

## BASCOM AVR Tutorial

BASCOM AVR is a compiler used for writing software to download on Atmel 8 bit microcontrollers. It is easy to learn and use and is very inexpensive.

As you go through the tutorial, you can navigate back and forth with the toolbar on the left of your screen or with the back and forward buttons on the bottom of the page.

At the end of the tutorial, there is a link to open a printable version of this tutorial.

If you have any suggestions on improving this or other tutorials, would like to see another added or even have one ready that you would like to add to the site, send the webmaster an email with the description and, if applicable, a copy of the tutorial you wish to submit.

There is no fee for submitting a tutorial nor is there a fee paid to the submitter. This site is to make available free information and is maintained through affiliate advertising.

### *Downloading*

BASCOM AVR is available for download from several locations. Two that I will mention are the MCS Electronics site, [http://www.mcselec.com/download\\_avr.htm](http://www.mcselec.com/download_avr.htm) . Also, you will find the user manual available on those same pages. I performed this installation on a computer running a 2 Ghz Celeron processor with Windows XP as the operating system.

### *Unzipping*

Start by creating a folder on your desktop named 'bascom.' Download all six of the files (each labeled bcavrd 1.zip, bcavrd 2.zip, etc) into the bascom folder you created on your desktop. After all of the files have been successfully downloaded, unzip the files into the same folder.

### *Installing*

Open the unzipped folder bcavrd 1 and click the setup icon. You may get a warning for the digital signature validation.



Figure 4.1

Click Run and proceed with the installation. The installation is just like installing any other software. You should allow the software to install in the default locations and settings it selects if possible.

When the installation begins, it will prompt you to insert disk #2. If you downloaded and unzipped the files as suggested in this manual, all you need to do is backspace over the 1 in the file location, DESKTOP/Desktop/bascom/bcavrd\_1 and make it DESKTOP/Desktop/bascom/bcavrd\_2 as shown in figure 4.2.



Figure 4.2

You will be prompted to insert the remaining disks through disk 6. Simply perform the same action of typing the new number in the Source Pathname as you did for disk 2 for all of the remaining prompts, typing 3 for disk 3, 4 for disk 4, etc.

After the software has been installed, it is a good idea to restart your computer. Most computers will not be adversely affected without a restart, but it is good practice to do so. When your computer is back up, go to start, all programs, BASCOM-AVR, and click on BASCOM-AVR. BASCOM will start up with an empty window.

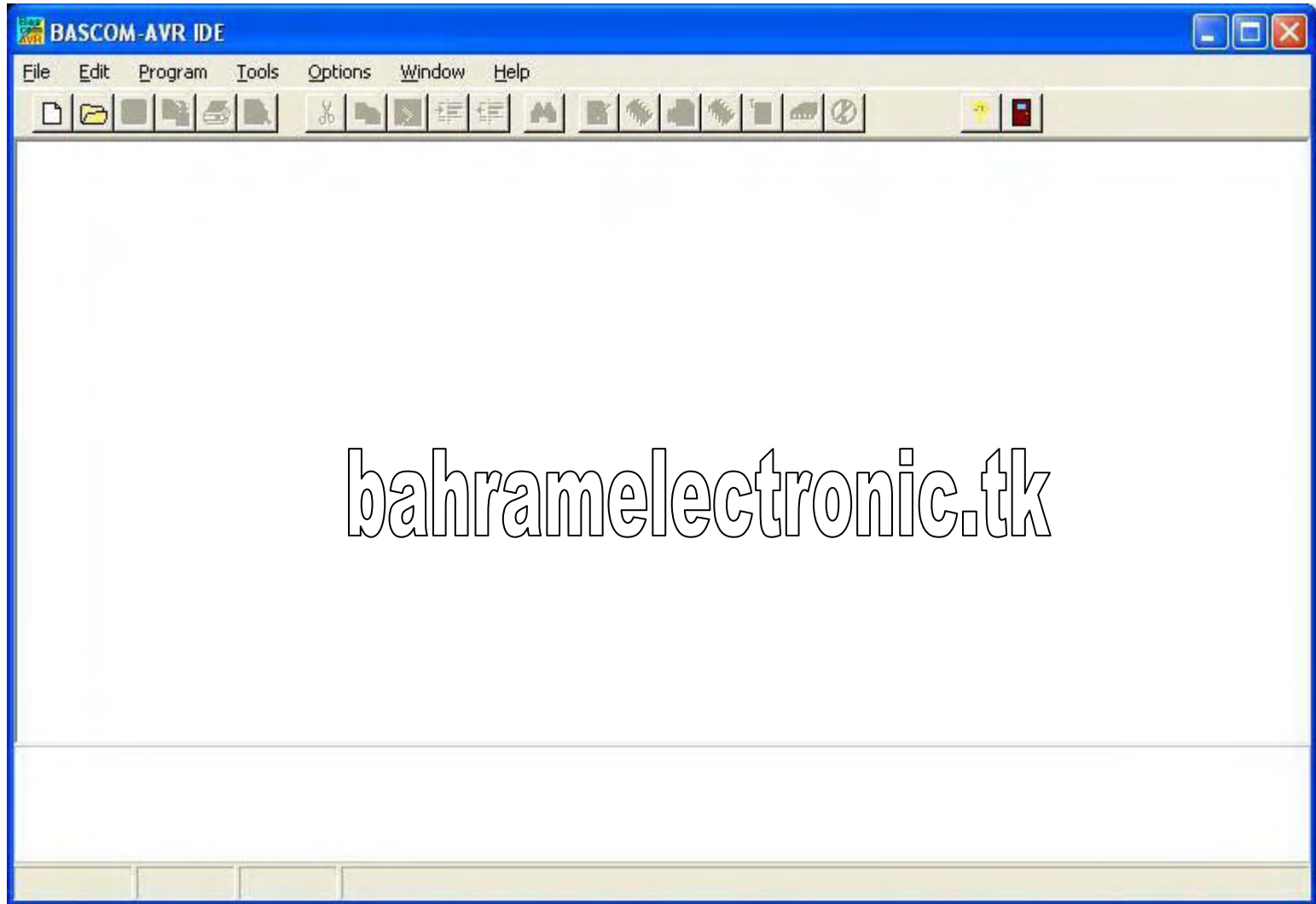


Figure 4.3

### Configuring

When it is opened, click options then programmer to open the page for selecting the programmer you will use with your project. Again, we will proceed assuming we will be using the STK500 programming kit.

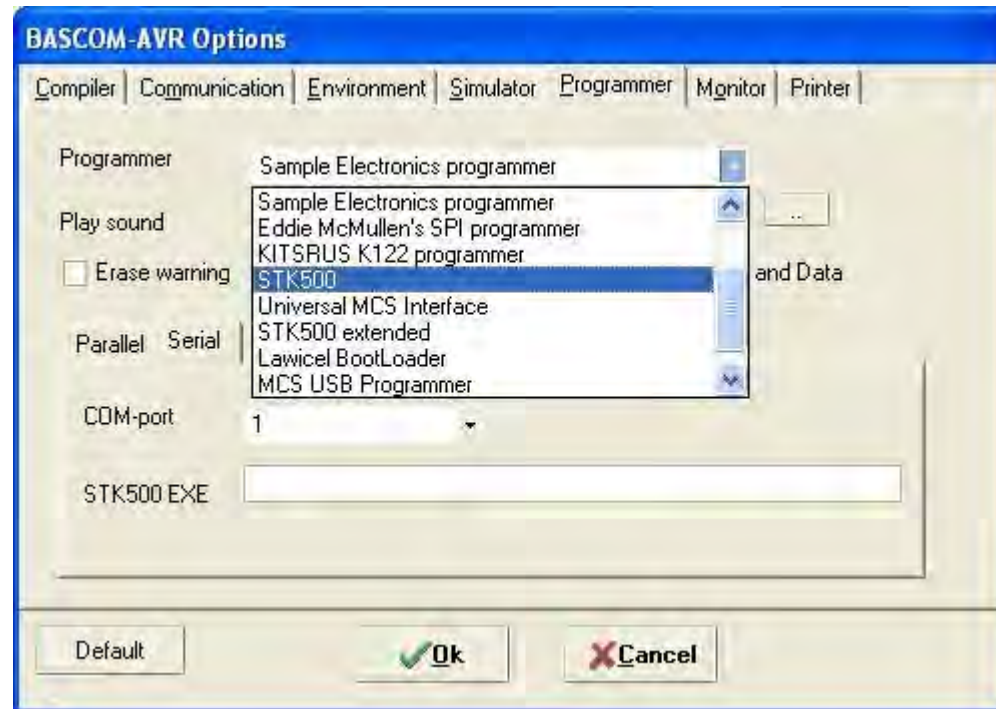


Figure 4.4

When you select your programmer and click OK, a warning dialog will be displayed informing you some of the settings in the options exceeds the selected chips capacities.

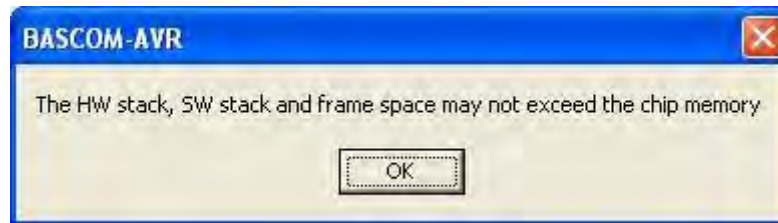


Figure 4.5

The reason for this is no chip has been selected yet. As a result, all settings exceed the chip's capacities (since there are no capacities available). Just click OK and continue.

At the top of the Options menu, click the Compiler tab to select a chip as shown in figure 4.6. If you will be using the same microcontroller in your version of our project, you should select the m16def.dat file. If not, simply select your chip's .dat file.

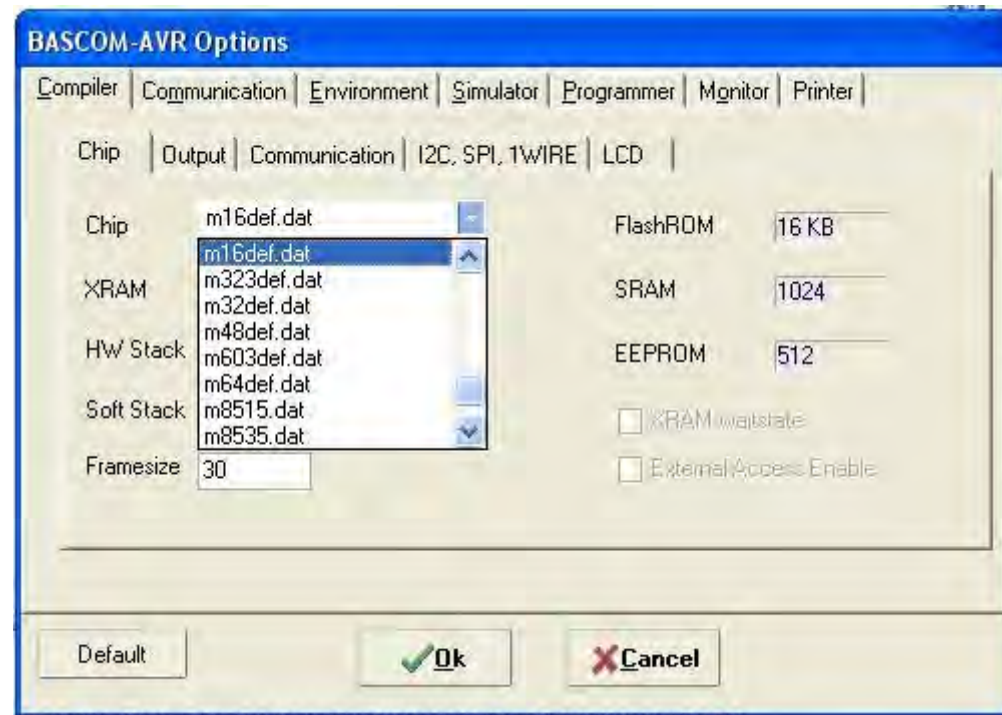


Figure 4.6

After selecting your chip, you can choose to disable the simulator for our project. If enabled, the simulator will appear each time you compile your program and you will need to close it out each time before proceeding. It is a minor inconvenience if you are not using the simulator, as we will not during this project. If you wish to disable the simulator, click on the Simulator tab and uncheck the box indicating the simulator will be used.

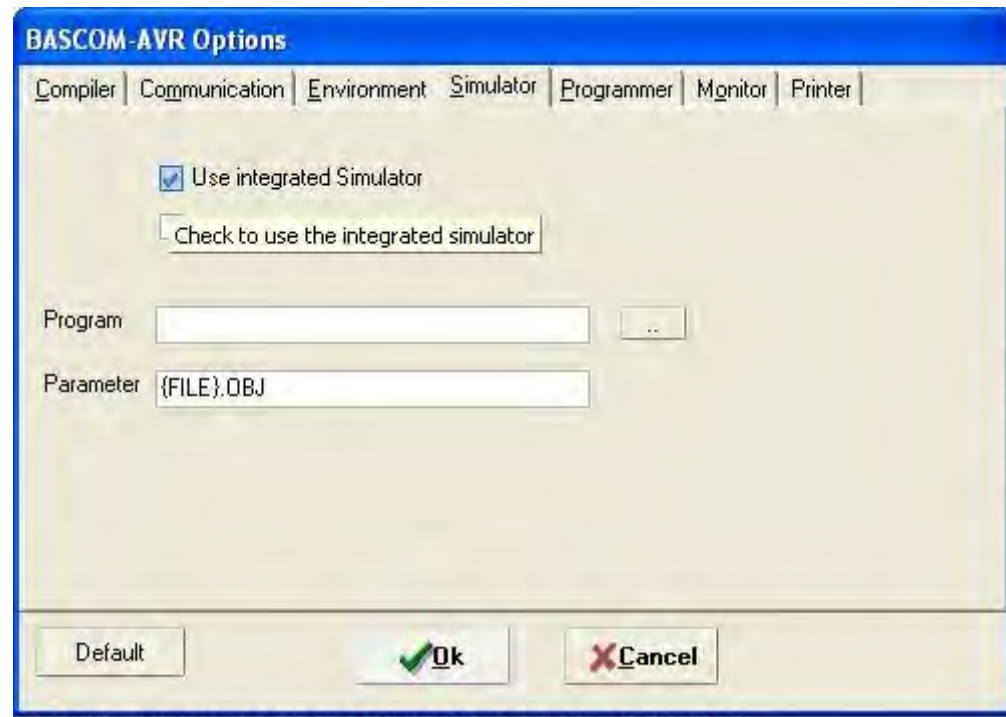


Figure 4.7

You can delete your initial folder and downloaded files you created at the beginning of the installation. There is not much need to keep these on your hard drive taking up space unless you have more storage than you think you will ever use. Be sure to save the copy of the BASCOM AVR manual, though.

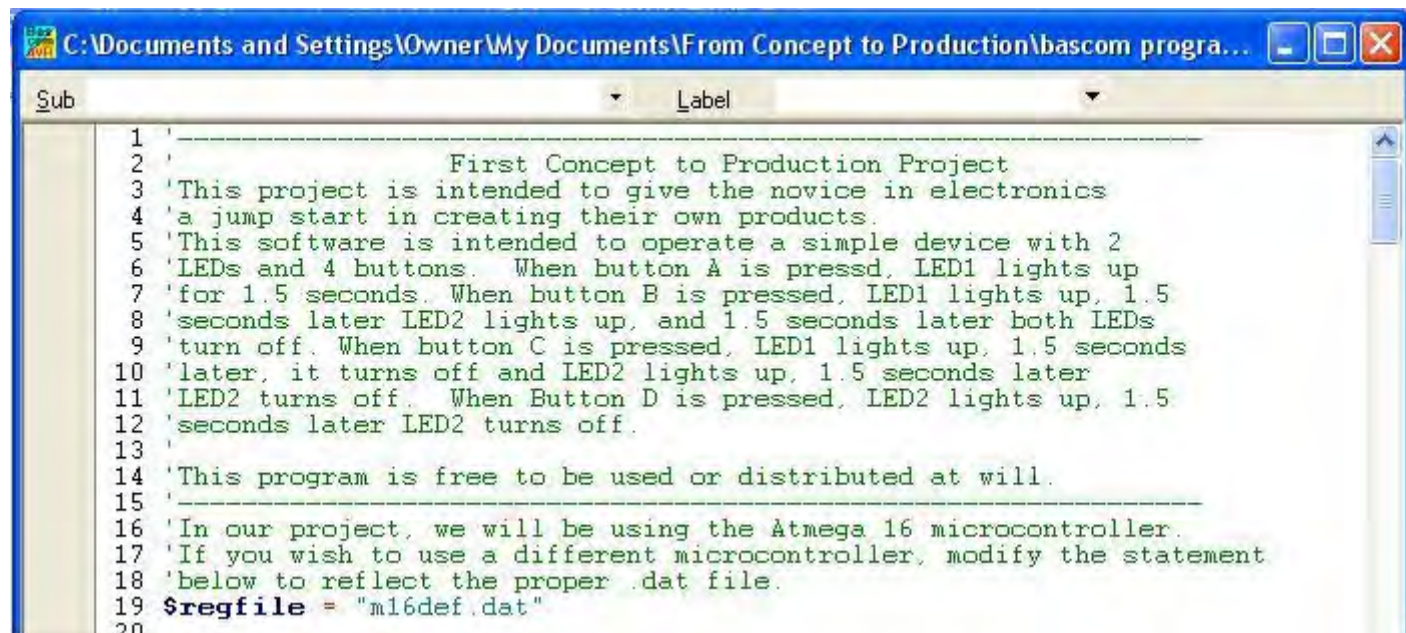
#### *Getting familiar with BASCOM*

I have always learned best by looking at examples and modifying them to suit my purposes. That is how we will show you the basics of programming the AVR microcontroller. As you go through the book, feel free to experiment with the program to get various results. Some will work and some will not. Typically you will learn more from the trials that fail because of the time you spend troubleshooting the reason for the failure. This entire program is listed in the appendix under 'Programs' and also on my website under downloads as project1.bas if you would like to just load the program and continue to the next section.

We will start by opening the BASCOM software you downloaded and installed. Instructions on downloading and installing the software is located in the Appendix under "Software" as well as on MCS Electronic's website.

### *Comments, chip files and clocks*

Now, start a new project and let's start typing some code. Click the blank page button to start a new project. After you have entered a line or two, click File -> Save As and choose a name and location for the project. Saving in the default location is fine, but it is best to create a folder specifically for each project and save the project in that folder.



```
1 '-----  
2 '           First Concept to Production Project  
3 'This project is intended to give the novice in electronics  
4 'a jump start in creating their own products.  
5 'This software is intended to operate a simple device with 2  
6 'LEDs and 4 buttons.  When button A is pressed, LED1 lights up  
7 'for 1.5 seconds.  When button B is pressed, LED1 lights up, 1.5  
8 'seconds later LED2 lights up, and 1.5 seconds later both LEDs  
9 'turn off.  When button C is pressed, LED1 lights up, 1.5 seconds  
10 'later, it turns off and LED2 lights up, 1.5 seconds later  
11 'LED2 turns off.  When Button D is pressed, LED2 lights up, 1.5  
12 'seconds later LED2 turns off.  
13 '  
14 'This program is free to be used or distributed at will.  
15 '  
16 'In our project, we will be using the Atmega 16 microcontroller.  
17 'If you wish to use a different microcontroller, modify the statement  
18 'below to reflect the proper .dat file.  
19 $regfile = "m16def.dat"  
20
```

Figure 5.1

The above figure shows the beginning portion of our program. The lines with a ' in front of them are remarks. They are shown in green by default and are ignored by BASCOM. They are there only for our reference. It is a good idea to put as many comments in the program as possible to insure you know what it was you were doing when you return to the program to debug or modify the code.

The last line specifies the type of chip to BASCOM. The command is the \$regfile = and is in dark blue followed by "m16def.dat" in light blue which is the file. If you are using a chip other than the Atmega 16, you can locate the available reg files by going to Options -> Compiler -> and clicking on Chip. It will bring up a screen for you to see the names of the reg files for the chips as you can see in figure 4.2.

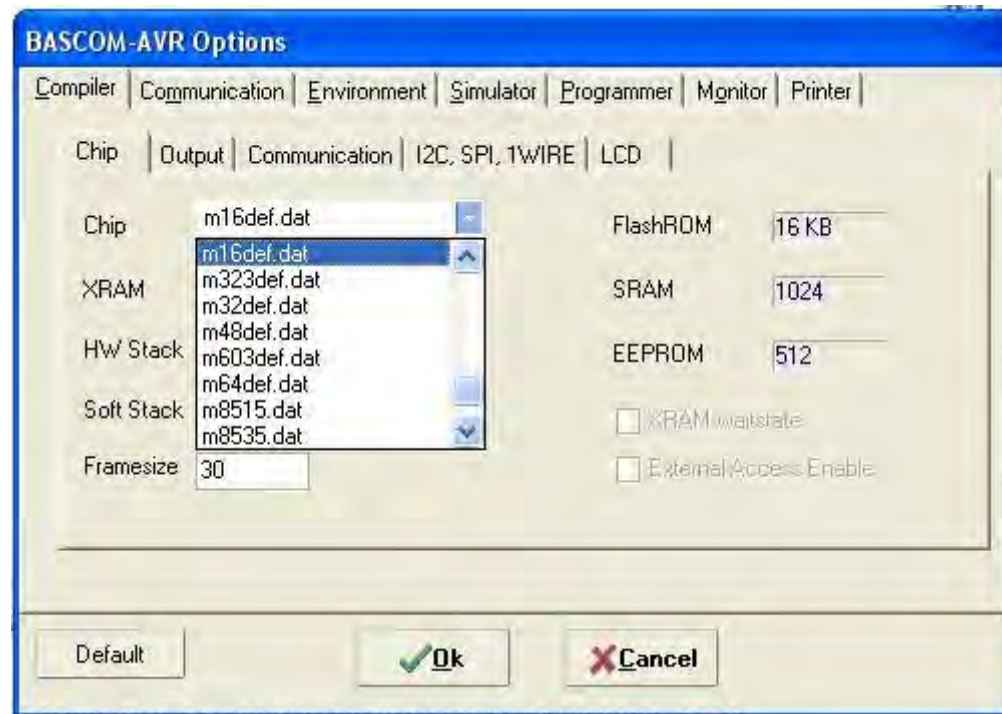


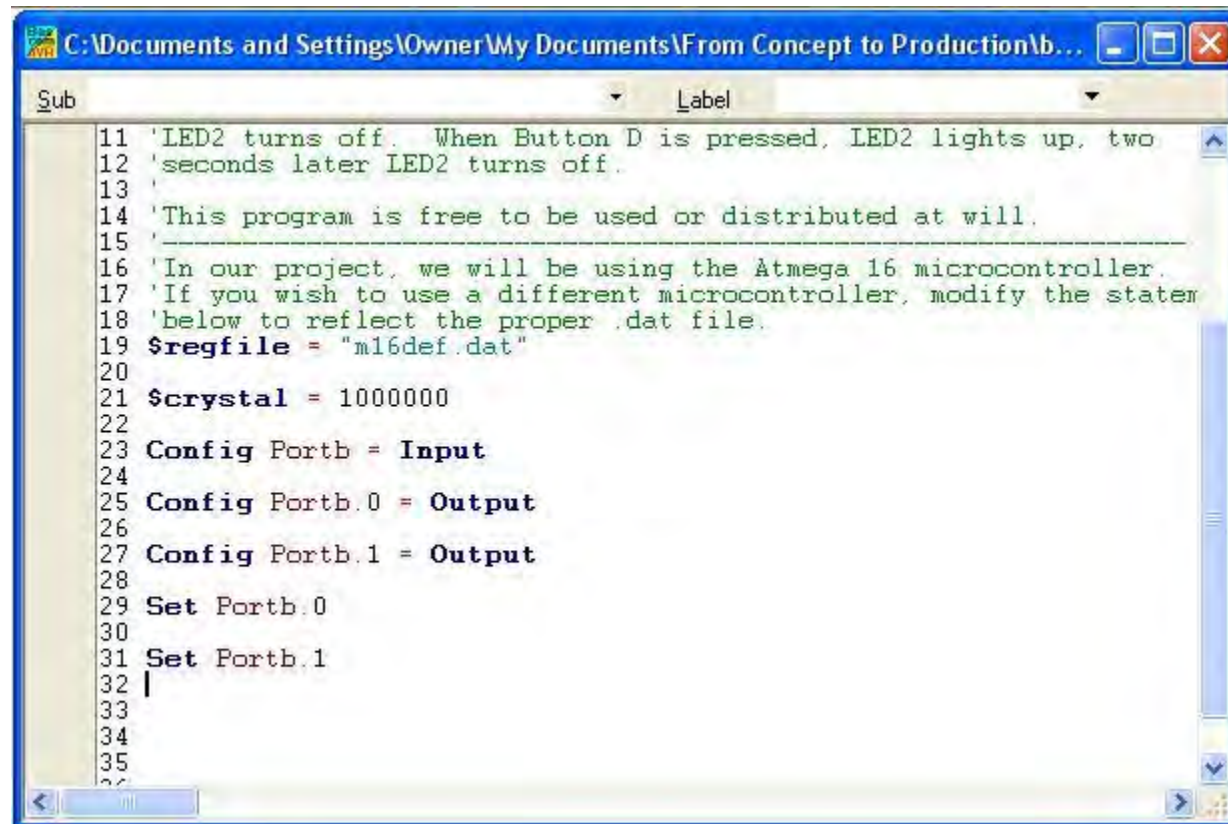
Figure 5.2

Next, you will see the line \$crystal = 1000000. This tells the software what frequency the chip is using. This is important for any application that times any event, such as communication or waiting 2 seconds before turning an LED on or off. If the line in the code is set incorrectly, for instance if you told the software the frequency was 8000000, instead of waiting 2 seconds to turn an LED on or off, it would wait 16 seconds because it thinks the system clock is ticking 8000000 times per second instead of 1000000 times per second so it will wait until 16000000 ticks have passed before carrying out the command.

The Atmega 16 comes set from the manufacturer to use the internal RC oscillator (system clock) set at 1000000 (1mhz). It can be changed to 8mhz with the STK500, but the line in the code would have to be changed to 8000000 to match the actual clock frequency. Directions on changing the internal oscillator is in the appendix under 'Programming the Fuse Bits.'

### *Setting ports and pins*

Now we need to configure the individual input/output pins to do perform the correct functions. First, we tell the microcontroller if the pins are going to be input (receive information) or output (send information) pins. You can choose to set an entire port (group of pins with the same letter prefix, i.e. A.0, A.1, etc), you can set one pin at a time, or you can do a combination of the two. We will do a combination of the two to illustrate how each is done.



```
C:\Documents and Settings\Owner\My Documents\From Concept to Production\l...
Sub                               Label
11 'LED2 turns off.  When Button D is pressed, LED2 lights up, two
12 'seconds later LED2 turns off.
13 '
14 'This program is free to be used or distributed at will.
15 '-----
16 'In our project, we will be using the Atmega 16 microcontroller.
17 'If you wish to use a different microcontroller, modify the state
18 'below to reflect the proper .dat file.
19 $regfile = "m16def.dat"
20
21 $crystal = 1000000
22
23 Config Portb = Input
24
25 Config Portb.0 = Output
26
27 Config Portb.1 = Output
28
29 Set Portb.0
30
31 Set Portb.1
32 |
33
34
35
36
```

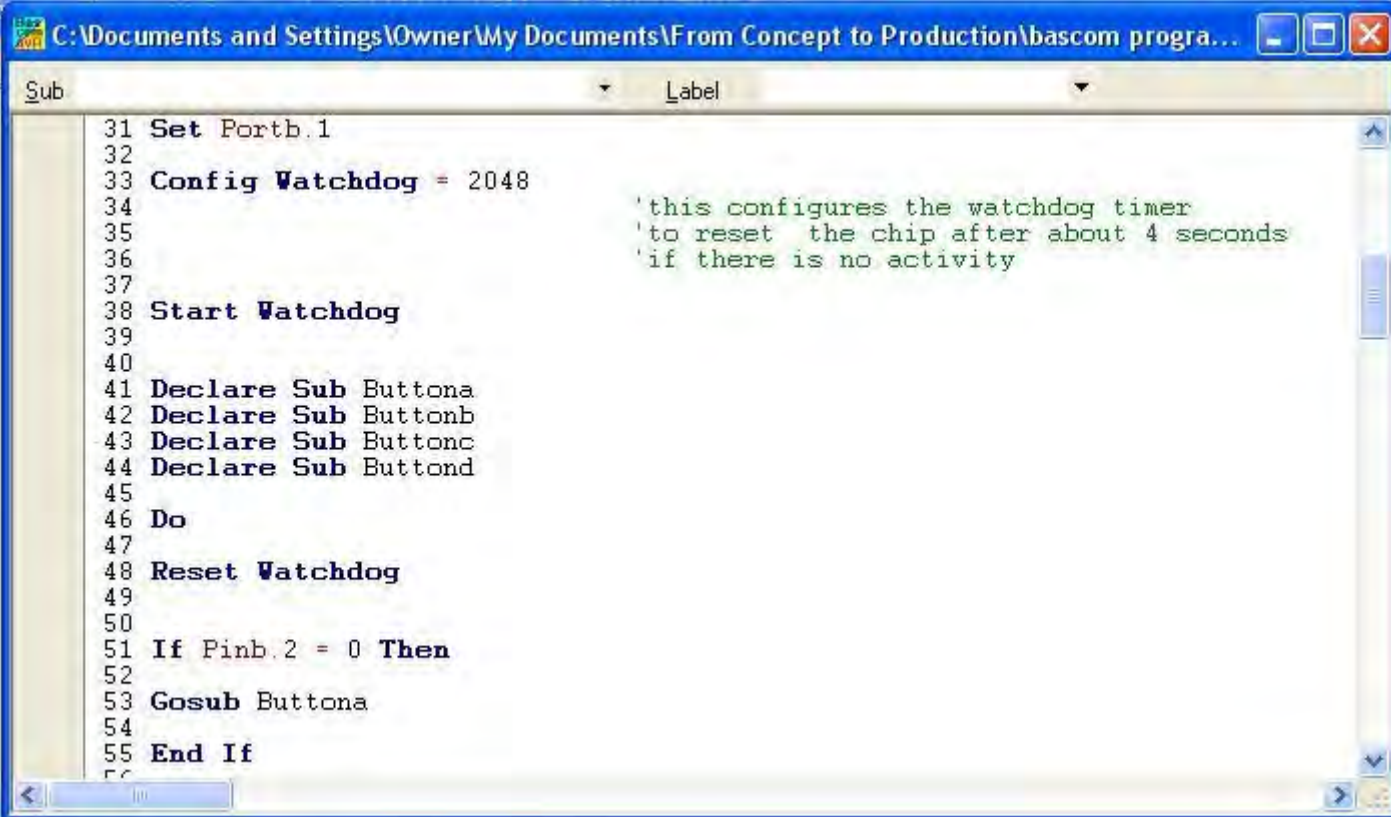
Figure 5.3

Figure 5.3 shows the command `Config Portb = Input`. `Config` is the command given to the compiler, `Portb` is the set of pins (b) being addressed, and `Input` determines the direction of the pins (receive). That is how you set an entire port of pins to one direction.

However, we wanted two of the pins in `Portb` to be output pins to drive the LEDs. So, the next two lines change just the two pins `Portb.0` and `Portb.1` to output pins to drive the LEDs. After the pins are configured as output pins, the next two lines set the pins to positive 5 volts (this chip's operating voltage) to insure the LEDs are off as the negative side of the LEDs will be tied to the microcontroller through a resistor (the negative side must be tied to a ground source with the positive side tied to a positive voltage for the LED to illuminate). If the positive side is tied to a positive source and the negative side is tied

to a positive source, the LED will not illuminate). The command SET drives the pin high – positive – while the RESET command drives the pin low – negative.

*Watchdog timers, subroutines and do loops*



```
31 Set Portb.1
32
33 Config Watchdog = 2048
34
35 'this configures the watchdog timer
36 'to reset the chip after about 4 seconds
37 'if there is no activity
38 Start Watchdog
39
40
41 Declare Sub Buttona
42 Declare Sub Buttonb
43 Declare Sub Buttonc
44 Declare Sub Buttond
45
46 Do
47
48 Reset Watchdog
49
50
51 If Pinb.2 = 0 Then
52
53 Gosub Buttona
54
55 End If
```

Figure 5.4

This section of code illustrates five functions you will likely use in most of your programs. The first is the watchdog timer. The lines pertaining to the timer are obvious as they all have `watchdog` in the statement. As you can see, you need to initialize the timer with the `config` statement along with the length of time you want it to wait until it resets the microcontroller. After that you start the timer with the `start` command. Finally, you will place reset commands throughout your code to reset the timer preventing it from resetting your chip.

It seems like a lot of hassle to add a function you do not want to engage. However, without the watchdog timer, if your program locks up due to software, hardware, static electricity, or whatever, you don't want it to stay locked up or for the user to have to reset the product. That is the function of the watchdog timer. If your program locks up, the reset command for the watchdog will not reset the timer and it will count down to zero. At this point, the chip completely resets itself and starts running the program from the beginning. Just make sure when you are programming the time delay for the watchdog timer and inserting reset statements, you have enough of a delay and reset statements to keep from resetting the chip during normal operation (if we have a 2 second delay on the LED command, you would not set the timer at 1 second – 1000 milliseconds – or the chip would reset every time the LED started its cycle).

Following the watchdog timer you see several declare sub commands. We will be using subroutines to carry out functions in the event a button is pressed. As you can see, the subroutines are named according to the corresponding buttons we will assign for the function. This enables easier reading of the code at a later date. Every subroutine must be declared before it can be called.

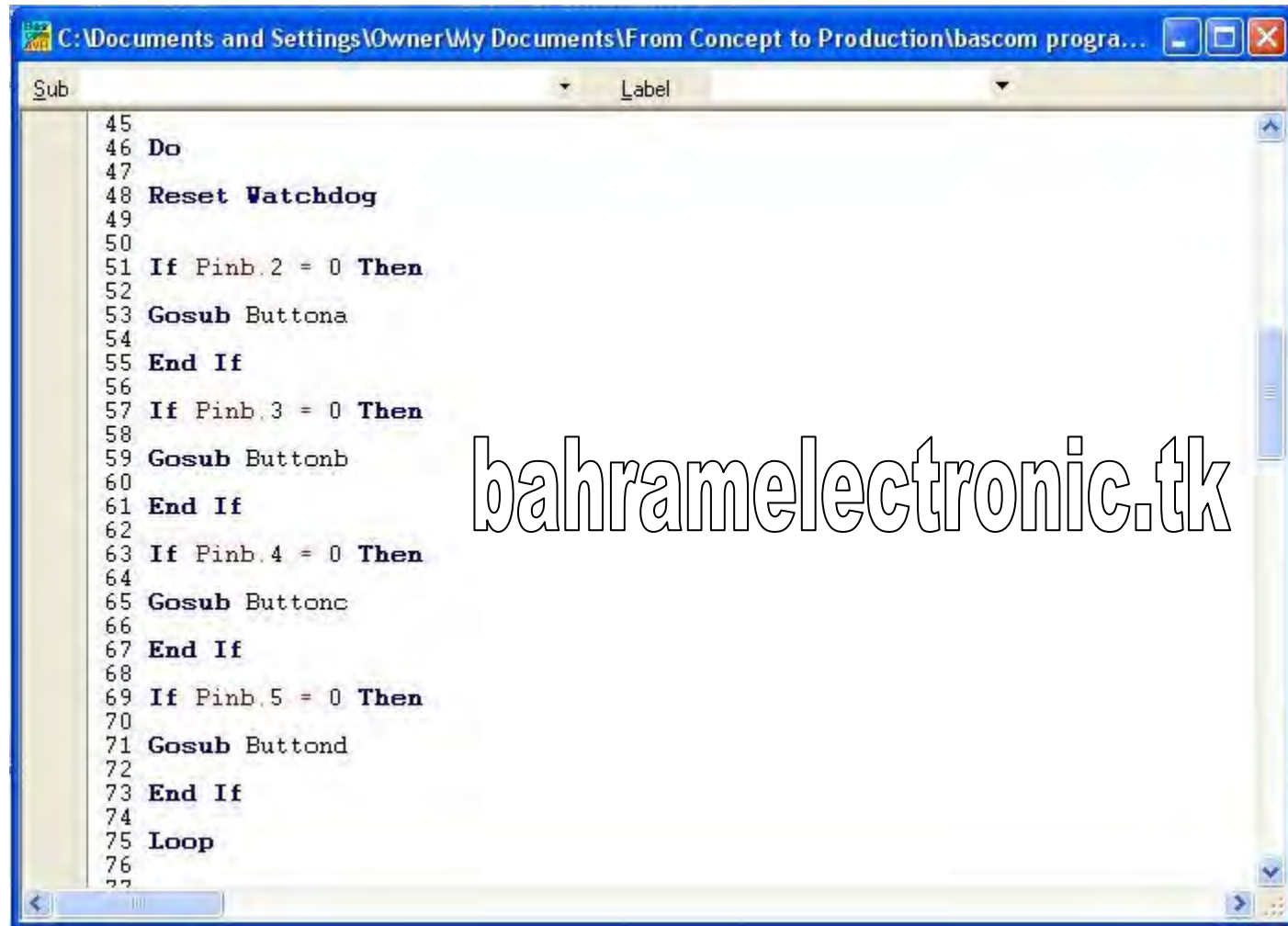
After the declare sub commands, you see the `Do` command. This is half of a whole command. The other half goes at the end of our main body of the program to keep the program looping, waiting for a button to be pressed.

#### *If then statements*

Following the `Do` command, you see the first of many reset watchdog commands. After that command is an if-then statement. Each if-then statement has at least three parts. The first part is the condition. In this case, the condition is `Pinb.2 = 0` which means button a is pressed and the pin is tied to ground.

If the button is pressed, the program executes the commands inside of the if-then statement (all commands before the matching end-if command as you can see in the example code in figure 4.4). The command executed in this case is to execute the subroutine `Buttona` with the command `Gosub Buttona`. At that point, the program jumps to subroutine `Buttona` and executes the commands in that portion of the program. You will see the subroutine soon.

If button A is not pressed, `Pinb.2 = 1` and the program skips to the end of the if-then statement which is the end-if statement and begins executing the next line of code following the if-then group.



```
45
46 Do
47
48 Reset Watchdog
49
50
51 If Pinb.2 = 0 Then
52
53 Gosub Buttona
54
55 End If
56
57 If Pinb.3 = 0 Then
58
59 Gosub Buttonb
60
61 End If
62
63 If Pinb.4 = 0 Then
64
65 Gosub Buttonc
66
67 End If
68
69 If Pinb.5 = 0 Then
70
71 Gosub Buttond
72
73 End If
74
75 Loop
76
77
```

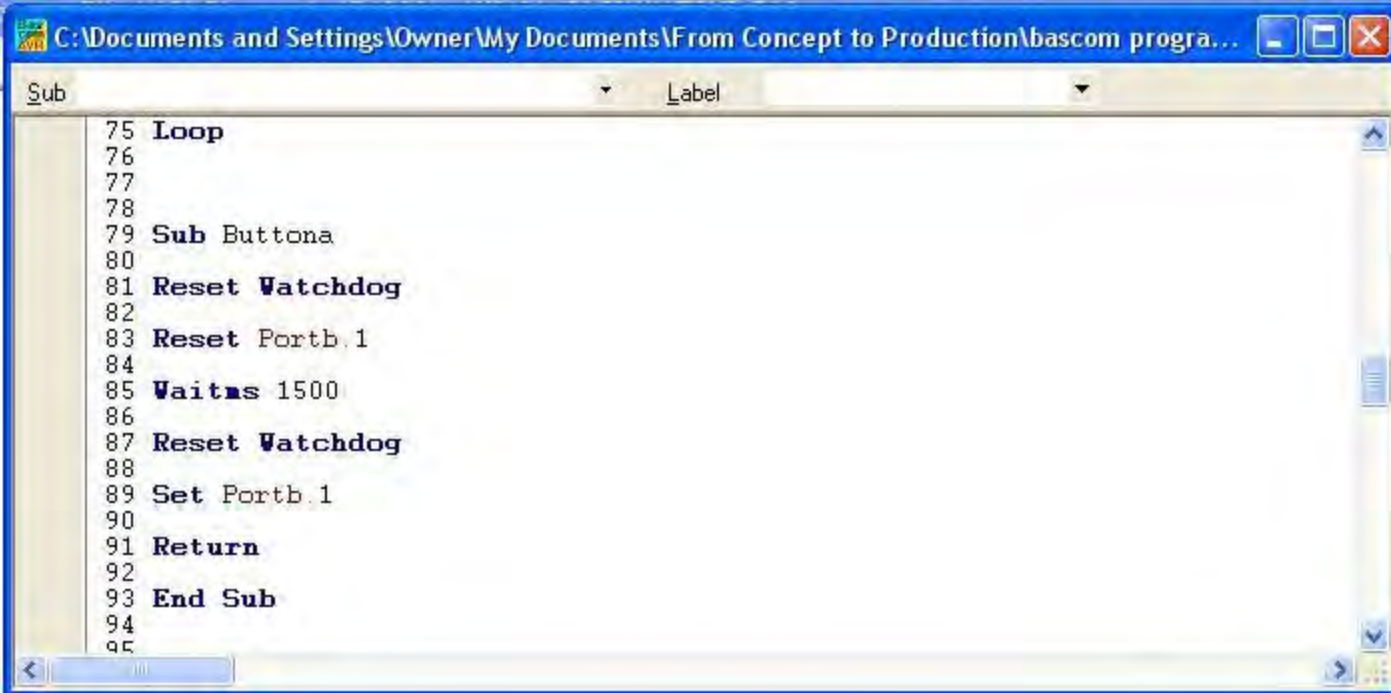
bahramelectronic.tk

Figure 5.5

In figure 5.5 we see the last three if-then – end-if statements, each calling the subroutine for the button assigned to each pin for a total of four statements responding to four buttons. That takes care of the input portion of our product in the software.

The last statement you see is the other half of the Do-Loop statement, Loop. The microcontroller will continue to go in circles within this loop unless power is removed, one of the if-then statements is executed, or the watchdog timer resets the chip and the whole process starts over again.

*Subroutines (continued)*



```
Sub
75 Loop
76
77
78
79 Sub Buttona
80
81 Reset Watchdog
82
83 Reset Portb.1
84
85 Waitms 1500
86
87 Reset Watchdog
88
89 Set Portb.1
90
91 Return
92
93 End Sub
94
95
```

Figure 5.6

In figure 5.6 we see the first subroutine for button A. When the if-then statement calls this subroutine, the microcontroller jumps to this point and starts with the first line of code, reset watchdog, and continues through until it reaches the Return statement which causes the microcontroller to return to the point in the program it jumped from to get to this subroutine.

After the reset watchdog command, you see another reset command for Portb.1. That makes Portb.1 go low or negative, allowing electricity to flow through the LED causing the LED to illuminate.

The next statement makes the program pause for 1500 milliseconds, or 1.5 seconds. It is a waitms command. Another command to create a pause in your programs is the wait command. If you typed wait 3 , the program would pause 3 seconds before continuing to the next command.

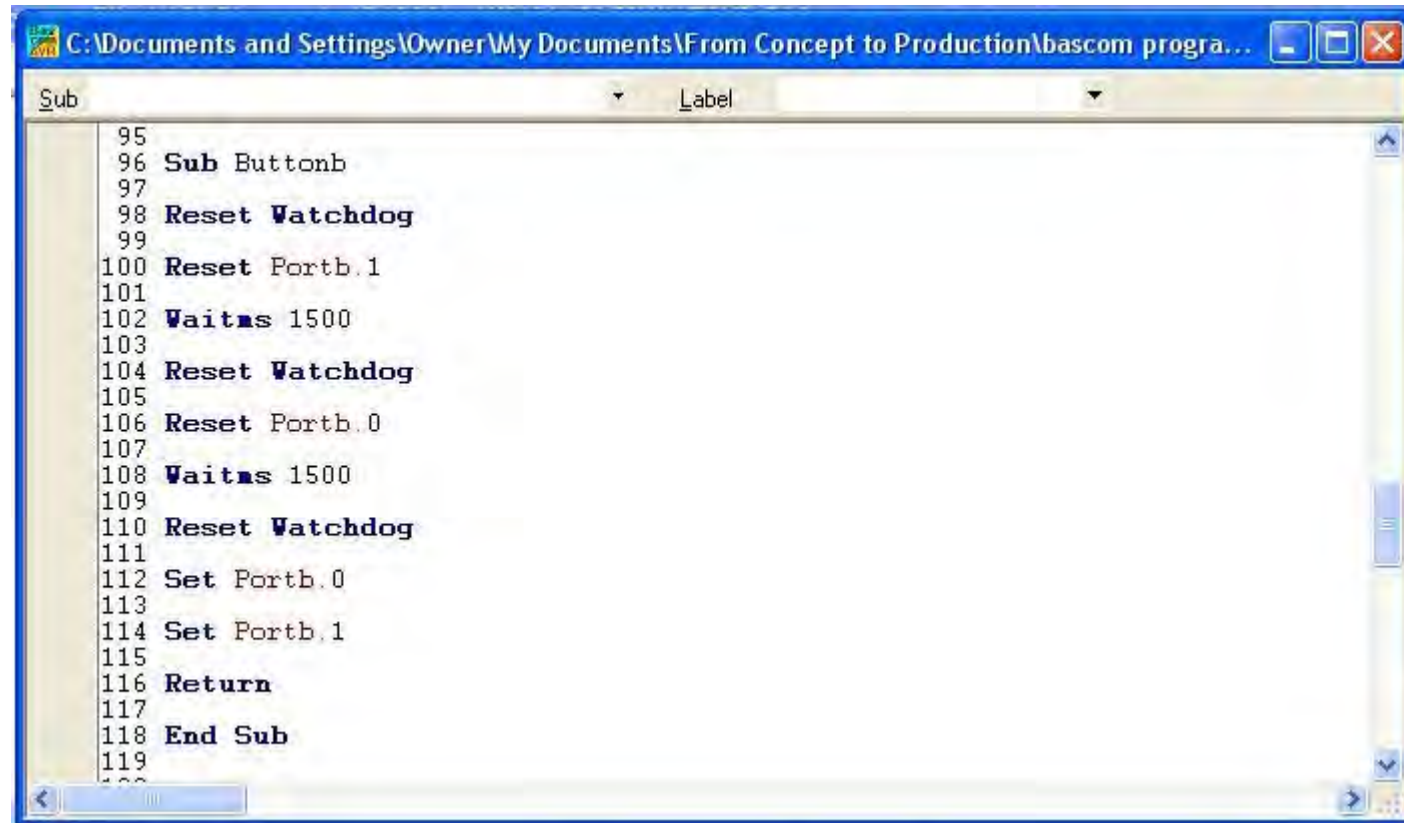
After the waitms 1500 command, you see another reset watchdog command. It is important to insure the watchdog timer is continually reset before it resets the microcontroller or your program will not function properly.

Following the watchdog reset, Portb.1 is once again set high, positive, stopping the flow of electricity.

The next line you see is Return which instructs the microcontroller to return to the main loop at the point it left. The last line is the End Sub line, letting BASCOM know the subroutine is finished.

So the order of the subroutine is pretty simple.

1. enter the subroutine
2. reset the watchdog
3. make portb.1(LED1) negative so the LED shines
4. wait 1.5 seconds
5. reset the watchdog
6. make portb.1(LED1)positive so the LED goes dark
7. return to the main program
8. let BASCOM know the subroutine is over



The screenshot shows a Windows Notepad window with the following assembly code:

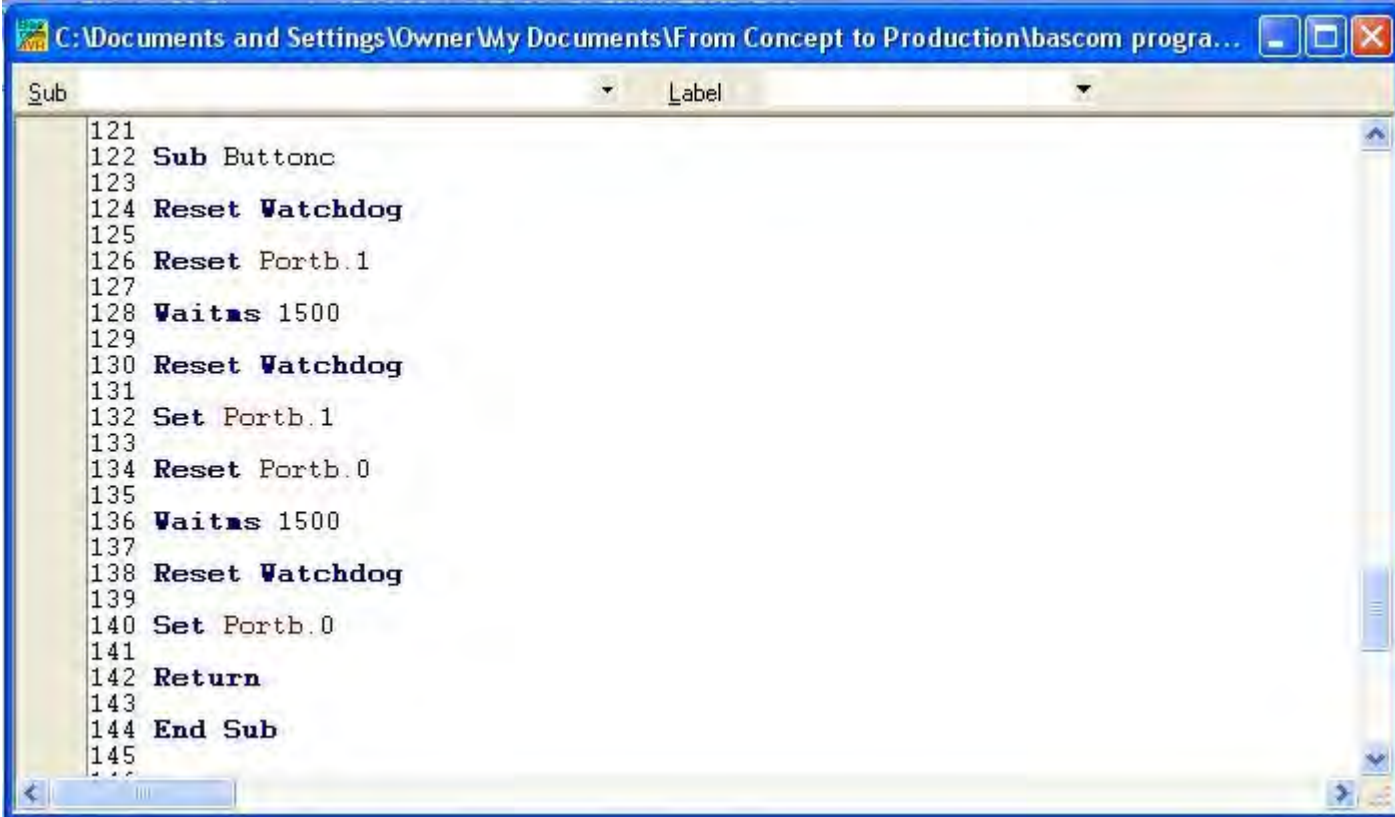
```
95  
96 Sub Buttonb  
97  
98 Reset Watchdog  
99  
100 Reset Portb.1  
101  
102 Waitms 1500  
103  
104 Reset Watchdog  
105  
106 Reset Portb.0  
107  
108 Waitms 1500  
109  
110 Reset Watchdog  
111  
112 Set Portb.0  
113  
114 Set Portb.1  
115  
116 Return  
117  
118 End Sub  
119
```

Figure 5.7

The order of Buttonb subroutine in figure 5.7 is similar.

1. enter the subroutine
2. reset the watchdog
3. make portb.1(LED1) negative so the LED shines
4. wait 1.5 seconds
5. reset the watchdog timer
6. make portb.0(LED2) negative so the LED shines

7. wait 1.5 seconds
8. reset the watchdog
9. make portb.1(LED1) and portb.0(LED2) positive so the LEDs goes dark
10. return to the main program
11. let BASCOM know the subroutine is over



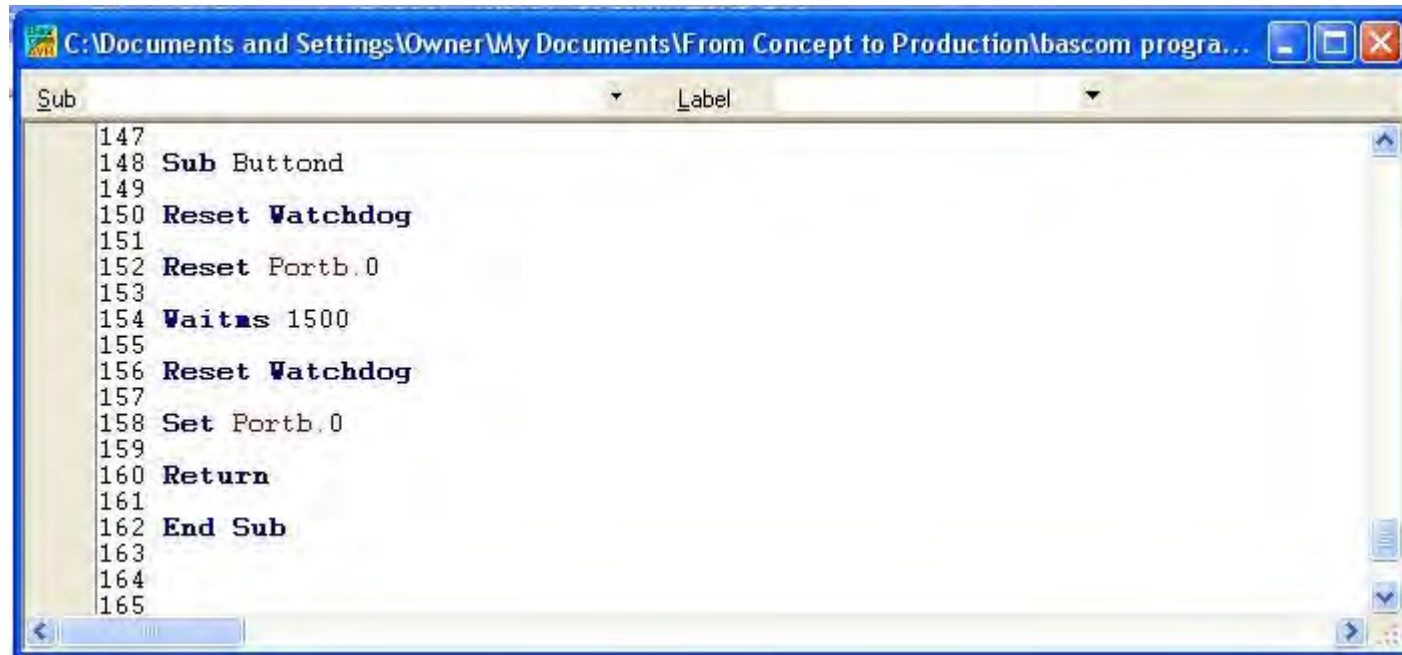
```
Sub
Label
121
122 Sub Buttonc
123
124 Reset Watchdog
125
126 Reset Portb.1
127
128 Waitms 1500
129
130 Reset Watchdog
131
132 Set Portb.1
133
134 Reset Portb.0
135
136 Waitms 1500
137
138 Reset Watchdog
139
140 Set Portb.0
141
142 Return
143
144 End Sub
145
```

Figure 5.8

Subroutine Buttonc is shown in figure 5.8 with the following order.

1. enter the subroutine
2. reset the watchdog
3. make portb.1(LED1) negative so the LED shines
4. wait 1.5 seconds
5. reset the watchdog timer
6. make portb.1(LED1) positive so the LED goes dark
7. make portb.0(LED2) negative so the LED shines
8. wait 1.5 seconds
9. reset the watchdog timer
10. make portb.0(LED2) positive so the LED goes dark
11. return to the main program
12. Let BASCOM know the subroutine is over

bahramelectronic.tk



The screenshot shows a window titled "C:\Documents and Settings\Owner\My Documents\From Concept to Production\bascom progra...". The window contains a BASIC program editor with a "Sub" dropdown menu and a "Label" dropdown menu. The code is as follows:

```
147  
148 Sub Buttond  
149  
150 Reset Watchdog  
151  
152 Reset Portb.0  
153  
154 Waitms 1500  
155  
156 Reset Watchdog  
157  
158 Set Portb.0  
159  
160 Return  
161  
162 End Sub  
163  
164  
165
```

Figure 5.9

So the order of the subroutine is pretty simple.

1. enter the subroutine
2. reset the watchdog
3. make portb.0(LED2) negative so the LED shines
4. wait 1.5 seconds
5. reset the watchdog
6. make portb.0(LED2) positive so the LED goes dark
7. return to the main program
8. let BASCOM know the subroutine is over

*Compiling*

When you have typed in all of the code (or copied it from my website), compile the code in BASCOM. To do that, just press the compile button as shown in figure 5.10.



Figure 5.10

If you happen to have the simulator active, a new screen simulating the chip running with the program in the background will appear. To exit that screen, just close out the simulator window.

If the program does not successfully compile, you will get a list of errors at the bottom of the BASCOM window. You can get an idea of what you did wrong by looking at the errors and then finding the corresponding lines in your code. Many errors you encounter will be typing errors. Check your program against the copy in this book's appendix and try compiling the program again until you are successful.

<http://bahramelectronic.tk>

این مطلب توسط وبلاگ بهرام الکترونیک به PDF تبدیل شده است .