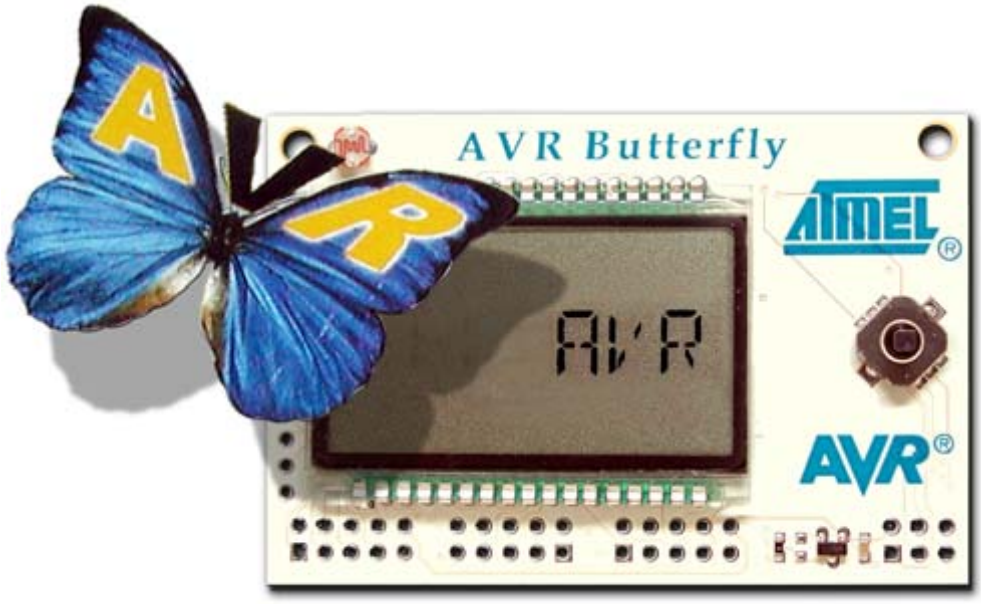
Quick Start Guide for Using the WinAVR Compiler with ATMEL's AVR Butterfly



Joe Pardue

SmileyMicros.com

The Butterfly Quick Start was extracted from C Programming for Microcontrollers – Featuring ATMEL's Butterfly and the free WinAVR compiler (available from SmileyMicros.com)

Published by Smiley Micros

Smiley Micros
5601 Timbercrest Trail
Knoxville, TN 37909
Email: book@SmileyMicros.com
Web: http://www.SmileyMicros.com

ISBN 0-9766822-0-6

Chapter 1: Introduction

Table of Contents:

Chapter 1: Introduction

C Programming and microcontrollers are two big topics, practically continental in size, and like continents, are easy to get lost in. Combining the two is a little like traipsing from Alaska to Tierra del Fuego. Chances are you'll get totally lost and if the natives don't eat you, your infected blisters will make you want to sit and pout. I've been down this road so much that I probably have my own personal rut etched in the metaphorical soil, and I can point to all the sharp rocks I've stepped on, all the branches that have whacked me in the face, and the bushes from which the predators leapt. If you get the image of a raggedy bum stumbling through the jungle, you've got me right. Consider this book a combination roadmap, guidebook, and emergency first aid kit for your journey into this fascinating, but sometimes dangerous world.

I highly recommend that you get the book, 'The C Programming Language – second edition' by Kernighan and Ritchie, here after referred to as K&R. Dennis Ritchie, Figure 1, wrote C, and his book is the definitive source on all things C.

Figure 1: Dennis Ritchie, inventor of the C programming language stands next to Ken Thompson, original inventor of Unix, designing the original Unix operating system at Bell Labs on a PDP-11

I have chosen to follow that book's organization in this book's structure. The main difference is that their book is machine independent and gives lots of examples based on manipulating text, while this book is machine dependent, specifically based on the AVR microcontroller, and the examples are as microcontroller oriented as I can make them.

Why C?

Back in the dark ages of microprocessors, software development was done exclusively in the specific assembly language of the specific device. These assembly languages were character based 'mnemonic' substitutions for the numerical machine language codes. Instead of writing something like: 0x12 0x07 0xA4 0x8F to get the device to load a value into a memory location, you could write something like: MOV 22 MYBUFFER+7. The assembler would translate that statement into the machine language for you. I've written code in machine language (as a learning experiment) and believe me when I tell you that assembly language is a major step up in productivity. But a device's assembly language is tied to the device and the way the device works. They are hard to master, and become obsolete for you the moment you change microcontroller families. They are specific purpose languages that work only on specific microprocessors. C is a general-purpose programming language that can work on any microprocessor that has a C compiler written for it. C abstracts the concepts of what a computer does and provides a text based logical and readable way to get computers to do what computers do. Once you learn C, you can move easily between microcontroller families, write software much faster, and create code that is much easier to understand and maintain.

Why AVR?

As microprocessors evolved, devices increased in complexity with new hardware and new instructions to accomplish new tasks. These microprocessors became known as CISC or Complex Instruction Set Computers. Complex is often an understatement; some of the CISCs that I've worked with have mind-numbingly complex instruction sets. Some of the devices have so many instructions that it becomes difficult to figure out the most efficient way to do anything that isn't built into the hardware.

Then somebody figured that if they designed a very simple core processor that only did a few things but did them very fast and efficiently, they could make a much cheaper and easier to program computer. Thus was born the RISC, Reduced Instruction Set Computers. The downside was that you had to write additional assembly language software to do all the things that the CISC computer had built in. For instance, instead of calling a divide instruction in a CISC device, you would have to do a series of subtractions to accomplish a division using a RISC device. This ‘disadvantage’ was offset by price and speed, and is completely irrelevant when you program with C since the complier generates the assembly code for you.

Although I’ll admit that ‘CISC versus RISC’ and ‘C versus assembly language’ arguments often seem more like religious warfare than logical discourse, I have come to believe that the AVR, a RISC device, programmed in C is the best way to microcontroller salvation (halleluiah brother).

The folks that designed the AVR as a RISC architecture and instruction set while keeping C programming language in mind. In fact they worked with C compiler designers from IAR to help them with the hardware design to help optimize it for C programming.

Since this is an introductory text I won’t go into all the detailed reasons I’ve chosen the AVR, I’ll just state that I have a lot of experience with other microcontrollers such as Intel’s 8051, Motorola’s 68xxxes, Zilog’s Z’s, and Microchip’s PIC’s and I’m done with them (unless adequately paid – hey, I’m no zealot). These devices are all good, but they require expensive development boards, expensive programming boards, and expensive software development tools (don’t believe them about the ‘free’ software, in most cases the ‘free’ is for code size or time limited versions).

The AVR is fast, cheap, in-circuit programmable, and development software can be had for FREE (really free, not crippled or limited in any way). I’ve paid thousands of dollars for development boards, programming boards, and C compilers for the other devices, but never again -- I like free. The hardware used in this text, the ATMEL Butterfly Evaluation Board can be modified with a few components to turn it into a decent development system and the Butterfly and

needed components can be had for less than $40.00 (See Appendix 1 Project Kits). You can't get a better development system for 10 times this price and you can pay 100 times this and not get as good.

Okay, maybe I am a zealot.

Goals

What I hope to accomplish is to help you learn **some** C programming on **a specific** microcontroller and provide you with enough foundation knowledge that you can go off on your own somewhat prepared to tackle the plethora (don't you just love that word, say it 10 times real quick) of microcontrollers and C programming systems that infest the planet.

Both C programming and microcontrollers are best learned while doing projects. I've tried to provide projects that are both useful and enhance the learning process, but I've got to admit that many of the early projects are pretty lame and are put in mainly to help you learn C syntax and methods.

Suggested Prerequisites:

- You should be able to use Windows applications.
- You should have an elementary knowledge of electronics, or at least be willing to study some tutorials as you go along so that you'll know things like why you need to use a resistor when you light up an LED.
- I've received lots of suggestions about what needs to be in this book. Some folks are adamant that one must first learn assembly language and microcrocontroller architecture and basic electronics and digital logic and bla bla bla before even attempting C on microcontrollers. I politely disagree and say that you should just jump right in learn whats fun for you. You'll run across lots of stuff that you will want to learn about, but I won't cover in the book so you should be able to bracket your ignorance (and mine) making a note when you hit something you don't know but would like to. Then you can learn it later. I'm using lots of things that aren't directly relevant to C programming (like communicating with a microcontroller from a PC using a serial port or like what the heck is that transistor motor driver thingee…). If you get really curious, then GOOGLE for a tutorial on the topic.

By the time you complete the text and projects you will:

- Have an intermediate understanding of the C programming language.
- Have a elementary understanding microcontroller architecture.
- Be able to use the WinAVR and AVR Studio tools to build programs.
- Be able to use C to develop microcontroller functions such as:
    - Port Inputs and Outputs
    - Read a joystick
    - Use timers
    - Program a Real Time Clock
    - Communicate with PC
    - Conduct analog to digital and digital to analog conversions
    - Measure temperature, light, and voltage
    - Control motors
    - Make music
    - Control the LCD
    - Flash LEDs like crazy

On the CD you will find the ATMEL ATMEGA169 data book. At 364 pages, it is the comprehensive source of information for the microcontroller used on the AVR Butterfly board. Open it on your PC with Adobe Acrobat and look around a bit: intimidating isn't it? But don't worry; one of the purposes of this text is to give you enough knowledge so that you can winnow the wheat from the chaff in the data book and pull out what you need for your C based control applications.

I know how easy it is to get bogged down in all the detail and lose momentum on this journey, so we'll begin with the 'Quick Start' chapter by learning only enough to make something interesting happen: kind of a jet plane ride over the territory. Then we will proceed at a comfortable pace from the simple to the complex using as interesting examples as I can come up with. I'm partial to LEDs so you are going to see a lot of flashing lights before we are through, and hopefully the lights won't be from you passing out from boredom and boinking your head on the keyboard.

Chapter 2: Quick Start Guide

The purpose of this quick start guide is to help you modify the Butterfly hardware so you can use it as a development board and to show you how to use the FREE software for writing and compiling C code and downloading it from your PC to the Butterfly.

The AVR Butterfly is an evaluation kit for the ATMEGA169 microcontroller that was custom designed with an AVR core and peripherals to make it both a general-purpose microcontroller and an LCD controller. This little board is by far (at this writing) the lowest cost system for learning and developing that I've ever seen. I don't know how much these things cost them to make, but Digi-Key (www.digikey.com) sells them for $19.99 (Spring 2005), which has to be a real loss leader for ATMEL (www.ATMEL.com). But their loss is our gain, and I'm sure they are happy to prime-the-pump a little, knowing that we'll get hooked on the AVR and buy lots of their product.

It is simply amazing what the Butterfly has built in:

- 100 segment LCD display
- 4 Mbit (that's 512,000 bytes!) dataflash memory
- Real Time Clock 32.768 kHz oscillator
- 4-way joystick, with center push button
- Light sensor
- Temperature sensor
- ADC voltage reading, 0-5V
- Piezo speaker for sound generation
- Header connector pads for access to peripherals
- RS-232 level converter for PC communications
- Bootloader for PC based programming without special hardware
- Pre-programmed demos with source code
- Built-in safety pin for hanging from you shirt (GEEK POWER!)
- Kitchen sink.

I mean this thing has everything (except a kitchen sink… sorry). If anyone can find a development platform with anywhere near this much for this price, I want to hear about it. And, no, I don't own stock in ATMEL, or work for them, I just

couldn't find anything that comes close to this system for my goal of teaching C programming for AVR microcontrollers (or any microcontrollers for that matter). If I seem to be raving a bit, get used to it, I do that a lot.

There are sufficient instructions on the AVR Butterfly box to show you how to use all the built-in functions. Play with it now before you risk destroying it in the next step. Don't say I didn't warn you. If you break it, you'll have to order a new one from Digi-Key (www.digikey.com). I shudder to think how many of these things will get burned up, blown up, stepped on, and drenched in coffee. And that's just me this morning.

Note: in order to save you money, rather than selling you the Butterfly and the experiments kits, you will find a parts list (Appendix 1) so that you can buy this stuff directly from the vendors. But check my website: www.smileymicros.com, no telling what you'll find. (Hopefully, not a 'going out of business' sale.)

If you purchased the e-book, you can download the WinAVR software from http://sourceforge.net/projects/winavr (this book uses version 20040404) and the AVRStudio software from the http://www.atmel.com web site. On the ATMEL website search for the AVRStudio version 4.11 (later versions may not correlate to this book). If, for some reason, these sites are not available (I can't guarantee what they'll do to their sites) look on the http://www.smileymicros.com website for updated information on how to get the software. If you purchased a hard copy of the book, you will find the software on the accompanying CD.

Don't get bogged down in all the installation choices given, just accept suggested defaults so your installation will match this book. And, as an aside, by the time you install all this software, the WinAVR and the AVRStudio will have new and improved versions available on their web sites. DON'T USE THEM! This text is based on the versions on the CD or on the SmileyMicros.com web site and using the new and improved software may only confuse things. Of course, by the time you finish this text, you will be encouraged to get the latest and greatest, by then you'll know all you need to use it wisely.

Software

We will use three FREE software packages, the WinAVR compiler from sourceforge.net, the AVRStudio 4 from ATMEL, and Br@y++'s Terminal.

WinAVR – Oh, Whenever…

WinAVR is a set of tools for C programming the AVR microcontroller family. A bunch of folks have volunteered their time to write this software and give it away as part of the free software movement (www.sourceforge.net). These folks generously giving there time to help others is almost enough to change my cynical opinion of humanity. You can spend thousands on C compilers for microcontrollers and before WinAVR you had to spend several hundred even for a crappy compiler. This software is FREE, but SourceForge has expenses so send them some money at www.sourceforge.net/donate.

At http://sourceforge.net/projects/winavr/ you see the summary:

"WinAVR (pronounced "whenever") is a suite of executable, open source software development tools for the ATMEL AVR series of RISC microprocessors hosted on the Windows platform. Includes the GNU GCC compiler for C and C++."

Go to: http://winavr.sourceforge.net/index.html and check out their homepage.

But don't get too distracted with all that yet, just use the tools as shown here, and once you reach the end of this book, then you'll have the skills to fully exploit those web sites.

Programmers Notepad

We'll be writing our software using the most excellent Programmers Notepad, another FREE program available at sourceforge.net and included in the WinAVR distribution package. Imagine what Microsoft would charge for this FREE software. Be a good guy or gal and send them some money at http://www.pnotepad.org.

AVRStudio – FREE and darn well worth it.

AVR Studio is provided free by the good folks at ATMEL Corporation, who seem to understand that the more help they give developers, the more they will sell their microcontrollers. Actually, this too could cost hundreds and still be darn well worth it, but unless you just really like Norway, don't send them any money, they'll get theirs on the backend when you start buying thousands of AVRs for your next great invention.

The AVR Studio will be used for two things: first, to download your software to the AVR Butterfly, and second, to simulate the ATMEGA169 running your software.

Br@y++ Terminal:

The original Quick Start Guide chapter used HyperTerminal, which is hard to setup, clunky, and hated by so many folks on the AVRFreaks.net forum that I contacted Br@y++ and he gave me permission to use and distribute his highly recommended and easy to use and understand terminal package. You can get it at http://bray.velenje.cx/avr/terminal or http://www.smileymicros.com. It is shown in Figure 7: Bray's Terminal. The examples in the text still show the HyperTerminal, but it shouldn't be a problem substituting Bray's. If you want to use HyperTerminal, the introduction to it is in Appendix 1.

Hardware

Constructing Your Development Platform

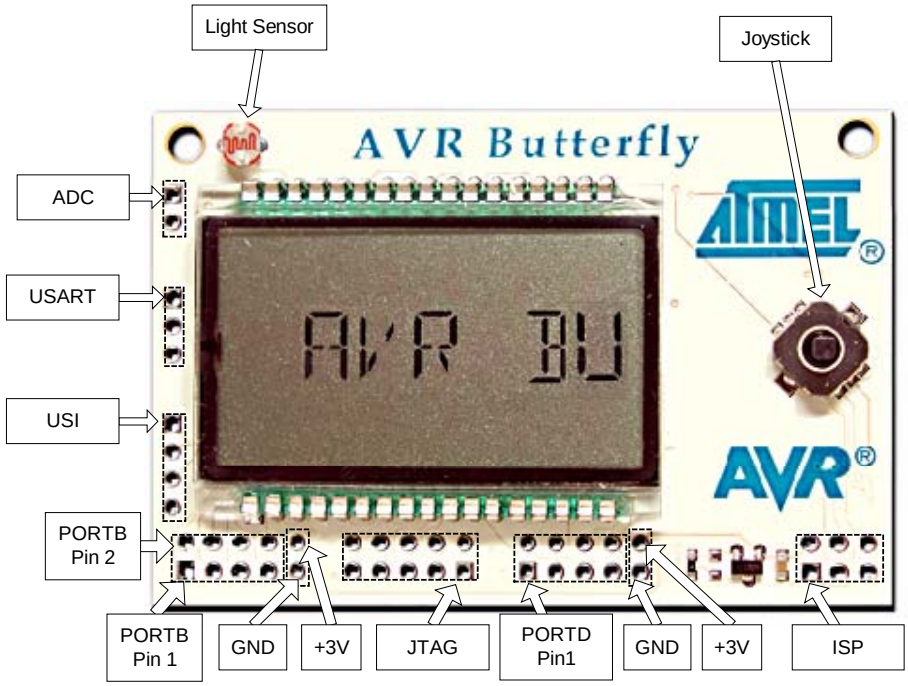


Figure 2: The Butterfly front

Solder the female headers to the ADC, PORTB, and PORTD lands. Note that the square pads are pin1 and that PORTB and PORTD seem to have 10 pins, but they don't, pins 9 and 10 are ground and power respectively (see Figure 2).

The RS-232 Connection:
Communication with the PC requires three lines: TXD, RXD, and GND. The TXD is the transmit line (data from the PC to the Butterfly), RXD is the receive line (data from the microcontroller to the PC) and GND is the common ground. Notice that there is a bit of relativity in this equation, the microcontroller's RXD wire is the PC's TXD wire and vice versa. I can't count the number of times I've

done stupid things like connecting the microcontroller's RXD pin to the DB-9 RXD pin, because I didn't think 'RXD – receive - relative to what?'

The parts list has a DB-9 female solder cup RS-232 connector. Follow the illustrations in Figure 3.

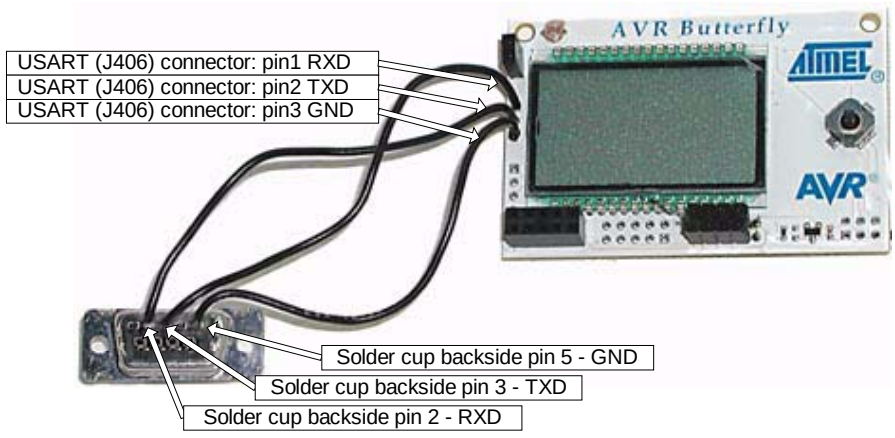


Figure 3: RS-232 connections.

NOTICE HOW THE RXD AND TXD LINES CROSS OVER – PAY CAREFUL ATTENTION AS IT IS EASY TO GET THESE REVERSED.

Constructing the power supply:

The Butterfly comes with a CR2450 coin battery that will power the LCD for a long time, but will be used up quickly by the RS-232 connection and our experiments. Remove the coin battery and construct a battery pack with parts from the JAMECO parts list (Appendix 7) using the following pictures. Be sure and get the power, red wire, and ground, black wire, correct: as shown in Figure 4 and Figure 5.

NOTE: ALL THE ILLUSTRATIONS SHOW PORTD WITH AN 8-PIN HEADER AND THE POWER WIRES SOLDERED IN PLACE. THE PARTS KIT SPECIFIES 10-PIN CONNECTORS FOR BOTH PORTS B AND D. USE THE 10-PIN HEADER ON PORTD AND INSERT RATHER THAN SOLDER THE POWER WIRES.

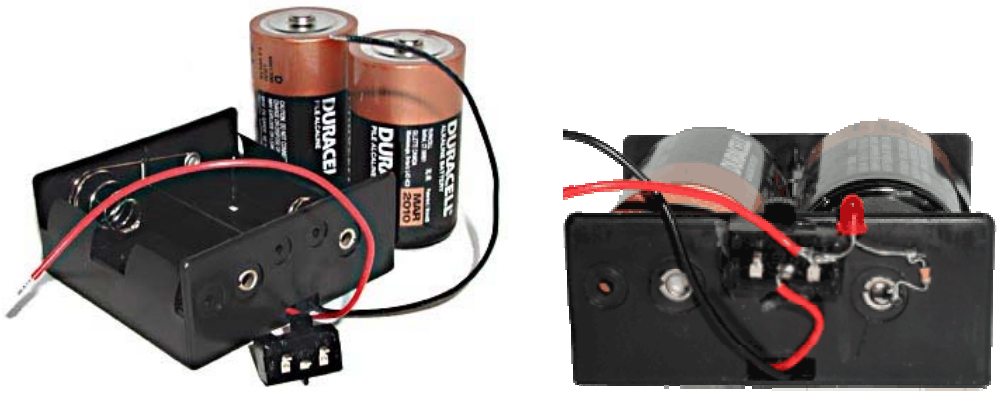


Figure 4: Battery holder, switch, and batteries.

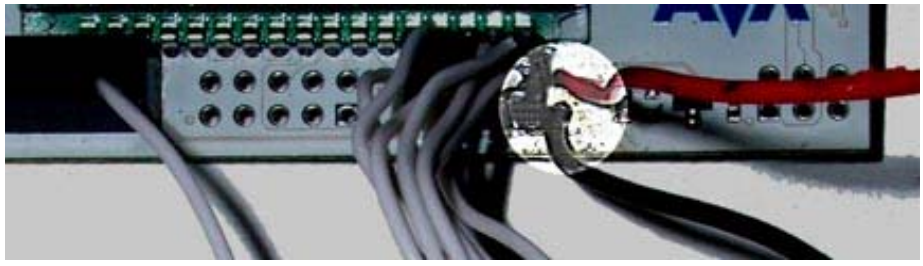


Figure 5: External battery connection to Butterfly

A few days after making the power supply I left it on all night, so I added an LED (Figure 4) to the switch so that I'd know that it was on. You can solder the long leg of an LED to the rightmost pin on the switch, where the +3v goes to the Butterfly, and then solder a 330 resistor to the short leg and the resistor to the rivet at the base of the battery on the right. The LED is lit when the switch is to +3V.

Test your Connection using Brays Terminal:

Hook your RS-232 cable to the Butterfly as in Figure 6. The run Bray's Terminal, (well, Br@y++'s to be exact – available at http://bray.velenje.cx/avr/terminal and http://www.smileymicros.com) and configure it as in Figure 7 with the radio buttons set to select your COM port, 19200 Baud rate, 8 Data bits, parity of none, 1 Stop bits, and no handshaking. Click the connect button. Turn on your Butterfly

power supply, then with the joystick button centered press it and watch the stream of ?????? question marks that should be coming from the Butterfly. This is the Bootloader telling you that it is alive and ready to be boot loaded, or perhaps it is just curious as to what's going on?

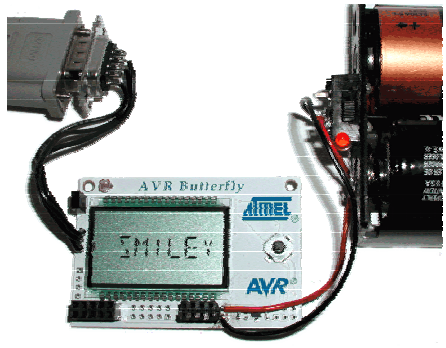


Figure 6: Butterfly hooked up to RS-232

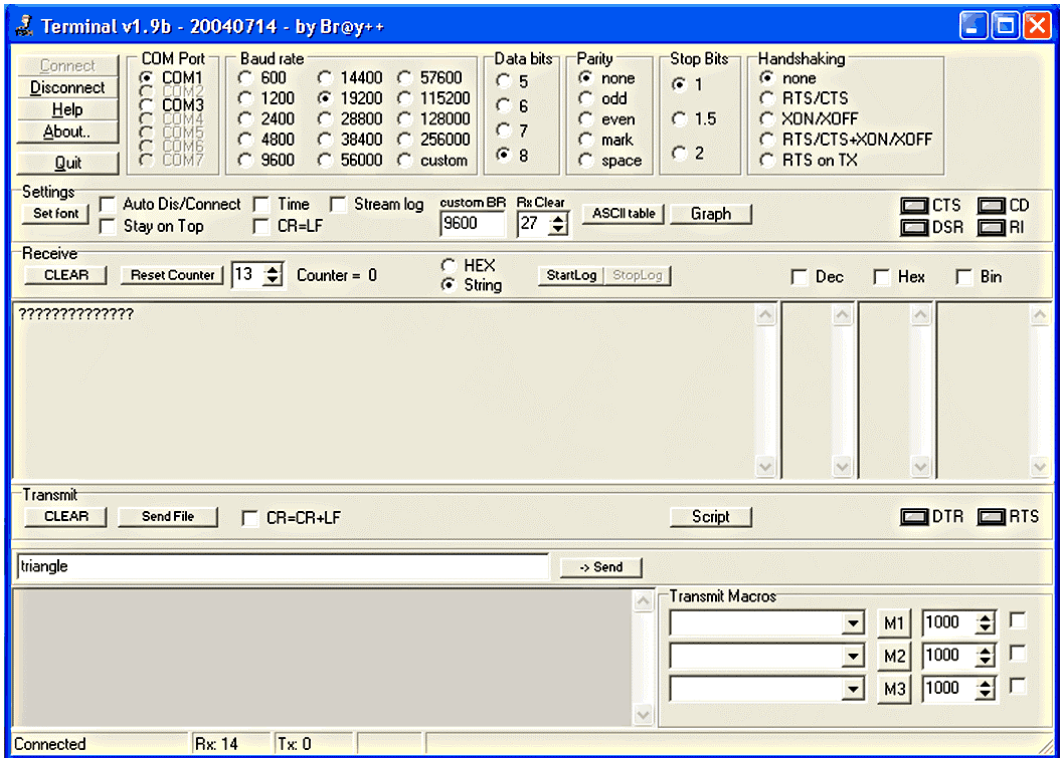


Figure 7: Bray's Terminal

If you don't get the string of question marks, then try the other COM ports (in Figure 7 only COM1 and COM3 are shown for my machine, yours may be different. Press disconnect then connect and try again. If it still doesn't work, carefully check that the RS-232 cable is connected. Try again. Still no? Recheck that you've got the DB-9 soldered correctly to your Butterfly. Try again. Still no? Is it turned on? If you move the joystick upward do you get the LCD scrolling message? Yes? Turn it off and on and press the center again. Still no? If its not working by this point go back and meticulously retry everything you can think of, including passing a dead chicken over the setup while chanting voodoo hymns. It took me a while to get all this running and I supposedly know what I'm doing, so don't feel bad if this is a little harder than you might hope. (You get what you pay for).

In a moment you will scoop out the Butterfly's brains, toss them aside, and stick in some brains that Igor got from a garage sale, so let's do one final test on the Butterfly as it came out of the package. If all goes well, you will eventually be able to reload the Butterfly's original brains, but all seldom goes will, as Igor will readily attest.

With the Butterfly hooked up to the RS-232 port and the Br@y++ Terminal running, turn the Butterfly on and click the joystick up to get the LCD scrolling. Move the joystick straight down three times till you see 'Name' then move the joystick to the right twice till you see 'Enter name' then move the joystick straight down once and you will see 'Download name' then push down the joystick center for a moment until you see 'Waiting for input'. Now write a name in the bottom text panel of the Br@y++ Terminal (Figure 8) and hit enter (or push it gently if you prefer). The name you entered should be scrolling across the LCD as shown in Figure 6.

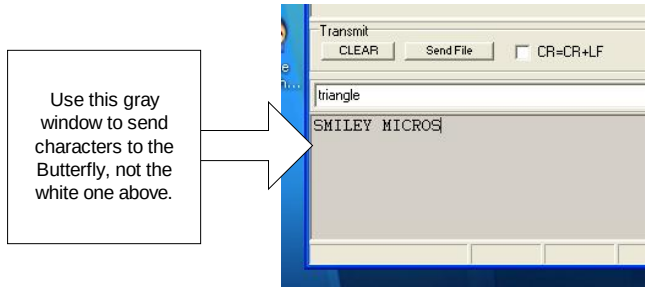


Figure 8: Enter name to send to the Butterfly

Let's Blink Some LED's:

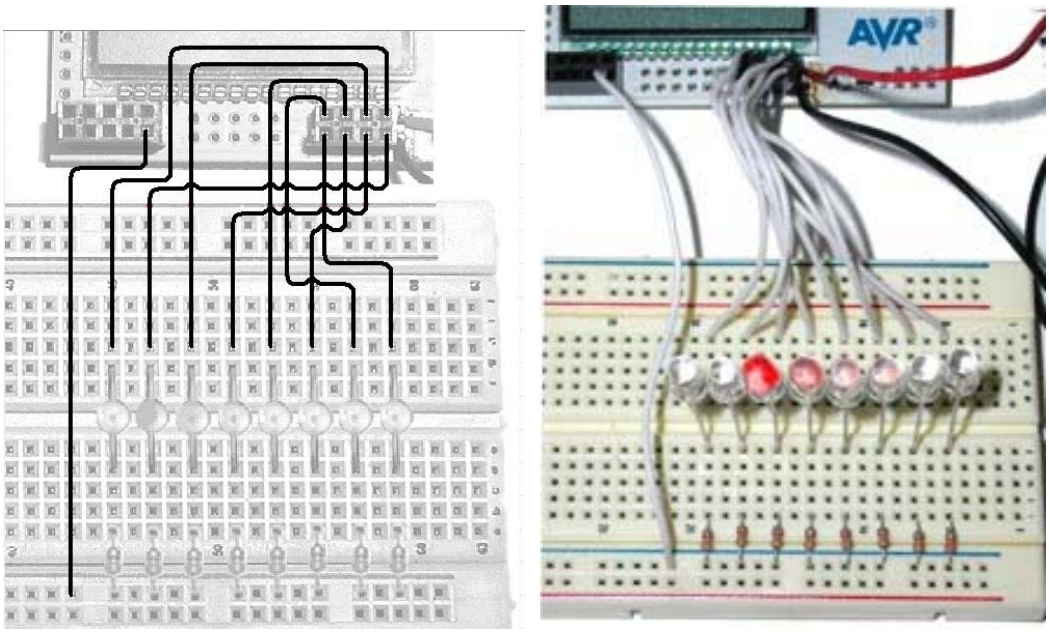


Figure 9: Blinky wiring diagram and photo of wired board

All the parts are listed in the JAMECO parts list Appendix 1. Put the LEDs in the breadboard with the short leg on the resistor side. Use the 330-Ohm resistors to jumper to the ground strip. You'll need to make a bunch of jumper wires, cut 9 pieces about 4 inches long strip each end about 3/8 inch, and connect them to the breadboard as shown in Figure 9, with the right most LED connected to pin 1 of PORTD (Figure 2) and subsequent pins connected sequentially. The pins are numbered with the odd pins on the bottom of the PORTD land and the even pins on the top. Cut a 6" wire and use it to connect the ground strip to the ground pin of PORTB as shown.

Now connect your RS-232 cable between the computer and the RS-232 connector you soldered to the Butterfly. Your hardware should look like Figure 10.

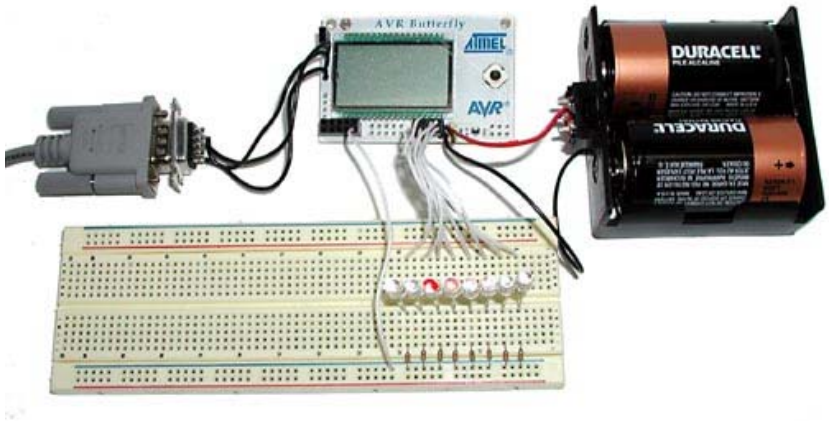


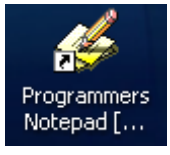
Figure 10: Hardware setup for Blinky.

Blinking LEDs – Your First C Program

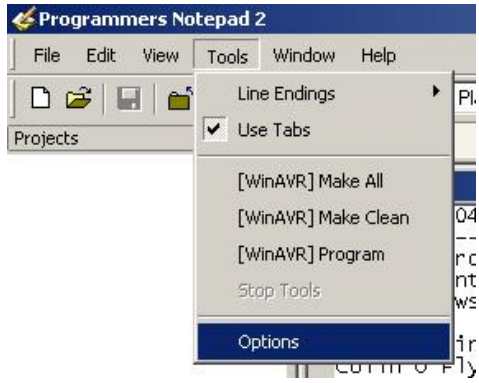
You might wonder why blinking an LED is the first project, when traditional C programming texts start with the classic "Hello, world" program. It certainly seems that since the Butterfly has an LCD that can show the words it would be easy. But the reality is that controlling the LCD is much more complex than blinking an LED, so we'll save the LCD for later when we've gotten a better handle on things.

- Make a directory called Blinky for this project.
- Copy '…/WinAVR/Samples/makefile' (notice that it has no extension) to the Blinky directory.

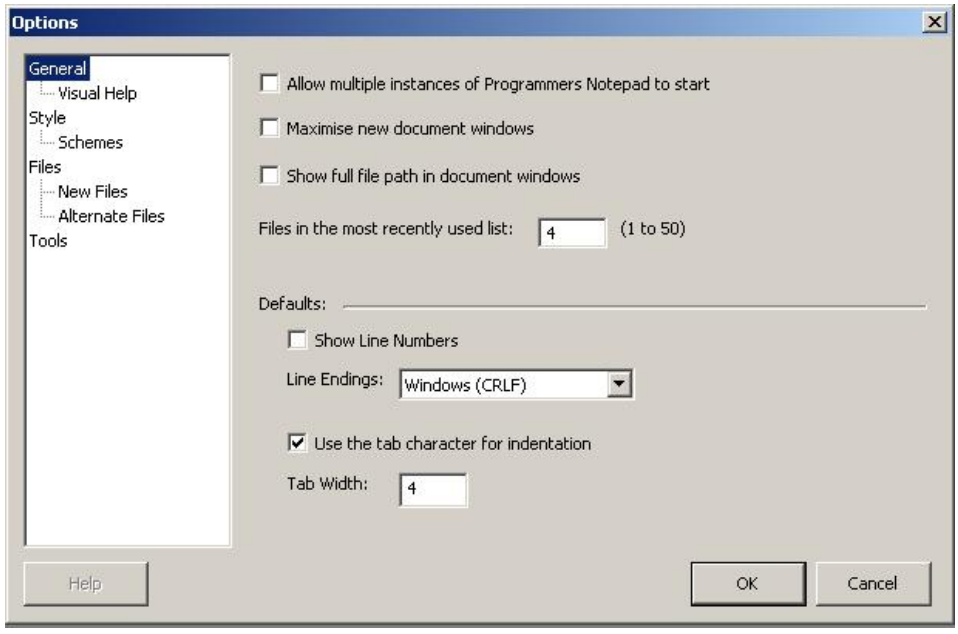
Write it in Programmers Notepad

- 

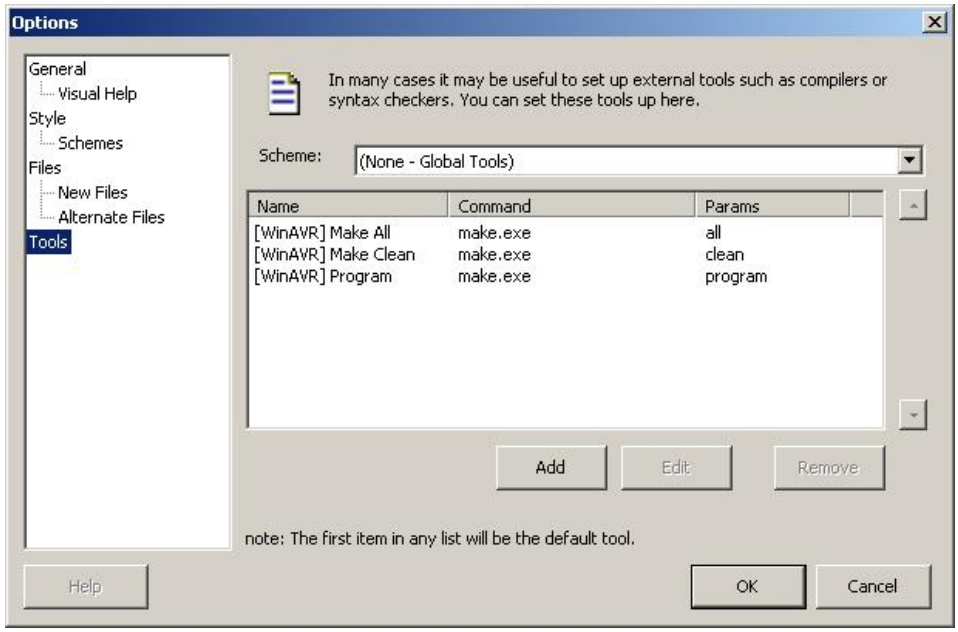
Find Programmers Notepad that was installed as part of WinAVR (you should have an icon for it on your desktop) and open it. You will need to add a tool, which will let you use the AVR Studio simulator.
- Open the Tools menu and click on Options.



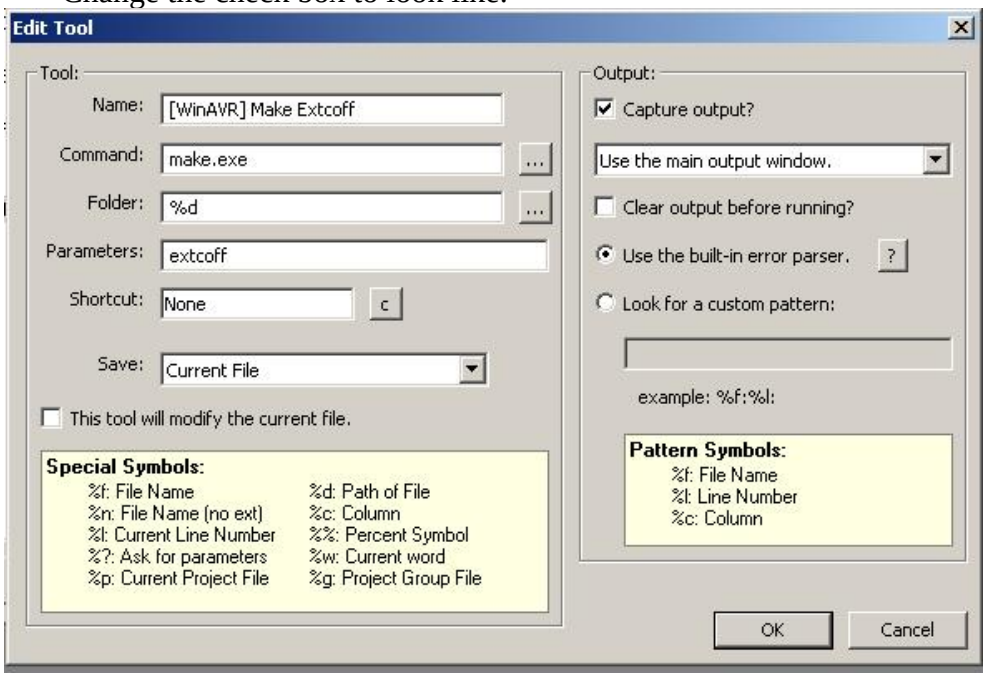
- In the Options window select Tools:



- Then select Add:



- Change the check box to look like:



- Click OK.
- Click File, then New, then C/C++, and name it Blinky.c.
- Save in Blinky directory and CAREFULLY TYPE exactly as shown:

```
// Blinky.c

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

int main (void)
{
   // set PORTD for output
   DDRD = 0xFF;

      while(1) {

            for(int i = 1; i <= 128; i = i*2)
            {
               PORTD = i;
               _delay_loop_2(30000);
            }

            for(int i = 128; i > 1; i -= i/2)
            {
               PORTD = i;
               _delay_loop_2(30000);
            }
      }
      return 1;
}
```

- Open File and again save ‘Blinky.c’ to your Blinky directory
- NOTE: YOU MUST ADD THE EXTENSION ‘.c’ TO THE NAME
- Open the file ‘makefile’ in your Blinky directory.
- Change these lines:

```
MCU = atmega128

# Output format. (can be srec, ihex, binary)
FORMAT = ihex
```

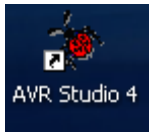
```
# Target file name (without extension).
TARGET = main
```

- To:

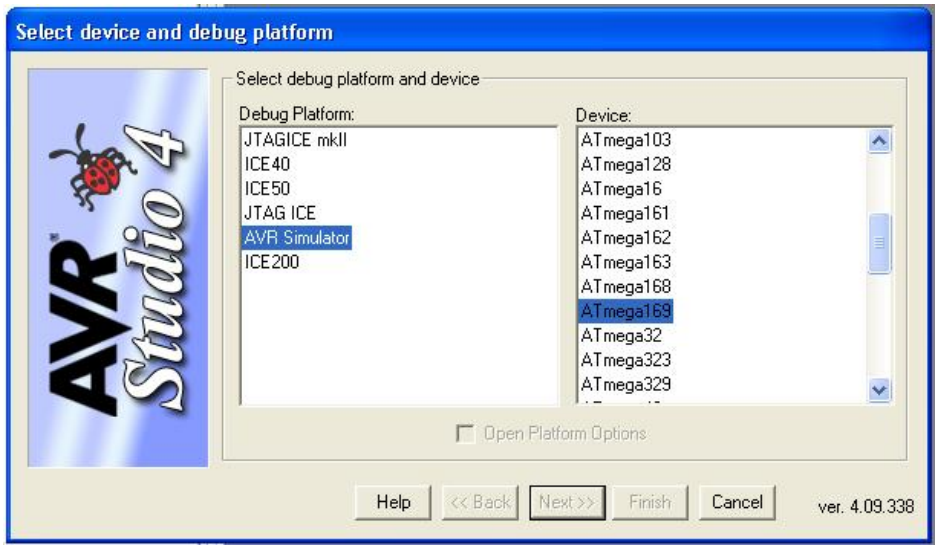
```
MCU = atmega169

# Output format. (can be srec, ihex, binary)
FORMAT = ihex

# Target file name (without extension).
TARGET = Blinky
```

- Close and save changes to makefile to the Blinky directory.
- Open Tools and click [WinAVR] Make All to make your Blinky.hex file
- Open Tools and click [WinAVR] Make Extcoff to make your Blinky_coff file.

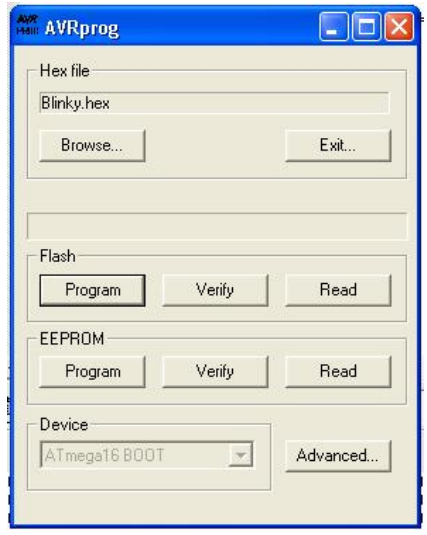
Download to the Butterfly with AVRStudio

- 

Find AVR Studio (you should have an icon for it on your desktop) and open it.
- In the File menu Open '…\Blinky\Blinky.cof
- Select the AVR Simulator and the ATMEGA169 as:



- Select Finish
- DO NOT try to run the simulation; the delay loop will take forever to run. We'll use the simulator later.
- Turn the Butterfly off.
- Press and **hold down** the joystick button.
- Back to the AVR Studio, open the Tools menu and WHILE HOLDING DOWN THE JOYSTICK BUTTON click the AVR Prog menu item. Then wait until you see:

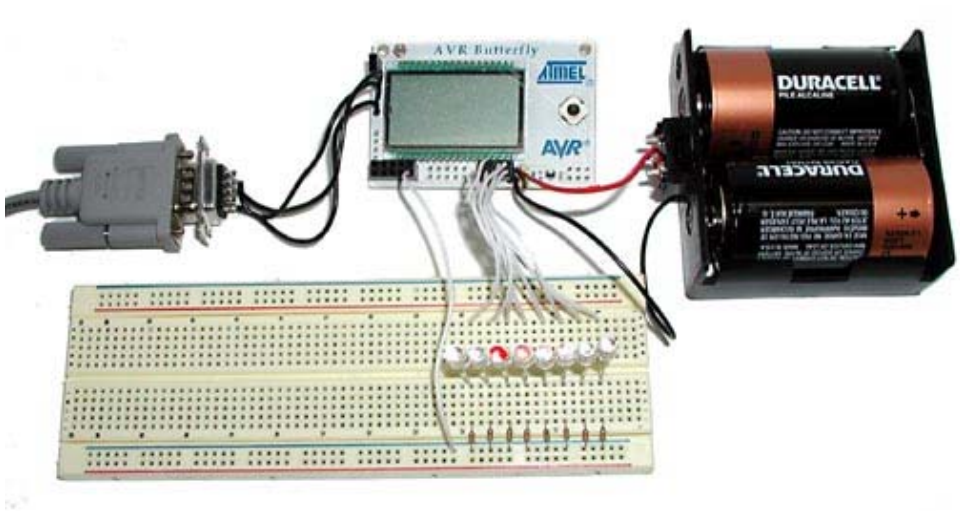


- Release the joystick button. Your finger hurts doesn't it? Enter Blinky.hex in the 'Hex file' box. Press the program button and the program should magically flow from your PC into the AVR Butterfly Flash memory.
- AVR Prog will say: Erasing Programming Verifying OK.
- WHEN YOU WANT TO DOWNLOAD A DIFFERENT HEX FILE, DON'T FORGET TO CHANGE THE HEX FILE NAME. DON'T SAY I DIDN'T WARN YOU AFTER YOU WASTE TIME SCRATCHING YOUR HEAD OVER WHY YOUR NEW PROGRAM SEEMS TO RUN EXACTLY LIKE THE LAST ONE YOU DOWNLOADED. I make this mistake a lot.
- If instead of the above window you get:

- Go back a few steps and try again. You probably left Bray's Terminal running so it has locked the port. Then maybe not.

Blinky Goes Live

- Turn the power supply off and then back on, the LCD will be blank, click the joystick up (maybe a couple of times) and:



- Your LEDs should be making like a Cylon with the light bouncing back and forth. If you don't know what a Cylon is, try Googling Battlestar Galactica, not that I'm recommending the series, but the bad guys had great eyes:

Figure 11: From the cover of the Battlestar Galactica comic *Red Cylon*.

When you compiled Blinky.c you may have suspected that a lot of stuff was going on in the background, and you would have been right. The compiler does a lot of things, and fortunately for us, we don't really need to know how it does what it

does. We only need to know how to coax it to do what we need it to do, which in our case is convert Blinky.c into Blinky.hex that we can download to the Butterfly. If you raise the hood on WinAVR you would see a massively complex set of software that has been created over the years by folks involved in the open software movement. When you get a little extra time check out www.sourceforge.net.

When you have questions about WinAVR, and you will, check out the forums on www.AVRFreaks.net, especially the gcc forum, since WinAVR uses gcc to compile the C software. Try searching the forums before asking questions since someone has probably already asked your question and received good responses. Forum helpers tend to get annoyed with newbies who don't do sufficient background research before asking questions.

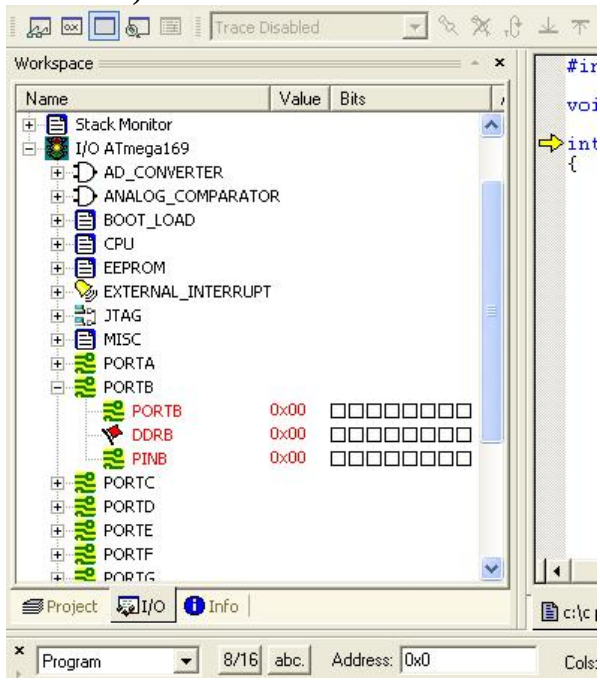
Simulation with AVRStudio

Now that you've gone to the trouble to construct the hardware, and have the burned fingers to prove it… guess what? You didn't need to do any of that to test Blinky or get an introduction to C programming for microcontrollers. With a minor modification you can run Blinky in the AVR Studio simulator and learn the introductory C programming ideas in the next chapter without any of the hardware. I decided to do things the hard way, ummm… hardware way because our goal is to control 'real' things like LEDs, not virtual things like little boxes on your PC screen. Theoretically, we could have a whole slew of virtual things to control, from LEDs to motors to full blown Cylon robots reeking havoc on your screen, which actually sounds kind of fun, but not nearly so much fun as having a real Cylon robot stomping around your neighborhood scaring the noodles out of your enemies. Fun aside, it is often more practical to simulate software before running it in the real world. You wouldn't want your Cylon to mistake you, the imperious leader, for an enemy, would you?

The simulator runs your program in a virtual environment that is MUCH slower than the real microcontroller. Most of your code will run plenty fast to simulate, but some things, such as the delay functions take too long to simulate. In Blinky we call _delay_loop_2(30000); We don't know yet how this function works, but we can guess that we are telling it to do something 30000 times. If we simulate

the delay, the simulated LEDs will move at geologic speeds, making glaciers seem fast, so we remove the delay before simulation.

- Open Blinky.c in Programmers Notepad and save it to a new directory, SimBlinky, as SimBlinky.c.
- Put comment lines in front of **both** of the _delay_loop_2() function calls in main():
- // _delay_loop_2(30000);
- Open the makefile in the Blinky directory
- Change the target: TARGET = SimBlinky
- Save the makefile to the SimBlinky directory
- Run the Make All, then Make Extcoff.
- In the AVRStudio open the SimBlinky.coff file.
- In the AVRStudio Workspace window click the I/O ATmega169, then the PORTD, you should see: (the following image shows PORTB instead of PORTD, -- live with it)



- In the toolbar click the AutoStep button:

- The simulator will run showing the LED scan as a scan of the PORTD and PIND items in the Workspace window:(this shows PORTB but you'll actually see PORTD)

- See, I told you it wasn't as much fun as watching real LEDs blink.
- Spend some time with the AVR Studio simulator and associated help files; you'll find the effort well worth it in the long run.

GOOD GRIEF!

That was a 'Quick Start'???? Well, maybe things would go quicker if you wanted to pay a fortune for a software and hardware development system, but for **FREE** software, and unbelievably cheap hardware, you've got to expect to do a little more of the work yourself. Besides, you couldn't pay for all the debugging education I bet you got just trying to follow what I was telling you. If you think the 'Quick Start' section was confusing, you should try reading all the stuff it's based on.

Shameless Self-Promotion

If you learned something from these free first two chapters of C Programming for Microcontrollers – Featuring ATMEL's Butterfly and the free WinAVR compiler, and just can't wait to learn more: you can get the full 300 page text from SmileyMicros.com, either as a printed book or as an e-book.

Appendix 1: Project Kits

Note: check the website: www.smileymicros.com to see if any of these items are available there for less (don't forget to include shipping and handling in your calculations when figuring 'less'). And yes, they are available and for less.

Parts Lists (Prices for Spring 2005):

From Digi-Key:
Note: Digi-Key charges a $5 handling charge on all orders under $25. Since the AVR Butterfly is $19.99 they add the $5 charge, but if you buy $5.01 additional items, they drop the $5 handling charge, giving you $5 worth of stuff for a penny. So add $1.84 of extra parts to the order and get $5 free. I like free, don't you?

Description	Part Number	Quan	Price/Unit	Total
AVR Butterfly	ATAVRBFLY-ND	1	19.99	19.99
D-SUB 9 female solder cup	209F-ND	1	0.63	0.63
Female header single 2 pin	S4002-ND	1	0.21	0.21
Female header single 3 pin	S4003-ND	1	0.30	0.30
Female header single 4 pin	S4004-ND	1	0.39	0.39
Female header double 10 pin	S4205-ND	2	0.82	1.64
Subtotal				23.16
MORE STUFF – see note	**Your choice - see note**			1.84
Total				25.00

From JAMECO:

Description	Part Number	Quan	Price/Unit	Total
Breadboard	20722CP	1	8.95	8.95
Wire cutter striper	127870	1	5.49	5.49
22 awg 100' white solid wire	36880	1	5.49	5.49
Battery holder – 2 size D	216389	1	0.89	0.89
Switch	204142	1	0.39	0.39

Data I/O				
LEDs red	11797	10	0.19	1.90

Resistors 330 Ohm 1/8 watt	107967	100	0.0069	0.69
8 position DIP switch	30842	1	0.89	0.89
Potentiometer – 10 kOhm	264410	1	1.18	1.18
PWM Motor Control				
Motor	231896CA	1	0.99	0.99
Power Transistor – TIP115	288526	1	0.48	0.48
Optoisolator – 4N28	41013	1	0.46	0.46
Diode – 1N4001	35975CA	10	0.03	0.30
9v Connector	216451	1	0.34	0.34
Slotted Interrupter – H21A1	114091CL	1	0.69	0.69
2.2 k Ohm resistor	108054	100	0.0069	0.69

Solder Kit

Description	**Part Number**	**Quan**	**Price/Unit**	**Total**
Soldering iron	208987	1	2.99	2.99
Solder	170456	1	1.39	1.39
Solder wick	410801	1	1.49	1.49

The Butterfly Quick Start was extracted from C Programming for Microcontrollers – Featuring ATMEL's Butterfly and the free WinAVR compiler (available from SmileyMicros.com)