attiny13 -c avrispmkii -P usb -U flash:w:blink.hex
```

Note: If you lack the proper authorizations on your system, you may have to prepend the command with "sudo" and supply your administrator's password.

That completes a single iteration of the development cycle. Now go back and change the code. rinse, repeat. Always repeat.

Appendix

Required Laboratory Apparatus

- Relatively modern personal computer with spare USB port (Windows/Linux) and Internet access
- ATtiny13A microcontroller
- USB-ISP device programmer
- ISP breadboard adapter
- Solderless breadboard and jumper wires
- Miscellaneous electronic components