8051 Development Board

Version 1.0
(Manual Revised 8th July 2005)
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Why An 8051?</td>
<td>3</td>
</tr>
<tr>
<td>Why An 8051 Development Board?</td>
<td>3</td>
</tr>
<tr>
<td>Getting Started</td>
<td>6</td>
</tr>
<tr>
<td>What You Should Have</td>
<td>6</td>
</tr>
<tr>
<td>Powering The Board</td>
<td>6</td>
</tr>
<tr>
<td>Basic Board Layout</td>
<td>7</td>
</tr>
<tr>
<td>Loading The Software</td>
<td>8</td>
</tr>
<tr>
<td>Installing The Compiler</td>
<td>8</td>
</tr>
<tr>
<td>Installing Example Code</td>
<td>9</td>
</tr>
<tr>
<td>Compiling Your First Program</td>
<td>9</td>
</tr>
<tr>
<td>Programming The Microcontroller</td>
<td>12</td>
</tr>
<tr>
<td>Installing The Programming Software</td>
<td>12</td>
</tr>
<tr>
<td>Using FLIP (Pre-check)</td>
<td>13</td>
</tr>
<tr>
<td>Using FLIP (Programming)</td>
<td>14</td>
</tr>
<tr>
<td>Program Listings</td>
<td>16</td>
</tr>
<tr>
<td>Demo1 Listing</td>
<td>16</td>
</tr>
<tr>
<td>Demo2 Listing</td>
<td>17</td>
</tr>
<tr>
<td>Demo3 Listing</td>
<td>18</td>
</tr>
<tr>
<td>Demo4 Listing</td>
<td>19</td>
</tr>
<tr>
<td>Circuit Schematics</td>
<td>20</td>
</tr>
<tr>
<td>Spares Parts</td>
<td>22</td>
</tr>
</tbody>
</table>
Introduction

Why An 8051?

The first question I am normally asked is, “Why are you using a chip that’s over 20 years old?” Luckily there are two good answers!

Firstly the 8051 uses a classic architecture that has proved to be easy to understand, easy to interface to logic and other systems like memory and mapped IO and finally its very versatile. This has resulted in the 8051 family being adopted by many hardware designers so there are a great number of systems that use this family and its many derivatives.

The second answer is that although the architecture remains essentially the same as the original ’1980’ Intel microcontroller core, there have been many enhancements to the design including increased on-board RAM, programmable on-board Flash EPROM (no more nasty UV EPROMs to erase!!) and ISP (in system programming) capabilities. Of course there are many more enhancements such as embedded logic like MP3 decoders and Ethernet interfaces - so you simply select the device that has the features you want.

Of course the 8051 family is one of may that exist - PIC, AVR, ST62 etc. etc. Basically what drives designers to use a particular family is either familiarity - you know a particular type of device so you stick to it, or cost - its so cheap you don't mind learning something new! Of course it has to have the capability and speed to satisfy the application but that's normally not a problem as there are so many variants to choose from it comes down to careful selection again.

Why An 8051 Development Board?

We have been designing for the 8051 for many years and realised early on that most design ideas require a prototype with the same basic features each time:-

Power Supply - A clean 3.3V/5V supply is required to run the microcontroller before you can even begin to experiment - so you need this as a basic requirement. Most micros will work from 5V but there are some ‘extended’ range variants that will work at anything from 2V upwards.

It can be very useful if you are interfacing to circuits that don’t operate at 5V, to be able to run the 8051 at this lower voltage and dispense with any unnecessary level shifters.

Microcontroller - You will obviously need a microcontroller that is suited to the application you have in mind. Basically you should decide what major functions and features you need and select accordingly. Price always plays a part but unless you’re making thousands of a particular unit, a few pence extra on a device that can be reprogrammed in-circuit using 5V (as opposed to 12V where a separate supply would be required), may well turn out to be worth the extra.

The micro included on the board has ISP (in system programmability) at 5V with 64Kb of FLASH memory for your programs. This makes it very much easier to develop as the IC doesn’t need to be removed and there is no waiting around for an EPROM to erase etc.
Reset Circuitry - To ensure the micro starts correctly you will need to supply a reset condition at power up. This can be done with a few basic components or you can use a specialised supervisory IC to monitor the system and provide reset signals when required.

The second option is useful for providing protection against brown-outs, to give watchdog capability (if the chosen device doesn't include a watchdog) and a de-bounced manual reset input.

For simplicity and because its adequate for all devices in the family, we've chosen the first option - a few basic components that will give a good reset condition at power-up.

Programming Circuitry - When first powered some devices will automatically go into program mode and wait for instructions while others require a 'program' condition before they'll respond properly. Which ever device you have, the chances are you won't get the code right first time - and even if you did you'd soon want to embellish your design!

To this end you need a few external components that are used to force the microcontroller back into program mode. On the Development Board you can force the micro into program mode by holding down the 'Program' button then momentarily pressing and releasing the 'Reset' button.

Depending on the exact type of microcontroller you have fitted, the jumper JP1 will ensure that the correct combination of signals are generated to trigger program mode - more about that later...

Serial interface - To get anything into or out of the Development Board you need to be able to connect it to a computer. Unfortunately although the serial interface basically speaks zeros and ones, the serial interface on your PC and the microcontroller represent this in different ways.

The microcontroller uses 0V and 5V levels while the PC uses voltages between -15V and -3V for one state and voltages between +3V to +15V for the other.

If this all sounds very complicated then don't worry - its all taken care of by the level shifter IC at the top-right of the Development Board. This allows the microcontroller to talk with your PC and allows it to be programmed, read, erased as well as many other operations etc.

Test buttons - The addition of a few test buttons allows external stimuli to be introduced i.e. user input. The Development Board has four switches that can be jumped to any of the pins on the micro allowing signals to be introduced.

Debug LEDs - In order to help with debugging it is quite useful to have some LEDs that can be illuminated to signal an event or status. Provision has been made for 8 LEDs which are jumped directly into Port 2 but can be isolated if necessary and jumpered to any other pin the microcontroller.
PCB - The last item but a very important one is the PCB itself. The PCB has been arranged to allow easy access to all the pins on the device both from a connection point of view and also physically for easy testing with a voltmeter or logic probe. A clear legend details the purpose of each pin to save constant referral to a datasheet etc. In summary the Development Board gives you everything you need to start experimenting and designing with the 8051 family.

The board accepts 40-pin devices but adaptors are available to allow connection of devices in different packages - PLCC for example. Of course a change of package does not effect the code you compile so having a 40-pin device during the development stage can actually help by making pins easier to probe etc.
Getting Started

What You Should Have

When you unpack your Development Kit you should find the following items:-

1. Development Board
2. Power Supply
3. Serial Lead
4. Manual (what you're reading now!)
5. CD (a compact disc containing examples, SDCC, programming software etc.)
6. Connecting Wires

In the unlikely event that something is missing check to see if an errata sheet has been placed in the box - it is possible that a part is no longer required or has been replaced at the last moment with an alternative.

Powering The Board

You will most likely be eager to see something happen so place the board on a flat non-conductive surface, unpack the power supply and plug it into the mains.

The power supply has a small red LED to indicate that it has power and is working correctly.

Plug the lead into the power socket of the Development Board and move the power switch to the 'ON' position.

When the board is powered the green LED will illuminate and the microcontroller will begin to run the demonstration program (programmed in before its shipped). The row of red LEDs in the 'LED Block' will now begin to count up in a simple binary fashion (LSB on left and MSB on right).

The program can be restarted by pressing the 'Reset' button or by cycling the power using the power switch.
**Basic Board Layout**

The board is quite easy to navigate as the components are grouped into functional 'blocks' according to their purpose. If you ever need to trace a fault or check that signals are getting through correctly this 'block' arrangement makes it very much easier.

- **Power Supply Block**
  Provides a clean and constant voltage to the microprocessor & other ICs. Can be preset for either 3.3V or 5V.

- **Serial Interface**
  Converts the 0-5V levels of the microcontroller to a suitable level for connection to a PC.

- **Reset Circuit**
  Ensures correct power up.

- **Sockets**
  Easily connect to any pin on device with a wire etc.

- **Crystal Oscillator**
  Controls the speed of the board.

- **Switch Block**
  Allows user input and helps with debugging on prototypes.

- **Programming Block**
  Ensures correct signals required to enter program mode.

- **Prototyping Area**
  Allows extra circuitry to be added to your design so there is no need for tacked on pcbs. Do it all on one board!

- **7-Segment Display**
  Allows numbers to be displayed easily for error codes, debugging etc.

- **Debug LEDs**
  Ideal for indicating status or signal levels. Can be connected directly to port 2 or jumpered directly to any pin.

- **Sockets**
  Easily connect to any pin on device with a wire etc.

If the prototyping area is not required it can be simply unscrewed to reveal a matrix of pad-board. This allows for wire-wrapping or direct soldering of ICs and components etc. for more permanent prototypes. Ideal when you want to keep a prototype indefinitely. If you don't want a prototyping area carefully snap it off - its already pre drilled.
**Loading The Software**

Before you can begin to program your board you need two basic pieces of software. The first is a compiler that will take your instructions and turn them into run-able code. The second is some programming software that will take the raw instructions and put them into your microcontroller.

There are a number of excellent compilers available but most of them cost money. I always consider spending money to be a very bad thing - especially when you don't have to. That's why we use an open-source compiler called SDCC that doesn't cost anything. Its an optimised C compiler that supports 8051, Z80, 8086 and some other families including some PIC devices I believe.

If the fact that its free and supports lots of devices isn't enough it also has the benefit of supporting arithmetic functions beyond the 8-bit level - in fact it will quite happily do 16-bit, 32-bit and floating point. The only drawback is that it doesn't really come with any code or examples to get you going on 8051's. This is a shame because in every other respect its perfect for the job.

**Installing The Compiler**

A copy of the compiler is included on the CD in the SDCC directory for your convenience. We always try to ship the current stable version of the compiler but its worth checking out the associated website in case there have been any important updates etc.

The main address for SDCC is [http://sdcc.sourceforge.net](http://sdcc.sourceforge.net)

Note that builds for various operating systems are included on the CD but we will only outline the installation of the Windows version. Instructions for installing the compiler can be found on the SDCC website. One reason why we don't do an install guide for Linux is that there are so many flavours of Linux its difficult to supply the correct information that will work with 100% of all the variants.

The steps required to perform the install will be numbered to make it easier to follow the procedure:

1. Navigate to the /SDCC directory on the CD and run the “sdcc-2.5.0-setup.exe” program. This will guide you through installing the compiler environment on your machine. Accept all the defaults as this ensures the compiler will be in the correct place for the examples to work.
2. At the end of the install you will be asked if you want to add SDCC to the search path. Ensure that you click “Yes” or the examples will not be able to correctly find where SDCC is located.
3. To check SDCC is installed correctly go to the command line by clicking Start/Run and typing “cmd” (for XP) or “command” (for Win95, 98 etc.), then press return.
4. Now type “sdcc -v” and press return. This will return the version of the compiler and you should see something similar to the screen shot below.
If you get an error message or something wildly different please check your installation is correct. Normally the only reason for getting an error is that you clicked “No” at the end of the install and SDCC cannot be found properly.

If this happens just re-run the install. It will first insist on removing everything but then it puts it all back in the correct place.

If you still have trouble you might want to look on our user forums at www.clayzer.com.

**Installing Example Code**

Also included on the CD is example code that can be used to check the compiler is working correctly and allows you to very quickly get to grips with the “tool flow”.

If you are interested in the example code you can copy it directly to your system and start experimenting.

To copy the code navigate to the “Examples” directory on the CD using Explorer. Now right click and select Copy. You can now navigate to where you’d like the examples stored on your computer and select Paste to complete the process.

To simplify the process you can run the “Install-Examples.bat” file in the Examples directory. This will automate the copy process and place the files in the /Program Files/SDCC/Examples directory.

Please note that if you copy the files manually you will need to remove the read-only attribute from them before you can edit and save changes to the files. If you use the automated batch file (above), this will perform the operation for you.

**Compiling Your First Program**

You are now ready to compile some code. For this we are going to use the command line as it does not require any setting up of environments or editors etc. At some point you might want to integrate the compile process into an editor such as XEmacs or SourceEdit but for now the command line will suffice.

To compile your first program perform the following steps:-

1. Open a command line window by clicking Start/Run and typing either “cmd” (for XP) or “command” (for Win95/98 etc.), then press return.

2. You should now see a command prompt like this...

3. Now type “cd \Program Files\SDCC\Examples” and press return. You should now be moved to the examples directory.
4. Type “dir” then press return to list the contents of the directory. Something similar to the following should appear. Each directory eg. “Demo1” contains the code required to demonstrate a particular point.

```
C:\Program Files\SDCC\Examples>dir
Volume in drive C has no label.
Volume Serial Number is 14A8-8F49

Directory of C:\Program Files\SDCC\Examples
05/07/2005 14:44  <DIR>  .
05/07/2005 14:44  <DIR>  ..
05/07/2005 14:44  <DIR>  Demo1
05/07/2005 14:44  <DIR>  Demo2
05/07/2005 14:42  <DIR>  Demo3
05/07/2005 14:42  <DIR>  Demo4
05/07/2005 14:41  355 Install-Examples.bat
05/07/2005 14:07  10 test.txt
      2 File(s)
      365 bytes
    5,192,908,800 bytes free
C:\Program Files\SDCC\Examples>
```

5. Move into the directory required by typing “cd Demo1” for example, then press return. Type “dir” and press return to list the files we are about to compile.

```
C:\Program Files\SDCC\Examples\Demo1>dir
Volume in drive C has no label.
Volume Serial Number is 14A8-8F49

Directory of C:\Program Files\SDCC\Examples\Demo1
06/07/2005  00:37  <DIR>  .
06/07/2005  00:37  <DIR>  ..
05/07/2005  23:41  5,057 at89c51xd2.h
05/07/2005  23:48  1,207 demo1.c
      2 File(s)
      6,264 bytes
      5,192,867,840 bytes free
C:\Program Files\SDCC\Examples\Demo1>
```

6. You should now see the following.

The demo1.c is the demonstration program and the at89c51xd2.h file is a header file which contains all the declarations for ports and pins on the microcontroller. Declarations allow you to control the microcontroller in a way that’s much easier to follow.

For example instead of saying: 0x90 = 55; which just looks like numbers, you can say: P2 = 55; which is the normal way you would set the value of Port 2 (pins 21 to 28 inclusive) on the microcontroller.

The header file contains all sorts of entries to you basically don't need to learn where the ports or registers are located, you just use them! Some of the declarations are for 8-bit registers like Ports 1 thru 5 while others are bit registers like EA. EA is used to enable or disable all the interrupts in the system so the only values it ever needs to be are 0 or 1 (true or false).
7. Earlier we tested to see if the compiler was accessible by typing: `sdcc -v` (return) so compiling the code should now be trivial. By default the compiler assumes you are using an 8051 based microcontroller and that you will be using its internal memory rather than any external memory ICs.

Type: `sdcc demo1.c` (return), to compile the program. If all is well you should just get a command prompt back without any errors or warnings from the compiler. Now type: `dir` (return) to see if any files were created. You should see that you’ve got quite a few more files!

When you ran the compile command the compiler did two things. Firstly it compiled the code which includes various checks to ensure its syntactically correct, variables are declared etc. Then it went on to link the code which is the step of turning it from the assembler code the compiler made, into code that is run-able on the microcontroller.

The upshot of all this is that you should see a “demo1.ihx” file. This is the Intel Hex file that contains the final program code that you’re going to put in the microcontroller. Check the file size and it should be 562 bytes as shown above.
Programming The Microcontroller

If you've followed the example in the previous section you should have created yourself an Intel Hex file that you can now program into your device.

If you have your own hex file created by another tool that you want to use instead then that's fine - just substitute “demo1.ihx” for the name of your hex file in the following dialogue.

Installing The Programming Software

Before you can program anything you need to be able to “talk” to the microcontroller. This normally involves putting the device into some sort of program mode so its ready to accept instructions, connecting the development board to your PC by means of the serial cable and using specialised programming software that speaks the correct language for your brand of microcontroller.

Note that each manufacturer normally has their own programming software that is tailored exactly to their devices. If you use the Atmel devices then you should use their programming software (which is called FLIP). Using software from some other manufacturer may work - but probably won't - and you'll waste loads of time.

The microcontroller supplied on the development board is the Atmel AT89C51ED2 device so you will need to install the Atmel FLIP programmer. You can do this by performing the following operations:-

1. Click Start/Run from your start bar and navigate to the “\Programmers\Atmel FLIP” directory on the CD
2. Now locate the “SETUP.EXE” file, select it and click OK on the run dialog box.
3. Follow the instructions to install the programming software, accepting all the default options when presented.

With your software installed you should be able to find it on your program list when you click Start on your task bar. Note that sometimes Windows will put it right at the bottom of the list of installed programs and sometimes alphabetically - depending on how it feels!
Using FLIP (Pre-check)

This is where you get to put code in your microcontroller – “finally!” you say!

Normally this is a straightforward process but bare in mind that establishing a connection to the development board can depend on a number of things:-

1. Physical connection - I have lost track of the number of times I've tried to program a board that I wasn't even plugged in. It may have power but if the serial lead is dangling at the back of your PC not much will happen.

2. Correct cable - If you are using the cable that came with your kit then you stand an excellent chance of it being wired correctly! The cable supplied is a straight-through pin for pin extension cable (i.e. No crossover) so the blue jumpers fitted directly below the 9W D-type socket on the development board should be in the vertical orientation. This ensures the RX and TX signals are swapped on the board to give the correct crossover of signals.

3. Correct Programming Software - As already discussed, if you try to program with the wrong manufacturers software you'll probably have trouble - and maybe even lock the microcontroller rendering it unusable.

4. Crystal Frequency - Some programming software will ask you for the crystal frequency. That way it can take educated guesses at what serial speeds to try and communicate with. Some software just tries a load of settings until it works while others send a 'U' character and see if its echoed back correctly - if it is then the communication speed is correct and established.

5. Serial Speed - Talking to your Development Board at an amazingly high speed isn't really that necessary. The code will only be a few tens of kilobytes at the most and reading or writing the whole device only takes seconds. For this reason you should be happy with a serial speed of 19,200 or 9,600. Anything more is a bonus but probably won't save that much time as the on-chip programming algorithms limit how fast the code is actually written... Basically you can try the higher speeds but if you have trouble lower the speed to something sensible. Of course once you know its working you can always edge the speed up until you find the limit!

6. Program Mode - Of course nothing will happen AT ALL, unless your microcontroller is actually in program mode! If your program normally flashes LED or something visual you can easily tell that you're in program mode as none of the usual activity will happen. You will be running the internal bootstrap code so you shouldn't expect to see any of your programs activities going on.

If you can't establish communications with the board go through the procedure for putting it back into program mode. Some particularly stubborn microcontrollers from Philips may require the power is turned off for a few seconds - basically they try to automatically negotiate a suitable serial speed and just plain get it wrong! Don't think they are a bad device though - they are just using one of a number of ways to establish a successful connection.

7. Other Programs - Quite often other programs can hog the serial port. These include modems, organisers like PalmPilots and UPS's. If you have trouble establishing a connection it could be that something is already using the port.
Using FLIP (Programming)

OK that’s enough of the woes of when things don’t work lets just get on with it! With all the pre-checks completed you are ready to perform the following actions to load your hex file into the microcontroller.

1. Load FLIP - Find FLIP on your Start menu under ATMEL\FLIP 2.4.2\FLIP and run the software. You should see the following appear.

2. Press F2 and select the device from the dropdown list. The AT89C51ED2 should be about 7th on the list. Select it and click OK. The software will remember the device you chose so you won’t have to select it each time you open the programmer.

3. Press F3 and you should see a dialog appear detailing how you’d like to establish connection. Some advanced microcontrollers have embedded USB so the programmer supports lots of things we won’t use just yet. The microcontroller supplied only uses serial connection so that’s what we will be configuring. The default speed is usually 115,200 so reduce it to something acceptable like 9,600 – you can always try 115,200 later!

4. Turn on the power to the Development Board so that the green LED is illuminated and put it into program mode. You can do this by holding down the “PROGRAM” button and momentarily pressing “RESET”. Any activity from the red debug LED should cease, indicating that program mode has been entered.

5. Now click “Connect” on the FLIP RS232 dialog box. The board and software will now establish a connection. If you can’t connect try resetting the development board and trying again. Almost all problems can be solved by resetting the board and using the Debug Mode of the FLIP software. Debug mode allows you to see the traffic on the link and can often give an insight into what may be going wrong.
6. Now select the file you want to install by pressing F4 and navigating to the Demo1 directory. This should be in “C:\Program Files\SDCC\Examples\Demo1”. Note that the programming software is expecting the file to end with .HEX (which is a slight annoyance because ours doesn't!). If you can't see the “demo.ihx” file click the “File Type” dropdown and select “All Files(*.*)”. The list should now show all the files in the directory including the one you need. Select it and click the “Open” button. If you are using your own hex file from another project just select that instead.

7. Your hex file will be read in and checked to make sure its the correct format for programming etc. If you load any old file the software will complain that it doesn't understand it - which is fair! Just above the Atmel logo on the programming software you should see the information opposite.

Check the number of bytes is 125 - as this is the expected length of the program. If its a lot more then you might not have the correct file loaded.

8. To summarise you have selected the correct device, put the board into program mode, established a connection from FLIP, loaded the hex file and are now ready to program your code into the microcontroller. On the “Operations Flow” panel you can now click the “Run” button to perform the Erase, Blank Check, Program and Verify operations in order. As each step completes the tick box should go green. When all steps are complete the status section at the bottom left should read “Memory Verify Pass”, indicating that all the operations were successful.

When you're used to the tool flow you will be able to perform the actions in whatever order you fancy by using the icons in the tool bar. For example you might want to check some device you just found to see if its blank or contains code. You'd want to Blank Check it first then maybe do a Read!

9. You can now execute your code by either resetting the board by pressing the “RESET” button on the development board or from the programming software click the big red button that says “Start”.

!!! Congratulations on programming success!!!

The demonstration code in “Demo1” should now be running on your development board. The debug LEDs will flash in sequence as the program counts down from 255 to 0 (then repeats).

It will no doubt take a few programming cycles before you are totally familiar with the tool flow but once you've done it a few times it becomes second nature.

If you wish you can now look at the other demonstration programs and see what they do. They will give you some practice at using the tools and development board as well as illustrating how to perform certain tasks like writing to ports, sending serial data, getting key input etc.
Program Listings

Below are listings of the demonstration programs. It is often very useful to check how something worked on the demo code so you can use it in your own programs!

Remember to check our forums at www.clayzer.com as code examples and suggestions are being added all the time...

Demo1 Listing

```c
#include "at09c51xd2.h"

void main( void )
{
    data char ctr; // Declare a variable for use as a quick 8-bit counter */
    data unsigned lop; // Declare 16-bit counter for a time delay */

    ctr = 0xFF; // Give it an initial value or compiler will warn you */
    while( 1 ) // Setup endless 'while true' loop */
    {
        ctr--; // Decrement counter variable (wraps to 255 if contains zero) */
        P2 = ctr; // Set the Port 2 to our counter value */
        lop = 50000; // Set up a delay of 50 thousand */

        while ( lop > 0 ) // Repeatedly decrement delay unit it finally hits zero */
        {
            lop--;
        }
    }
}
```

The “Demo1” code begins execution by initialising two variables called CTR and LOP for use as a counter and loop respectively.

CTR is declared as “char” which means its will have values between 0 and 255. The word DATA makes sure the compiler uses internal working memory (as we have no external RAM chips!). Internal memory is the default for the compiler so its not totally necessary - but included in case future versions of the compiler choose to use a different default memory model.

LOP is declared as “unsigned” which means its going to be a full 16-bit value (0-65535). CTR is initially set to 255 (0xFF) and decremented until it hits zero. Decrementing when on zero just causes the variable to roll-over to 255 again so there is no need to keep re-initialising it.

The while (lop > 0) section acts as a crude time delay so we can see activity on Port 2 easier. Note that LOP is reset to 50,000 each time - before the loop is entered.
```c
#include "at89c51xd2.h"

void main( void )
{
  data unsigned blink;
  blink = 0;
  while( 1 )        /* Setup endless 'while true' loop */
  {
    if (P1_0 == 0)
    {
      P2_0 = 0;
      P2_1 = 1;
      P2_2 = 1;
      P2_3 = 1;
    }
    if (P1_1 == 0)
    {
      P2_0 = 1;
      P2_1 = 0;
      P2_2 = 1;
      P2_3 = 1;
    }
    if (P1_2 == 0)
    {
      P2_0 = 1;
      P2_1 = 1;
      P2_2 = 0;
      P2_3 = 1;
    }
    if (P1_3 == 0)
    {
      P2_0 = 1;
      P2_1 = 1;
      P2_2 = 1;
      P2_3 = 0;
    }
    blink--;
    if (blink == 0)
    {
      P2_7 = P2_7;
    }
  }
}
```
Demo2 is used to test the switches and requires that each of the four switches is connected to the first four inputs of Port 1 i.e. Pins 1, 2, 3 and 4 of the IC.

When a switch is pressed a logic 0 (0V) is applied to one of the four IC pins and the corresponding debug LED will be illuminated.

The LED will remain lit until another key is pressed in which case a different LED will become lit.

Note that debug LED 8 will flash constantly to show the program is running correctly.

### Demo3 Listing

```c
#include "at89c51xrdz.h"

void main( void )
{
    data unsigned blink = 0;
    code char seg[] = {0xCC,0xFB,0xA4,0xB0,0x99,0x92,0x02,0x00,
                      0x8C,0x98,0x88,0x33,0xC6,0xA1,0x86,0x8E};
    data char num = 0;

    while( 1 )         /* Setup endless 'while true' loop */
    {
        P0 = seg[num];

        blink--;        
        if (blink == 0)
        {
            P2_7 = !P2_7;    
            num++;          
            num = num % 16;  
        }
    }
}
```

Demo3 shows how the 7-segment display can be driven. Note that the Port 0 pins have to be connected to the 7-segment display before anything will happen.

Connect P0_0 to segment A, P0_1 to segment B and so on until all of Port 0 is wired to the display digit.

When the program is run it will output patterns to the display causing it to count from “0” to “F” in hexadecimal.
Demo4 Listing

```c
#include "at89c51xd2.h"

void main( void )
{
    data unsigned blink = 0;
    EA = 0;  /* Disable all interrupts */
    SCON = 0x5E;  /* Configure Serial Control register */
    TH1 = 252;  /* Set reload value for Timer 1 baudrate generation */
    TMOD = 0x21;  /* Set Timer Mode register */
    TR1 = 1;  /* Enable transmits */
    TCLK = 0;  /* Tell IC to use Timer1 for TX baud rate generation */
    RCLK = 0;  /* Same as above but for RX */
    ES = 1;  /* Enable serial interrupts */
    EA = 1;  /* Enable all interrupts */

    while( 1 )  /* Setup endless 'while true' loop */
    {
        blink--;
        if (blink == 0)
            {
                P2_7 = 'U';  /* Provide a visual indication that board is running */
                SBUF = 'U';  /* Set a 'U' character to serial port */
            }
    }
}
```

Demo4 shows how to get very basic output to the serial port. Note that we have used no interrupt driven routines or buffers so only one character can be sent at a time.

Further improvements would be to allow sending of strings, numbers and other data using the printf and sprintf functions in the stdio.h library that comes with the compiler.

Although this is beyond the scope of this section there are details on the SDCC site for implementing serial routines.

To allow you to see the serial data you will need to use a terminal program. You can either use Hyperterminal that comes with Windows or the freeware terminal, not surprisingly called “Terminal.exe”!, supplied on the CD.

Which ever program you use the serial communications are at 9600 baud with 8 data bits, no parity and 1 stop bit (i.e. 9600,8,N,1). You should see 'U' characters appearing at the rate of about 2 a second.

Note that the supplied terminal program is particularly useful when you only have 1 serial port as it can be set to automatically disconnect when you swap to using a different program. On the settings panel just ensure that the “Auto Dis/Connect” checkbox is ticked.
Spares Parts

If you require any spare parts such as processor etc. they can be ordered through our website. We buy the components for the development kits in bulk so can obtain them much cheaper.

Spare PCBs are available if you wish to make your own boards as well as PSUs and cables if you want to easily work on development boards in two different locations - just leave the cable and PSU plugged into each PC!