So I Got a Butterfly – Now What?

Chuck Baird
avrfreaks.net, user name zbaird

Table of contents:

Introduction

This guide was written to help the novice user understand and appreciate the potential of the AVR Butterfly, Atmel's "demonstration and evaluation kit" for their line of LCD capable 8 bit processors. The term "AVR" allegedly means nothing more than a marketing term for the product line, although it coincidentally happens to be the initials of the inventors. At the time of this writing the Butterfly retails for about $20 in the United States.

We will take a look at the Butterfly board and its complement of peripherals, the microcontroller that provides its "smarts," how to upload new programs into it, and some of the documentation and support resources that are available for it. We will only be able to touch the high points to get you started; there is much more that we will omit but which you can discover with a little searching and digging. Always remember, Google is your friend.

If this is your first foray into the world of embedded computing, enjoy the trip. These small, general purpose computers are ubiquitous, assisting you hundreds of times a day in ways you never dream of. The Butterfly will give you a glimpse into their array of capabilities.

Features

Photo 1 shows the front view of an older model Butterfly when they were shipped with a light sensor (an LDR, or photo resistor, shown in the upper left hand corner). Because of RoHS restrictions against cadmium the sensor is no longer mounted, although you are certainly free to add one yourself since all the support circuitry is still included.

The small holes around the left and bottom edges of the board are for connecting various external circuits to the Butterfly. They use 0.1” spacing, so standard headers may be soldered in, or wires can be soldered directly to the board. However, nothing external has to be connected to the board for it to function right out of the box.

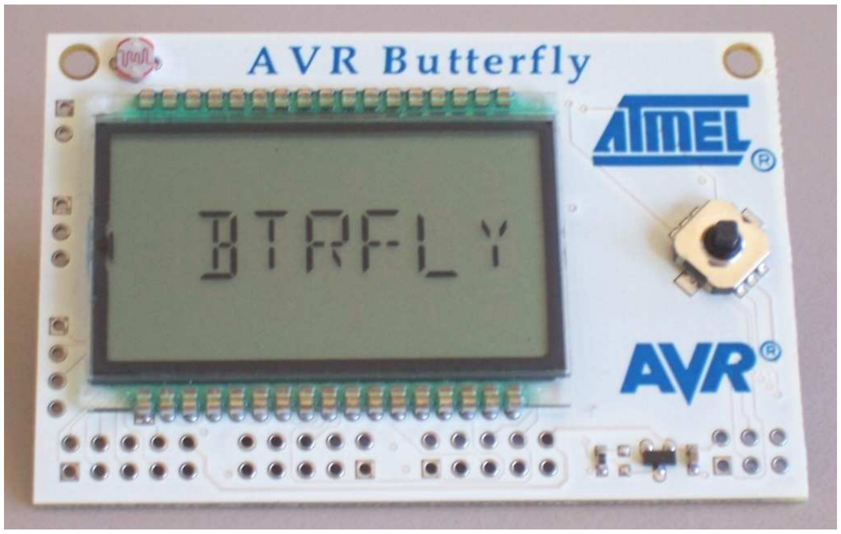

Photo 1: The mighty Butterfly (early model with LDR)

The Butterfly’s 6 character, 100 segment LCD is driven directly by the microcontroller – there is no separate LCD controller on the board. Other features of the Butterfly include a 5 position joystick (shown on the right hand side of Photo 1), a piezo element for making noise (or music, depending on your tastes), a 4 megabit nonvolatile dataflash memory, a temperature sensor, and RS-232 level converters for serial communications. It can read voltages (0 – 5v) and has a 32 KHz crystal for real time clock applications. And, as mentioned, it can read a variable resistance where the light sensor was designed to be mounted.

The Butterfly’s processor is an Atmel ATMega169 8 bit RISC MCU (reduced instruction set microcontroller unit) powered by a 3 volt coin battery (CR2450). The MCU includes 16K bytes of flash memory (for program storage), 1K bytes of static RAM (for variable storage), and 512 bytes of EEPROM (for nonvolatile variable storage). It has 3 timer/counters, an 8 channel 10 bit A/D converter, an analog comparator, watchdog timer, sleep modes, and more. The ATMega169 datasheet (see the *Resources* section) details all the features.

As shipped, the Butterfly is preloaded with a bootloader and an application demo program that shows off some of its tricks. The battery is included, but has a cardboard spacer which prevents battery drain prior to it reaching the end user. Removing the cardboard spacer should bring your Butterfly to life, although it may not appear to do anything until you press the joystick up and wake it from its nap. See the section below, *Getting your Butterfly to do something*.

The bootloader

A bootloader is a small program that can accept input, usually a ready-to-run program, from some source and write it into the memory of the computer the bootloader is running on. Typically a bootloader will either automatically or in response to some command jump to (execute) the new program after it writes it to memory.

The name comes from the phrase "to lift yourself up by the bootstraps," an idiom of unknown origin describing an impossibility but meaning to improve your situation by your own efforts. It was borrowed by the early computer geeks to mean getting a tiny program started which progressively brings in a larger program, and is the origin of the phrase "to boot a computer."

The Butterfly comes with a preloaded serial bootloader program. It also comes with a preloaded demo application, a separate program. The two programs reside in different parts of the ATMega169's flash memory.

The Butterfly's bootloader is called a "serial" bootloader because it receives its commands and code to write to memory from the serial, or RS-232, communications line on the Butterfly. This is a standard method of computer communications initially used by almost all personal computers, although USB has pretty much replaced it in recent years. If your PC does not have a serial port (a place to plug in an RS-232 connection), you will need to buy a USB to serial converter to use the Butterfly's serial bootloader. Or find a PC that does have a serial port.

There are other ways to program a Butterfly, and we will briefly mention them later. For now we will just concentrate on the preloaded serial bootloader.

The demo application program

On the Butterfly's box is a flowchart of the demo program. It is also reproduced in the Butterfly document (see *Resources*). Each white rectangle represents an activity or choice presented as a menu item on the LCD. Until you get used to poking the joystick correctly you may find yourself jumping around to places in the flowchart other than where you expect.

As you see from the diagram, you can set or read the time-of-day clock, play some music, enter text (via the joystick or the serial line) for display on the LCD, read the temperature, light sensor (if present), or an external voltage (0 – 5v), and set various options. When the joystick is inactive for a certain length of time, the demo program puts the MCU to sleep to conserve the battery.

The demo program is a large and sophisticated piece of software, and its C source code has been published by Atmel. The program was converted ("ported") to WinAVR C (more on that later) and that version is also available on the internet. Reading through the code, if you are interested in such things, will teach you a lot about how to program the Butterfly and AVRs in general.

Atmel has also made the Intel hex file of the demo program available on its website. You may reload it into your Butterfly to restore it to its pristine state.

Getting your Butterfly to do something

When you apply power to your Butterfly it resets the MCU. This is a design feature of the AVR line and most other embedded processors. It is also possible to reset an AVR without removing and restoring power, and later we will see how to add a reset button to your Butterfly.

Thus pulling the battery out and reinserting it causes a reset, as does that initial removing of the cardboard spacer isolating the battery on a new Butterfly. On a reset, the ATMega169 starts executing code in one of two places: at the start of the application section of memory, or at the start of the bootloader section of memory. The application section of memory starts at a fixed address (0x0000), and the bootloader section can start at one of four different addresses depending on how we have set things up.

How do we tell it which we want? There are some indicators (“fuses”) in nonvolatile memory which hold our choices. If a certain fuse called BOOTRST is programmed one way, a reset jumps to the application section. If it is programmed the other way, a reset jumps to the bootloader section. It is our responsibility to have previous loaded code at the proper address for the way we have set the fuse(s).

Perhaps needless to say, the Butterfly’s serial bootloader has been preloaded into the bootloader section of memory, and the demo application has been preloaded into the application section. The BOOTRST fuse has been programmed to make a reset jump to and execute the bootloader, *or at least it should be*.

The bootloader needs to know whether you are wanting to upload a new application or run the existing application. On the Butterfly the one source of easy user input is the joystick, so the bootloader waits for you to press the joystick up (jump to the application) or in (start uploading a new application using the serial communications line). If it doesn’t see either joystick press within a certain amount of time, it goes to sleep. That is, it stops executing instructions and enters a low power mode.

To you, the casual observer, this means the Butterfly just appears to sit there like a lump. It will continue to sit there until you press the joystick up or in. Doing either will wake it out of its sleep mode, and cause it to take the appropriate action.

But wait, there’s more

Let’s say you press the joystick up, meaning you would like to run the existing application (the demo, unless you have loaded something else). After some initialization that takes a second or two, the demo program starts scrolling a banner message across the LCD, waiting for further joystick input which navigates you through the menu structure.

When you see the banner, the bootloader is no longer executing, and the demo program is in control.

One of the menu choices on the demo program is to jump to the bootloader, and if you choose it you will execute a reset. At that point the entire process starts all over from the beginning.

And now for the bad news. If your Butterfly immediately (within a second or two) shows the scrolling banner on power up or reset, then either the BOOTRST fuse is set incorrectly or the bootloader is missing (or both). In early 2009 Atmel shipped some Butterflies with the BOOTRST fuse incorrectly programmed, although they were supposed to have been recalled. If your fresh-from-the-box Butterfly comes to life magically by itself and displays the banner without you pushing the joystick up, you have one of these units.

If you have an incorrectly programmed BOOTRST fuse you may exchange your Butterfly, find someone with the hardware programmer needed to fix the fuse, or invest in the hardware yourself. Even if you could run the serial bootloader somehow (you can't), it is unfortunately incapable of changing that fuse itself. It takes the intervention of a *Deus ex Machina* to save the day.

Adding some goodies to your Butterfly

If you want to use the serial line to input data (names) into the demo application, or if you want to use the Butterfly's serial bootloader to load a different program, you will need to add a connector to your Butterfly. This may be soldered directly on the board, or you may use 0.1" headers and have the ability to unplug it when it is not in use. Of course you will then need to make sure you orient it correctly when you plug it in.

You may also wish to make a battery power supply using two AA cells (or C cells, or D cells) to use in place of the coin battery that comes with the Butterfly. You will find that using the RS-232 interface will run your coin battery down fairly quickly. If you choose to make an external power supply, you may put a switch on it to provide an easy way of doing a reset, and killing power when the Butterfly is not in use. The sleep modes are low power, but they're not zero power.

And, if you want to go whole hog, you can even add a reset pushbutton.

All of this is described in excruciating detail in the link labeled *Appendix A* on my website at **http://www.zbaird.com**. The Reader's Digest condensed version is shown in the following three diagrams.

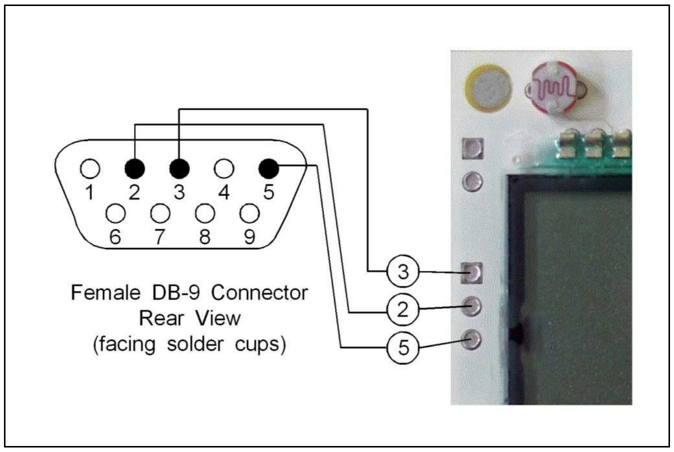

Figure 1: Adding an RS-232 connector

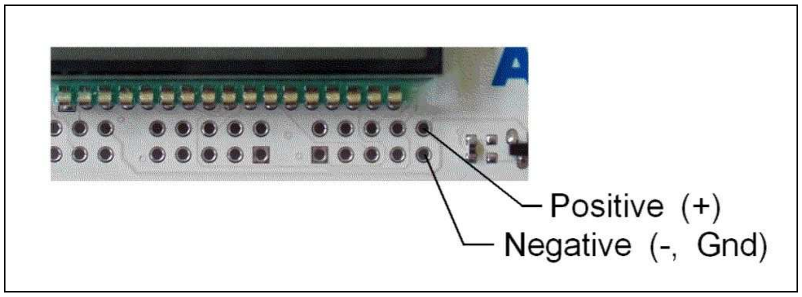

Figure 2: Adding a 3v battery power supply (remove the coin battery!)

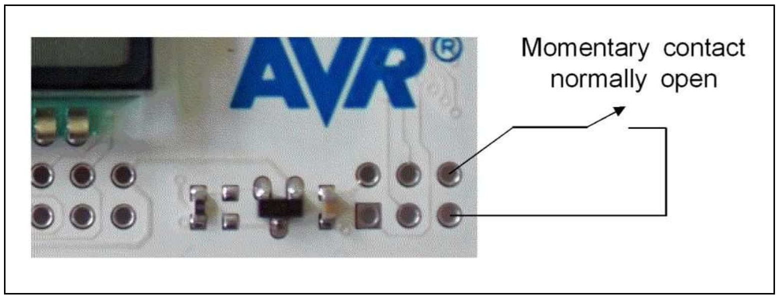

Figure 3: Adding a reset switch

Programming (uploading to) the Butterfly

Let's assume that your BOOTRST fuse is set correctly, and your Butterfly sleeps quietly until you press the joystick up. Eventually you may want to try something else besides the demo application, and that involves getting the bootloader to overlay the demo program with some other program, perhaps one you wrote.

The program needs to be in a file format called "Intel hex." This is a method of encoding data in printable characters that are highly portable (can be read by virtually all computers), and is the *de facto* standard for programming embedded MCUs. All embedded compilers and assemblers generate Intel hex files as output, either automatically or on request.

The Intel hex file for the Butterfly's demo program is available on the Atmel website, so you can reload it if you wish. The Butterfly's bootloader hex file is also there, as well as a combined file that includes both.

You need a PC program to talk to the Butterfly's bootloader and pass it data and the commands needed to write that data to memory. One such Windows program is *AVR Studio*, available free from Atmel. It is a very large download, over 125 megabytes, so if you do not have a high speed internet connection it is worth finding one and burning the installation file to a CD. There are other programs, both for Windows and other operating systems, but here we will only consider AVR Studio.

An AVR Studio – Butterfly bootloader session

Here we describe how to upload (program, burn) a new program into the Butterfly's flash memory using AVR Studio and the Butterfly's serial bootloader. As mentioned, you may also restore the original demo program using this technique.

You will need to have a copy of the Intel hex file you want to upload, and be able to tell AVR Studio where it is.

Install AVR Studio. Plug your newly added Butterfly's serial connector into your PC's serial port (or your USB to serial converter). You may need a *straight through* (not a *null modem*) cable to have enough wire length to make this connection. Turn on your Butterfly.

You can test your Butterfly's connection by running a terminal program such as HyperTerminal, configuring its serial port to a baud rate of 19200, 8 data bits, and either 1 or 2 stop bits. When you press the Butterfly's joystick in (from the sleep state – not while the demo is running) you should see a stream of question marks received by HyperTerminal. This is the bootloader trying to connect to the host software.

If you cannot receive the question marks, there is little use in trying to connect to AVR Studio. Check your connection, wiring, and COM port selection and parameters. Remove and reapply power to the Butterfly. You are not seeing the scrolling banner on the LCD, are you? (hint: you shouldn't be)

We will assume this part worked, and forge ahead.

Run AVR Studio. It's a large program and takes a while to load. Be sure your Butterfly is in the reset (sleep) state, not running the demo. You can toggle the power or choose the *Jump to Bootloader* option from the demo's menu.

While pressing the joystick in (this will hurt your thumb), click on *AVR Prog...* (marked "Yes") as shown in Photo 2.

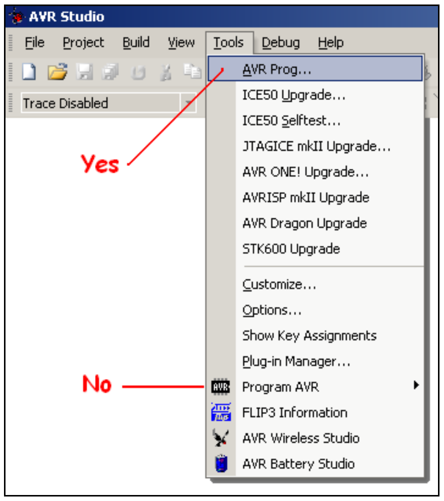

Photo 2: Connecting to the serial bootloader

The *Program AVR* selection (marked "No") is used for ISP programming such as might be done using an STK500. It uses a completely different protocol than that expected by the Butterfly's serial bootloader.

The window shown in Photo 3 should appear.

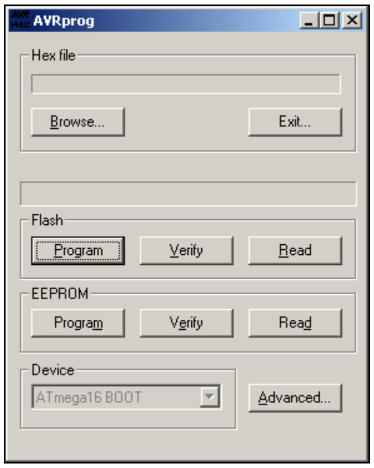

Photo 3: AVR Studio sees the bootloader

AVR Studio is pretty impatient, and doesn't wait long to see the question marks from the bootloader. If you aren't pressing the joystick in when you click the *AVR Prog...* menu item, AVR Studio is likely to tell you it cannot find the target board.

If it did not find the target board, and you're pretty sure you have done things right (and you passed the HyperTerminal test outlined above), one possibility is that you are using a COM port that AVR Studio isn't trying. This can especially happen if you are using a USB to serial converter, as they sometimes assign high COM port numbers to the virtual serial port they install. Look on Studio's Tools menu, in the Options, and increase the number of COM ports to try.

Another trick that sometimes works is to just try it again. Keep pressing the joystick in, close out the nastygram about the unfound target board, and click on *AVR Prog...* menu choice again.

Once you see the AVRprog window, click the *Browse* button and locate the hex file you wish to upload. Ignore the *Device* frame, and the *Advanced...* button, since you can't change the device or change the fuses using the Butterfly's bootloader.

With the hex file name in the textbox, click the *Flash* frame's *Program* button. Studio will erase, program, and verify the upload. When you are done, exit the window and AVR Studio, unplug your Butterfly, and it is ready to run its newly loaded program.

From this same programming window you can upload a hex file into EEPROM, read the contents of either the flash or EEPROM memories (the contents will be stored as a hex file), or verify either memory against the indicated hex file.

Other programming methods

It is unfortunate that the word "programming" has a dual meaning in the embedded world. It means both to design and write code to perform a specific task, and to upload code into the nonvolatile memory of the MCU. For now we are interested in the second meaning, and the first is covered briefly in a later section.

There is a nice tutorial by Dean Camera (user name: abcminiuser) on the avrfreaks website (see *Resources*) that discusses the various programming methods and hardware available for programming most AVRs. A serial bootloader is convenient because it doesn't require any specialized or additional hardware, but it isn't used as often as some of the other methods. On the Butterfly it also has the shortcoming of not being able to access the fuses and certain other parts of the MCU, although this is limited by the design of the bootloader, not the MCU itself.

ISP (In System Programming) is probably the most popular programming method, and one that is available on the Butterfly. With proper design, it allows an MCU incorporated into an existing design (that is, *in system*) to be programmed without removing the MCU chip from the circuit.

JTAG (Joint Test Action Group) is another method of programming that is available on the Butterfly. Originally developed for testing complex circuit boards after assembly, it has been extended to allow full access and control of the processor during execution. This means that for those AVRs that support JTAG, the processor can be single stepped, have registers and memory examined and modified, breakpoints added and removed, and so on while the processor is running. This obviously allows interactive, on chip debugging, as well as programming the flash memory.

High voltage programming is another technique supported by most AVRs, although the Butterfly is not designed to take advantage of it.

The AVR core

One of the nice things about the AVR product line is that almost all of the MCUs use the same processor core with some minor variations. Individual models may vary in speed, power consumption, peripherals, number of I/O bits, size of the three kinds of memories, and so on, but the instruction set and general programming considerations are more or less the same. Once you have learned to program one AVR you know a lot about all of them.

There are specialty lines within the larger AVR world – models that have built in USB support, LCD support (like the Butterfly's ATMega169), CAN bus (automotive) support, and so on. Again, other than the addition of the specialty peripherals, the basics of the processors themselves are virtually identical.

AVRs are RISC machines – reduced instruction set computers. They are optimized to run fast by keeping the instruction set simple and efficient. Most instructions execute in one clock cycle. The clock itself can come from a variety of sources – an internal R/C oscillator (which requires no external components), crystal, ceramic resonator, or external clock signal. The Butterfly is configured to run off the internal oscillator at a maximum speed of 8 MHz.

AVRs use a Harvard architecture, which means the instructions being executed exist in a separate address space (which usually means a different memory) than the data those instructions manipulate. It is actually a modified Harvard architecture, because there is a specialized instruction which allows the processor to read the instruction memory (flash) as data. And, as the presence of the bootloader indicates, it is possible for a running program to write data into the instruction memory.

The place to learn about a particular AVR model is its datasheet, available free from the Atmel website as a pdf file. Most datasheets will have a short summary (20 or so pages) and a full datasheet (possibly in excess of 400 pages), and you may wish to download both for any MCU you are working with. The datasheet is the mother load of information about the core and peripherals for a given AVR and is a necessary reference to keep handy while writing programs.

Assembly, and C, and other languages, Oh My!

A machine language instruction can be thought of as a series of 0s and 1s which, when encountered by the processor, cause a certain action to be taken. Ultimately all computers languages result in a series of machine language instructions, because they are the only things the MCU understands.

Assembly language is a set of mnemonics that make it easier for humans (and programmers) to generate those 0s and 1s. Usually a single line of assembly language results in a single machine language instruction.

Higher level languages, like C, C++, BASIC, PASCAL, and so on, are made up of statements. Each statement may result in one to perhaps dozens of machine language instructions.

An assembler is a computer program that converts assembly language to machine language.

A compiler is a computer program that converts a high level language to either assembly language or machine language. With most compilers you can view the assembly language output to see how it has done the translation from the high level language.

C is probably the most popular programming language in the embedded world. A C standard does exist, and all compilers adhere to it as much as possible. Embedded microcontrollers, however, present special challenges and requirements that exist outside the standard. Each compiler resolves these issues in the ways of their writers' choice, so there are small syntactical differences between each dialect of embedded C. The same is true for the other high level languages as well.

There are many compilers and assemblers available for the AVR line of MCUs. AVR Studio, mentioned earlier as a Windows AVR programming platform, includes an assembler (two, actually) and is free. One of the most popular of the AVR C compilers is WinAVR, a GCC port. WinAVR can be configured as a plug-in for AVR Studio so you can use it directly from Studio.

Something to play with

Here is a hex file for a little program that simulates a pair of dice on the Butterfly. You may highlight and copy the code right out of this pdf document, then paste it into Notepad or some other word processor. Clean up any extra characters, like leading or trailing blanks, that may have been introduced in the copy and paste. Save it as an ASCII text file with a name you'll recognize, perhaps *dice.hex*. Depending on which word processor you use, you may find the name gets a *.txt* extension added to it which you will probably want to remove.

Upload it to the Butterfly using AVR Studio. Then, when the Butterfly gets a reset, it will once again be running the bootloader, waiting for the joystick to be pressed up or in. Press it up, and this program will run. Unfortunately this program also sits there like a lump, waiting for you to press the joystick in. When you do, all the segments in a couple of character positions will light up while the dice are being "shaken." When you release the joystick, a pair of numbers (1 to 6) will display. Press the joystick in again, and the process repeats.

The randomness of this program depends on how long you hold the joystick in, counted to the microseconds. There is no true randomness in a normal computer, of course, but this is a good way of faking it.

This program does not go to sleep, so it will eventually run your Butterfly's battery down. You will need to disconnect the power when you are not using it. Or you can always reload the Butterfly's demo program if you wish, or toggle the power to get back to the bootloader, both of which do sleep.

The Diceman Rolleth code:

```
:020000020000FC
:1000000006C006025B044F0466046D047D0404E030
:100010000EBF0FEF0DBF00E800936100000270093B3
:10002000610064D024982C9A04E0102E03E0202E66
:100030006CD00020E9F72224239449D00027000047
:1000400000001C9905C003950432C9F70027F9CFB9
:1000500023E0222E5AD00A94E9F72ED0222437D05A
:1000600021D023E0022E06D002951CD025E0022EDE
:1000700001D0DACF1F922F92BF93AF935F93A591D8
:10008000B591112422242394 5EE0B695A79508F437
:1000900097D013945A95C9F75F91AF91BF912F9064
:1000A0001F9008950F93F0E0E2E00F70000FE00F53
:1000B00008F4F3950F9108954F934427063018F0F4
:1000C00043950650FBCF0295042B4F9108950F9254
:1000D0001F921F9313E0012E11241A9471D015E082
:1000E00012E6ED01F911F900F9008950F930CE07A
:1000F0000093E70007EB0093E50000E10093E600C2
:1001000000E80093E4000F910895BF93AF931F920E
:100110000F931F9308940027001F112011F01A94C9
:10012000FBCF102EBB27A0E2A20D0C90012004E60D
:1001300011271A95F1F71C911121101539F40A9520
:10014000B9F7002029F00024039402C000240A9487
:100150001F910F911F90AF91BF910895060502016 5
:10016000E0D0A091615121118406890002850 78D3
:10017000522A51292B03537B797A042C547C562E16
:10018000552D2F07577F7D7E083058805A325931C0
:10019000330B5B8381820C345C845E365D35370FB4
:1001A0005F87858610386088623A61393B13638BBC
:1001B000898A143C648C663E653D3F17678F8D8E3F
:1001C000FF93EF930F931F934F93F1E0ECE5002D16
:1001D00083010F044E61DC0013051F068F010E125
:1001E000250002331F04EE0140F0A95E9F701C0E8
:1001F00018E0E10F08F4F39542E0041610F04EE029
:1002000001C048E0141629F020F441E0E10D08F4A3
:10021000F395059108D04A95E1F74F911F910F9101
:10022000EF91FF910895DF93CF93D0E0CCEE102FA4
:10023000169516951695C10F08F4D3950770089476
:1002400011271 11F002311F00A95FBCF08812220EE
:1002500019F41095012301C0012B0883CF91DF9180
:020260000895FF
:00000001FF
```

Listing 1: Dice simulation

Butterfly (and other) resources

The Atmel website (**http://www.atmel.com**) is the place to start for anything related to AVRs and/or Atmel products. Here you will find:

- The Butterfly Quick Start Guide
- The Butterfly User Manual (includes schematics)
- The bootloader hex file (ready to be uploaded)
- The demo application hex file (ready to be uploaded)
- The combined bootloader and demo hex file (ready to be uploaded)
- The AVR Studio program
- The ATMega169 datasheet (summary and complete versions)

- Numerous application notes
- Other datasheets and documentation
- The assembly language instruction set manual
- And much, much more

Use the search feature of the website to locate these things since they could move around. The search term *Butterfly* is a good starting place.

The AVR Studio help files contain a wealth of information about all sorts of things. Have a look through them.

The AVR user community at **http://www.avrfreaks.net** is another extensive resource. It is very active and full of knowledgeable people who are eager to help. Be sure to check out the projects and tutorial sections. Of particular interest is stu_san's *Newbie? Start here!* thread in the sticky section of the AVR forum.

Please read the guidelines and observe posting etiquette when posting, or someone will likely toast your buns. Do your homework first – search the avrfreaks site and use Google and read up on your topic prior to posting.

I can be contacted via private message (PM) through avrfreaks, user name zbaird. I reserve the right to tell you to post your question as a new thread if I feel it is of general interest or value for the community at large, or refer you to some of the resources mentioned here.

Another AVR user community is at **http://www.avrbeginners.net**, although it is primarily oriented toward assembly language programming. It also has valuable tutorials, examples, and links.

An avid and esteemed avrfreak is Joe Pardue, author of a couple of books on AVR programming. Through his website at **http://www.smileymicros.com** he has put together several bundles of projects and parts with and without books. Any Butterfly you buy from him is assured to have its fuses set correctly, and he stands behind what he sells and will make sure you are happy.

Joe is writing an ongoing series of articles on programming the Butterfly and Arduino using C. They appear in *Nuts and Volts*, a monthly magazine, but he also posts past articles on his website under the name *Smiley's Workshop*. They provide a good introduction to AVRs, the Butterfly, and the Arduino.

Other distributors of Butterflies are Digikey, Arrow, and Mouser in the U.S., and others in Europe and elsewhere that can be found with your friend Google's help. At the time of this writing at least a few Butterflies from these sources were being shipped with incorrectly set fuses, although hopefully by the time you read this it will all be ancient history.

Software for development (compilers, assemblers, programmers) are available in several flavors from several sources. While C and assembly language are the most common, you can also program AVRs in BASIC and FORTH and probably other languages. I do not know of an AVR COBOL compiler; sorry. AVR Studio, which includes an assembler and extensive programmer support and can interface with the GCC (WinAVR) and ImageCraft C compilers, is available free from Atmel. WinAVR is free, while ImageCraft and most of the other commercial C compilers have free demo or trial versions that have some time or size limitations.

There are many, many sources of development boards for the AVR product line. The Arduino line is reportedly very easy to use and is AVR based. Everyone and his dog sells some specialty board built around some AVR MCU or another. Atmel has a line of development boards and programmers: the STK500, STK600, Dragon, JTAGICE MK2, and so on.

Summary

So have at it. Of course this has been the briefest of summaries, so most likely you will need to follow up by researching additional information for your topics of interest.

If you have never worked with embedded systems before, and you are interested in exploring programming, expect the learning curve to seem a little steep at times. However, as the *I Ching* says, perseverance furthers. Stick with it, do some reading, ask some questions, learn from your failures, enjoy some successes, and plow ahead. Before you know it you will be an old pro and have the battle scars to prove it.

Good luck!