SINGAPORE POLYTECHNIC
SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

ET0104 Embedded Computer Systems Laboratory

Laboratory 1 Introduction to the Laboratory set up

Objectives

- To learn how to work with the Single Board Computer (SBC) and a PC.
- To gain familiarity with the Input/Output (I/O) board by writing a timer program.
- To learn how to read and modify memory on a SBC
- Learn how to create a project in Visual C++ and download it to a SBC.

Introduction

The laboratory set up comprises of:

i) A PC running Visual C++ development environment. This environment has to be set up to work with an embedded system. We are using the Embedded ToolSuite(ETS) developed by Venturcom. This combination of software will be referred to as VC/ETS. The software is linked to a development board via a parallel port connection.

ii) A PC104 Single Board Computer System. (We will refer to this board as the SBC). The SBC has all the facilities of a desktop computer. A larger list of its capabilities can be found in the appendix.

The main interface a user has is the Reset button, to reset the system. It is hoped you do not use this button much! Note that it is possible to connected a computer monitor and keyboard to the SBC, but that will not be done here.

iii) A custom designed I/O (Input Output) board. This interfaces to the SBC through the PC140 bus interface to several pieces of hardware.

The I/O board allows a user to easily address up to 8 hardware devices in a 64K memory space. The board will be considered in more detail later.

iv) Various I/O devices, like keypads, stepper motors, loudspeakers may be attached through connectors to this board.

The main interface a user has is the Main Reset button, to terminate a user program.

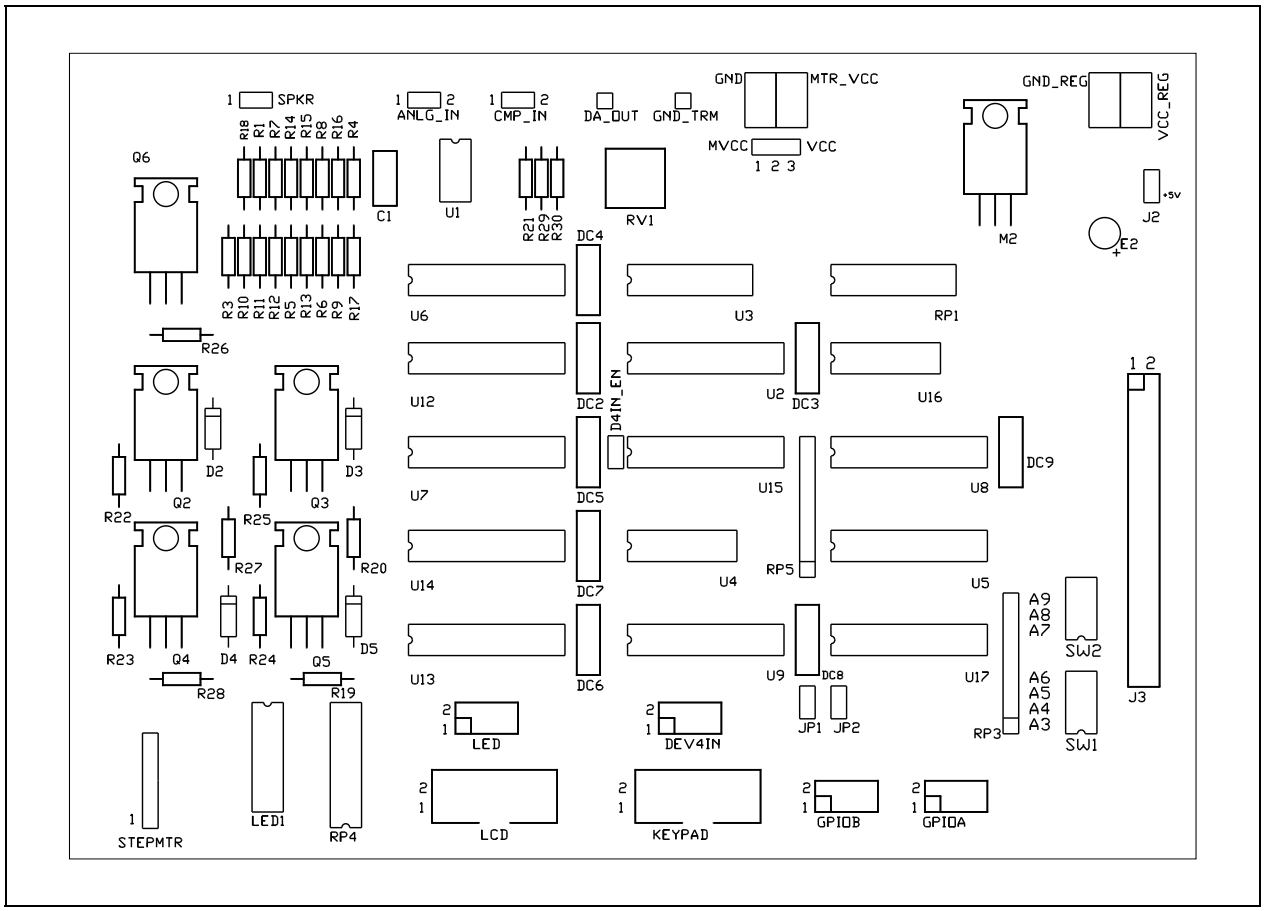
iii) A custom designed I/O (Input Output) board.This interfaces to the SBC through a 40 pin cable and effectively extends the Address Bus, Data Bus and most of the Control signals, to several pieces of hardware.

The I/O board allows a user to easily address up to 8 hardware devices in a (2^{10}) 1024K memory space. The board will be considered in more detail later.

iv) Various I/O devices, like keypads, stepper motors, loudspeakers may be attached through connectors to this board.

The I/O board

This board has 4 main areas of interest.



Layout of the I/O board

First is the I/O connector, designated J3, located on the right edge of the I/O board. This 40 pin I/O adapter cable connects all the Address and Data Busses and most of the control signals available from the PC/104 bus.

The I/O adaptor cable is actually made up of a cable and a circuit board. This board uses a 40 pin subset of the 104 connector pins available from the SBC. It also has an onboard 5 volt voltage regulator.

Second is the bank of two decoding switches, designated SW1 and SW2. These are located to the right of the board as well. They allow the top 6 address lines to be connected to a 74138 decoder on board. This in turn, will allow the user to access the 8 devices on board. We will consider them in greater detail later.

Third are the various buffers and latches. The board does have a single 7 segment LED for users to easily check out their programs initially. Fourth are the I/O connectors at the bottom edge of the

board. These will be used to connect to various hardware devices. For this course, you will only be using those marked “STEPMTR”, “KEYPAD” and “LCD”. The other header pins will not be used.

Last of all are the analogue circuitry located at the top left of the board. This consists of a Digital to Analog Converter (DAC) made up of an R-2R resistor bank, fed from a 74LS244 buffer chip. This signal is buffered through an op-amp and is available as an Analog output marked “DA_OUT”. This may be connected to a speaker, which is buffered by a transistor. The DAC may be used as an integrating Analog to Digital Converter (ADC). The variable resistor RV1 allows a user to manually control an input signal for this purpose.

There are 8 hardware devices set up as follows. Most are latched output devices.

Device 0 - Digital to Analog network
Device 1 - Device not used
Device 2 - LED
Device 3 - Liquid Crystal Display (LCD)
Device 4 - Lower 4 bits output to keypad / Higher 4 bits input from keypad
Device 5 - Stepper motor
Device 6 - Device not used
Device 7 - General Purpose Output

For this module, you will develop programs to control devices through this board, on a desktop PC. After that, you will download the program to the SBC, which will then interface to the I/O board.

The overall block diagram is shown below:

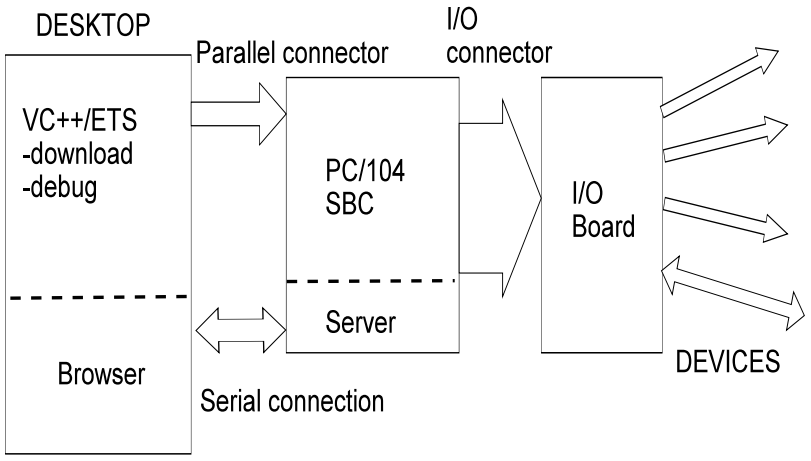


Figure 2 Block diagram of lab setup

Software setup

We are using Visual C++ (VC) which is part of Microsoft Visual Studio. As mentioned earlier, we develop programs on a desktop to download to the PC/104 SBC. The desktop PC running the Windows operating system, has generous resources to perform many tasks, often simultaneously.

In order to provide some functionality to the SBC with its limited resources, we have elected to use the Embedded ToolSuite (ETS) software developed by Venturcom. We will be using ETS version 12.0. Thus we will use VC in a rather "non-standard" way. The functions are accessed through a toolbar. Briefly they allow the user to:

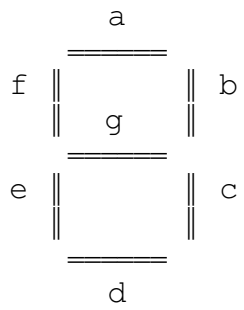
i) View and modify the memory and I/O values of the SBC
ii) Automatically download the program to the SBC
iii) Control running of the program on the SBC *from* the desktop.
iv) Set up a VC project easily, with all the required object files, libraries and compiler options. This can be a very involved step.

We will look at this in detail later.

Assignment

To gain some familiarity with the system, we will construct an LED timer which will update the time every 1 second.

First let us review the operation of a seven segment LED. This is a device made up of seven Light Emitting Diodes arranged so that when lighted appropriately, characters and numbers may be displayed. For example, the number "0" may be displayed by lighting up segments a to f. By showing the characters "A" to "F" as well, we can display one hexadecimal number. This is half a byte, or a nybble.



It is a well known fact that digital circuits in general are able to *sink* more current than they can *source*. An LED normally needs about 20 mA of current flowing through it before it lights up. Most TTL compatible circuits can sink this amount of current. A simple schematic is shown below of ONE LED segment. Thus to turn *on* this LED, we output a *zero* level voltage.
If connect each LED segment to a latch, we are able to display characters by outputting an appropriate byte to the latch. Let us utilise the following convention that latch pins 6,5,4,3,2,1,0 are mapped to segments g,f,e,d,c,b,a respectively. Complete the bit patterns for the following numbers. Remember, a "0" turns the segment ON.

0: _____ 1: _____ 2: _____ 3: _____

4: _____ 5: _____ 6: _____ 7: _____

8: _____ 9: _____ A: _____ b: _____

C: _____ d: _____ E: _____ F: _____

Since only the first four bits of a byte is used, we will need to "mask off" that is, set the high four bits to zero.

Secondly, we will use a system function which depends on a hardware timer to implement the delay.

Completing the program

Look at the program template at the appendix.

1. Using appropriate values, *fill in the blanks* for the array Bin2LED.

2. We will use the Sleep() function which provides a "nonblocking" delay of 1 millisecond. This function is available through Windows. This is reflected in the line: `#include <windows.h>`
Unlike other kinds of delay, this function:

i) Does not cause the CPU to execute a software loop which is inefficient.
ii) Does not "block" other programs from running, allowing multiprogramming.
iii) Uses a hardware timer for better precision.
iv) Gives a *minimum* delay time only, depending on other programs running.

Fill in the appropriate value for Sleep() in lab1.C.

3. Do not compile your program yet. First test out the LED values.

Setting up the hardware

Make sure the power supply to the SBC is turned off.

We need to set up the I/O board so it can detect the address of the LED latch. Set the switches as follows. Setting a switch to "Off" places a logic "1" at the signal.

Note: Please use only your finger tips - do NOT use pens, or other sharp objects which may break off and leave debris on the I/O board.

Address	A9	A8	A7	A6	A5	A4	A3
DIP setting	Off	Off	On	On	Off	Off	On

What this means is that for DIP switch block SW2, set the switch marked "A9" to the "Off" position. The same procedure applies for the other switch blocks SW1.

Now turn on the power supply to the SBC.

Testing the setup

Note about command options in VC:
In Windows, menu options may be selected by clicking on the mouse, or using accelerator keys, if a character is underlined. If there is an available function key, it will be put in brackets ( ) next to the command.

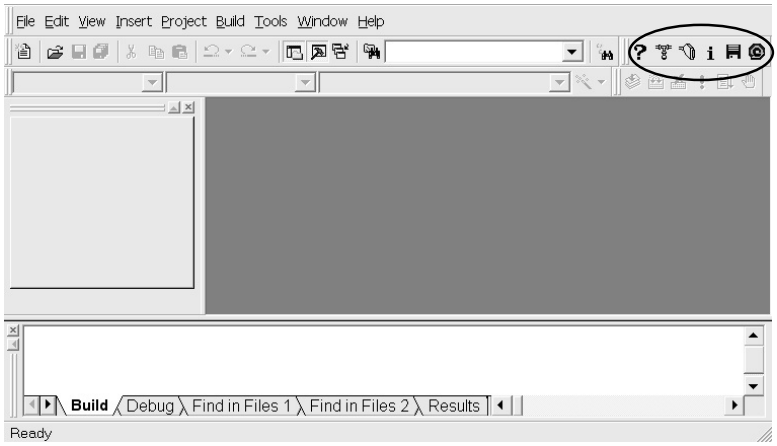
Using the Port Input/Output feature

We will use VC/ETS software to test our LED values without the use of a program. The screens for the most commonly occurring situations are shown. Depending on your set up, you may see slightly different screens. Select the options that will most likely allow you to progress in the lab. If in doubt, please ask!

If needed, turn on your PC and go into Windows. Start the Microsoft Visual C++ environment.

START button> Programs > Microsoft Visual Studio 6.0 > Microsoft Visual C++ 6.0

In the Microsoft Visual C++ Program, look for the ETS toolbar as shown below.



This toolbar is dockable, that is, it can be moved. A detailed view is shown below.

By hovering your mouse over the icons, in the toolbar, you will get a "tooltip" description of the function of the icons.

1. Click on the "Target Port Input/Output icon as shown below:

Figure 4 Target Port I/O Icon

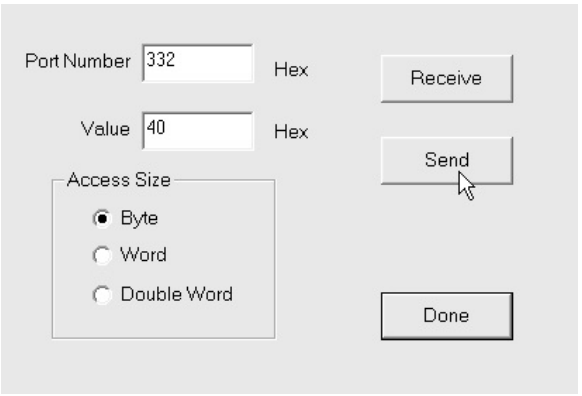


Figure 5 Target Port I/O Dialog

This will bring up the Target Port Input/Output dialog box:

2. Enter the Port number as 332 and Value as 40 and click the Send button. You should see 0 on the LED display. Do this for as many values as needed. All values are in hexadecimal.

Note that you can also *input* a value, but of course, to an appropriate device. Remember, you are using the desktop PC to *control* the SBC, through the parallel connection.

Creating a project

In VC, all program development is managed through a *project*. All files needed to create a program need to be defined to the project. This is similar to the *make* utility used in earlier programming projects. The advantage of doing this is that all compiler settings, libraries, linker options can be automatically set up, remembered and maintained. Another advantage is that when recompiling, only those files *which have changed* will be involved in recompiling. This can be a huge savings in time.

We will create a project for lab1 and modify a C program in that project. We will use the Visual System Builder which can be accessed by the ETS Toolbar. This will create the necessary files for us. *It is important* to do this, otherwise we will not be able to connect properly to the SBC.

1. Click on the *Visual System Builder* icon.

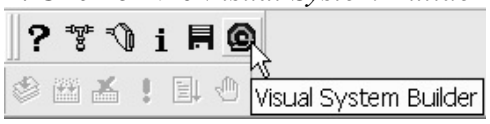


Figure 6 Visual System Builder

2. Use ***New*** to create a new project using the Visual System Builder. You may use this to open an existing project and modify settings later.

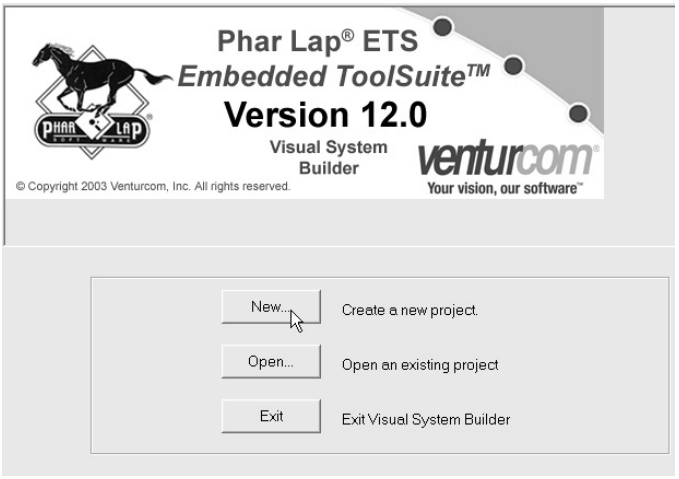
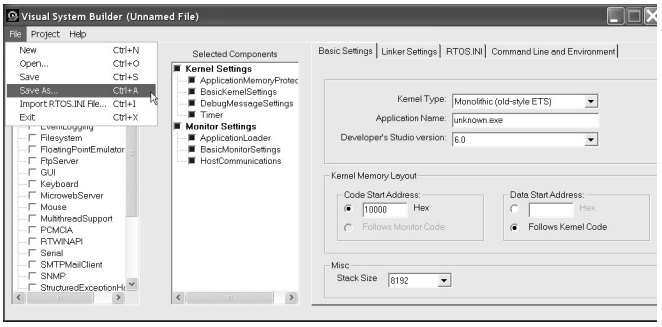


Figure 7 Step 1 in the project creation process

3. Then ***File Save As*** to create a VSB and workspace name of ***lab1*** in the directory ***\ECSLAB\lab1***.



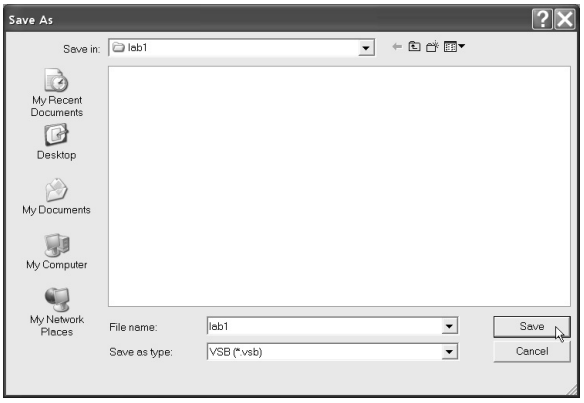
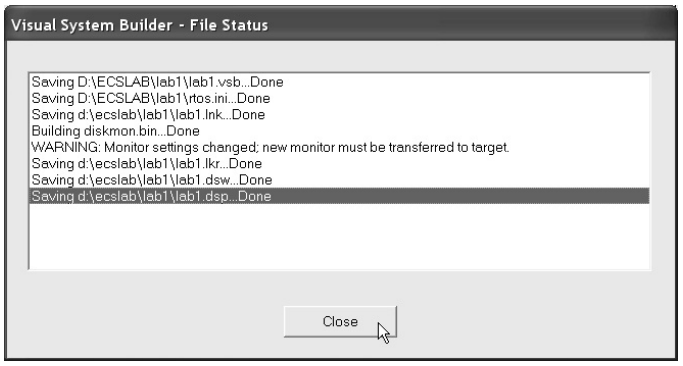


Figure 9 Step 2 in the project creation process

Use the file browse button if needed.

3. The required files will be built for you. A file status report will be shown, similar to the figure below. Click ***Close***.

4. Finally, ***File Exit*** Visual System Builder.



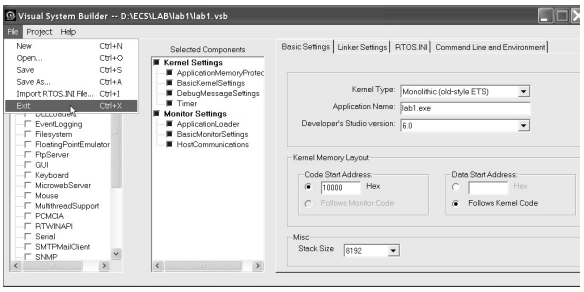
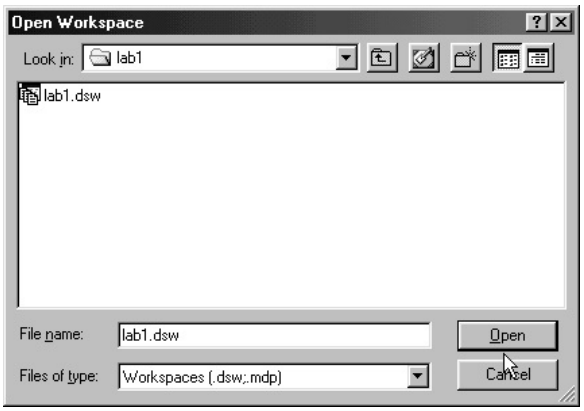


Figure 10 Exit VSB

Modifying the project files

We will now add in a C program to the lab1 project. The files contained in the project are accessed through a workspace.



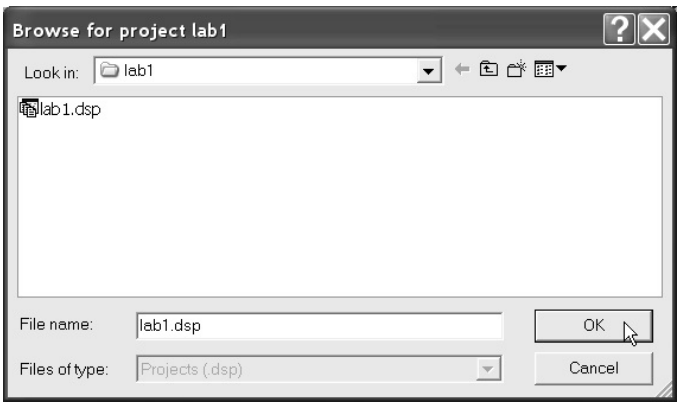


Figure 12 Open project

6. Open the workspace by: ***File*** > ***Open Workspace***. Use the file browser to the subdirectory ***/ECSLAB/lab1***. Select ***lab1.dsw***. Bring it in by ***Open***ing it

7. *If needed*, you will be asked for an associated *.dsp*. Click OK to open this file.

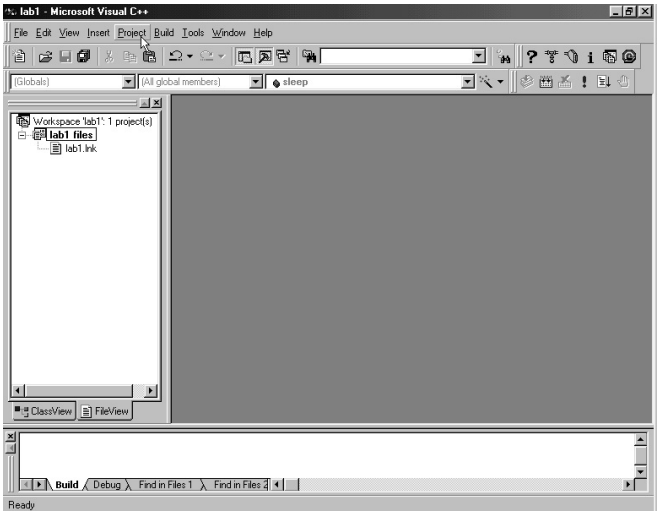


Figure 14 Tree View

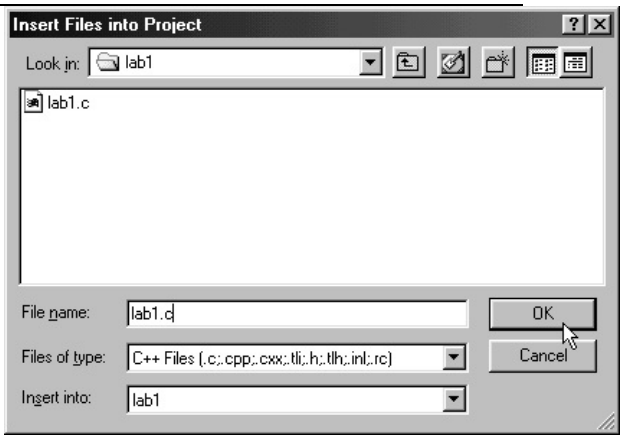


Figure 15 Insert file into project

7. *If not shown*, click on the File View tab to bring in a Tree View of the project. Click on the '+' button to expand the Tree View so you can see the files in the project. The display may differ slightly from what you see.

8. Do ***Project*** > ***Add to Project*** > ***Files*** to locate the lab1.c program in the *lab1* subdirectory.

Double click to bring it into the project.

9. In the Tree View, ***double click*** on ***lab1.c*** to bring the program into the edit window.

10. Complete the program by filling in the blanks for the Bin2LED array.

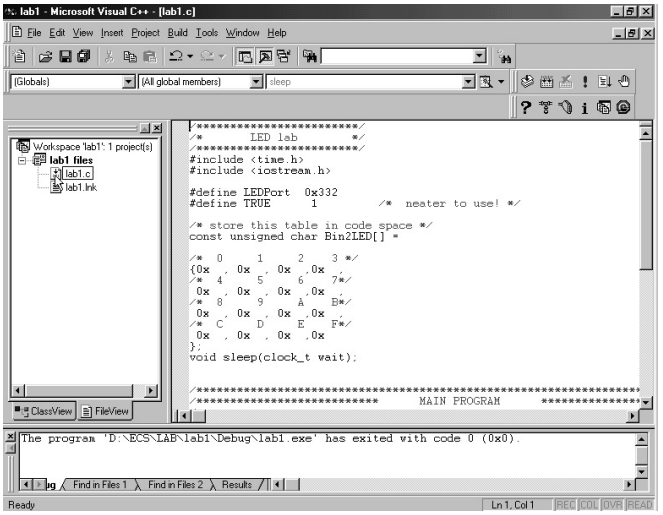


Figure 16 Edit C program

11. Compile the program: ***Build*** > ***Build lab1.exe (F7)*** key. Fix any errors. Warnings are all right.

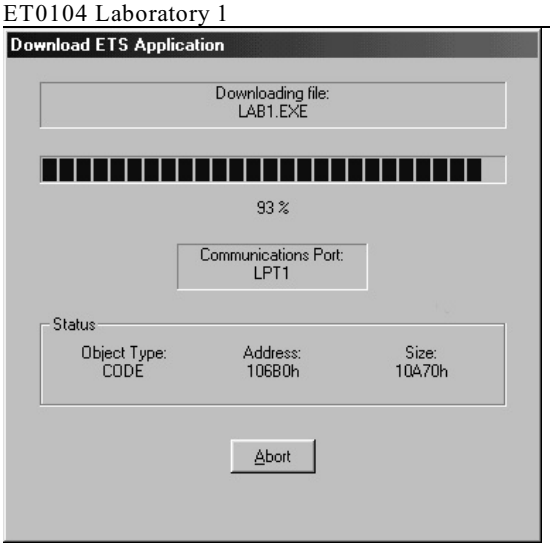


Figure 17 Download screen

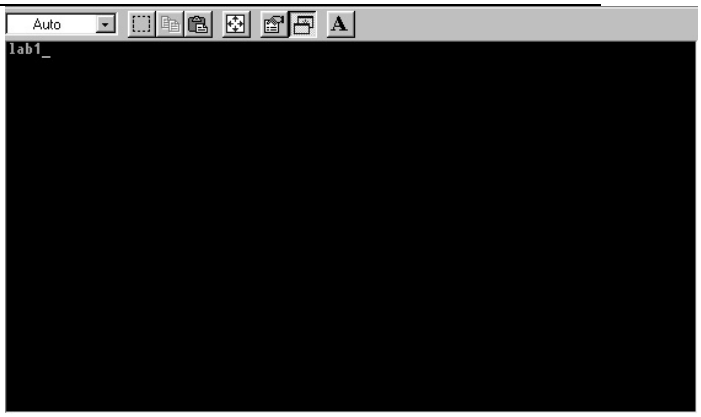


Figure 18 SBC program outputs results to host

12. Execute the program: ***Build*** > ***Execute (Ctl-F5)***. ETS will download the program to the SBC. You should see a downloading screen:

It is IMPORTANT you see this screen!

If this does not show, or you get some error message about there being no communications between the SBC and VC, you will have to RESET the SBC.

13. Finally, you should see a DOS box. Note the "lab1" message from the *printf* statement at the beginning of the C program. Remember, this is from the SBC, transferring data back through the parallel port.

14. Use your watch to check for the accuracy of the timer. Make sure all the digits of the 7 segment LED display correctly.

15. To **end**, you may type Ctl-C, Ctl-Break or use the X icon to close the DOS box.

In case of errors

If there are problems, you may modify your program or change the DIP switch settings.

If you need to change your DIP switch setting, it is highly recommended that you power off the SBC. Make your changes then power it on again.

Appendix:
Program template for lab 1.

```
/************************/
/*       LED lab        */
/************************/
#include <windows.h>

#define LEDPort  0x332
#define TRUE      1                          /*  neater to use! */

/* store this table in code space */
```

```
const unsigned char Bin2LED[] =

/*  0     1     2    3 */
{0x__, 0x__, 0x__,0x__,
/*  4     5     6    7*/
 0x__, 0x__, 0x__,0x__,
/*  8     9     A    B*/
 0x__, 0x__, 0x__,0x__,
/*  C     D     E    F*/
 0x__, 0x__, 0x__,0x__
};

/*****************************************************************************/
/**************************      MAIN PROGRAM      ***************************/
/*****************************************************************************/
void main (void)
{/* main entry for program */
      unsigned char i, LEDval;
      i = 0;
      printf("lab1");
      while (TRUE)
      {
            Sleep(______);           /* non blocking delay 1 second */
            LEDval = Bin2LED[i];
            _outp(LEDPort, LEDval); /* output to LED */
            i++;
            if (i == 10)
                  i = 0;             /* only 1 to 10 */
    }   /* while */
}   /* main */
```

Some specifications of the Advantech PCM4823 “Biscuit PC” series of Single Board Computers

- Supports up to two IDE hard disks with BIOS auto-detect and PIO Mode 3 transfer protocol.
- One serial RS-232 port.
- One serial RS-232/485 port and two 16550 serial UARTs.
- One parallel port which supports SPP/EPP/ECP mode.
- Infrared port shared with COM2, transfer rate up to 115 kbps.
- Mini-DIN connector supports a standard PC/AT keyboard and PS/2 mouse.
- Supports power saving modes including Normal/Standby/Suspend modes.
- APM 1.1 compliant.
- Supports M-Systems’ DiskOnChip® 2000 Flash disk up to 1.44 MB.
- Watchdog timer with 1.6 second interval.