7

ToolStick University Daughter Card

7.0 Introduction

SiLab has developed a C8051F020 based target board specifically aimed for use in teaching microcontrollers and embedded programming. There are many peripheral resources on the board such as four push buttons, eight toggle switches, four LEDs and a potentiometer to generate a variable analog voltage for ADC input. Many other I/O pins are available on the board such as the voltage comparator inputs and DAC outputs. One can also connect a thermistor for which header pin connections are available. Connections to some digital I/O ports are available through header pin connections. There is an on-board 22.1184 MHz crystal oscillator for the system clock, though one can use the microcontroller's internal oscillator too. All these resources and facilities make the board, called *ToolStick University Daughter Card (DC)*, very versatile for teaching and learning the C8051F020 microcontroller. It immensely facilitates experiments which can be done very easily without requiring laboratory instruments.

This chapter will explain the ToolStick University Daughter Card in details and provide example programs to initialize it and use some of the resources.

Block Diagram of the ToolStick University DC

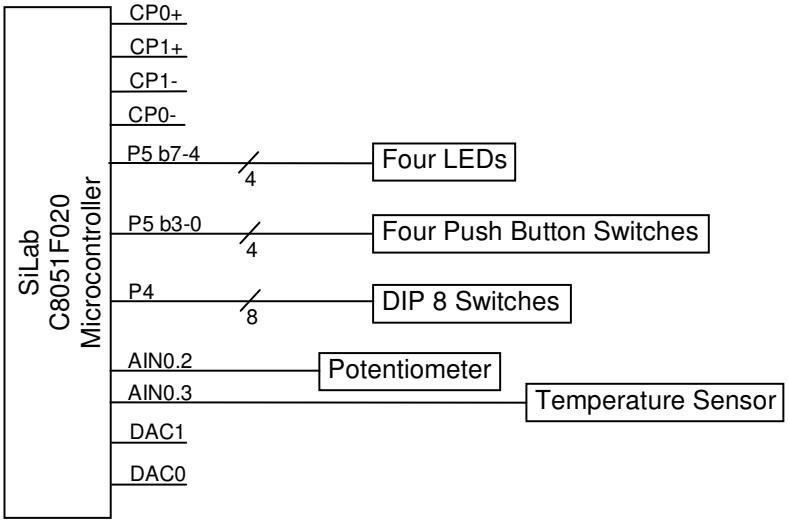

Figure 7.1 University Daughter Card Functional Block diagram

7.1 Peripheral Resources and Connections

The various sub-sections of the ToolStick university daughter card are explained next.

LEDs

In most applications, several LEDs are required, often to depict port status and program diagnostics. Thus four LEDs are provided on the board, connected to the upper half of Port 5 (P5.4 – P5.7). The LEDs are active high and when turned on, approximately 10mA flows through them. Figure 7.2 shows the LED circuitry.

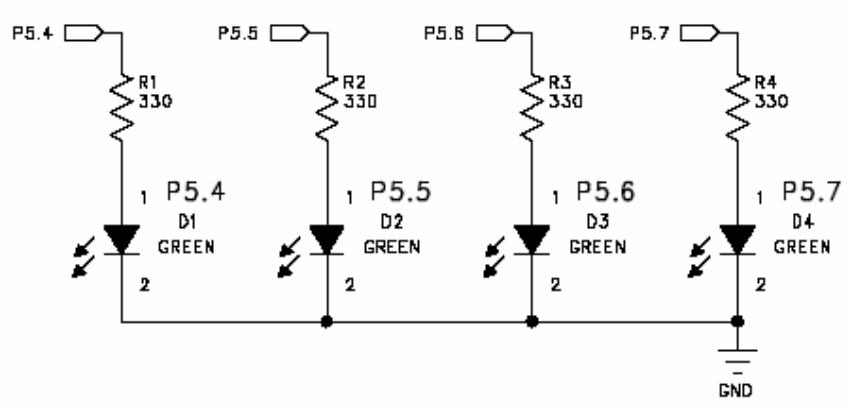

Figure 7.2 LED connections on the University Daughter Card

Pushbutton Switches

Pushbuttons and toggle switches are required in any micro-processor development system for generating digital input signals. Four Pushbuttons are provided on the daughter card; these are connected to the lower half of Port 5 (P5.0 to P5.3). The pushbuttons are active low, i.e. logic ‘0’ is generated at the port pin when the pushbutton is pressed. Figure 7.3 shows the pushbutton circuitry.

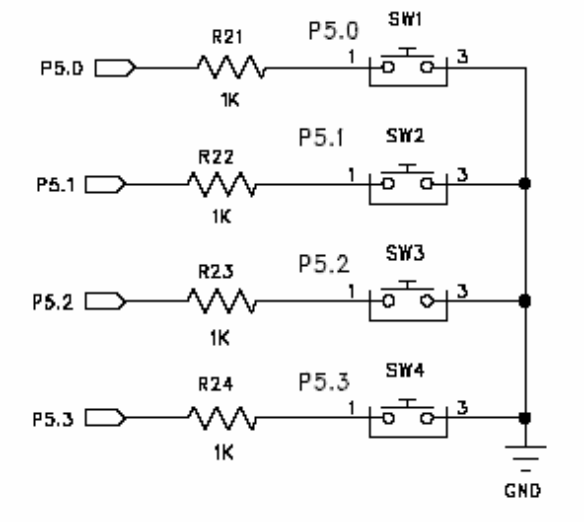

Figure 7.3 Pushbutton connections on the University Daughter Card

Toggle Switches

To further increase the capabilities of the daughter card to provide digital inputs, there are eight toggle switches on it. These are in the form of DIP (Dual-In-Line package) micro-switches connected to Port 4. The toggle switches are active low, i.e. logic '0' is generated at the port pin when the switch is on. Figure 7.4 shows the toggle switch circuitry.

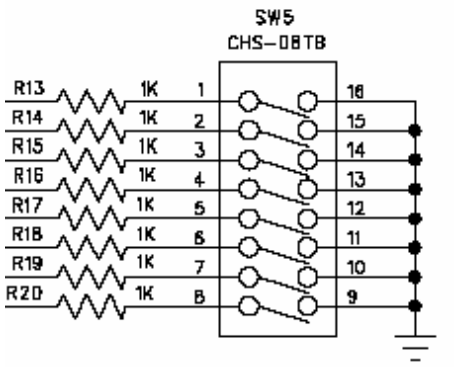

Figure 7.4 Toggle switch connections on the University Daughter Card

Potentiometer

A potentiometer is used as a voltage divider. It allows the ADC to be used with no danger of the input exceeding the maximum rated voltage. The variable analog voltage (0 to 3.3V) from the potentiometer is connected to the input of the 12-bit ADC0 on channel 2 (AIN0.2). Figure 7.5 shows the voltage divider circuitry using the potentiometer.

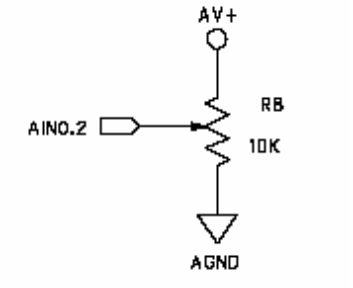

Figure 7.5 Voltage divider circuit using a potentiometer on the University Daughter Card

Temperature Sensor Input

A thermistor can be connected to J4 on the daughter card. The circuitry for the thermistor is shown in Figure 7.6. The analog output of the circuit is fed to the 12-bit ADC0 on channel 3 (AIN0.3).

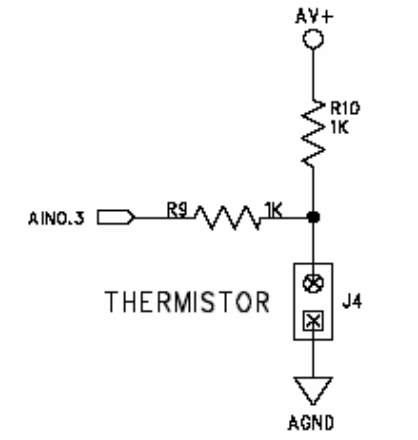

Figure 7.6 Circuit for connecting a thermistor to the University Daughter Card

I/O Port Connections

Pin header connections are provided on the daughter card for external connections to ports 0 to 2. These are on J1 to J3. Figure 7.7 show the connection diagrams for the I/O ports.

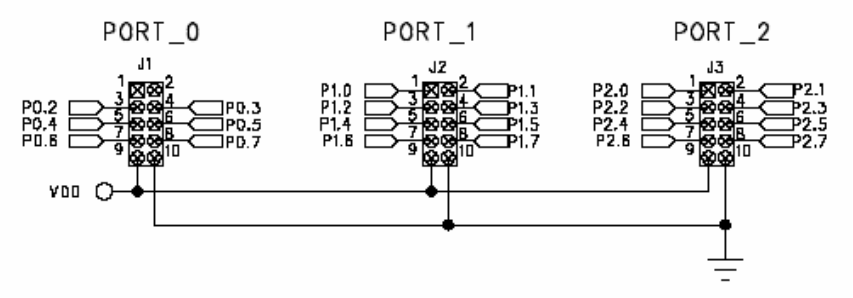

Figure 7.7 Port I/O connections on the University Daughter Card

Analog Signal Connections

Pin header connections are provided on the daughter card for external connections to comparator inputs and DAC outputs. These are on J5. Figure 7.8 show the connection diagrams for the analog inputs and outputs. For various experiments, the DAC outputs or analog voltages at AIN0.2 (potentiometer) and AIN0.3 (thermistor) can be connected to the comparator inputs.

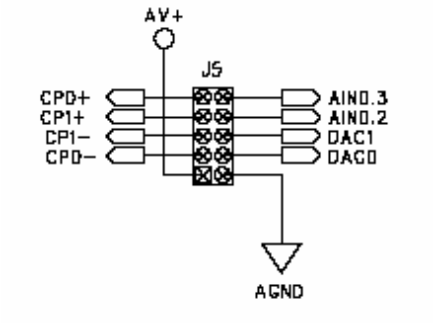

Figure 7.8 Analog I/O connections on the University Daughter Card

7.2 Configuring the University Daughter Card

The following function shows how to configure the microcontroller ports, given all the resources detailed in the previous section.

The entire Port 4 has to be set up for input operation as it is connected to the 8 toggle switches. This is done by configuring its output mode to open drain and writing a logic '1' to all its pins.

The lower four pins of Port 5 also need to be set up for input operation as they are connected to the four push button switches. This is done by configuring the output mode of P5[3:0] to open drain and writing a logic '1' to the associated pins.

The upper four pins of Port 5 need to be configured for output operation as they are connected to the four LEDs. The output mode of P5[7:4] is thus set to push-pull.

In the initialization function shown below, the weak pull-ups have been disabled by programming the WEAKPUD bit (XBR2.7) to 1. The crossbar is enabled by programming the XBARE bit (XBR2.6) to 1.

```
void Init_Port(void)   //-- Configures the Crossbar & GPIO ports
{
        XBR0 = 0x00;
        XBR1 = 0x00;
        XBR2 = 0xC0;    //-- 11000000: Enable Crossbar and
                        //-- disable weak pull-ups

        //-- Port 7-4 I/O Lines
        P74OUT = 0x08; // Output configuration for P4-7
        // (P5[7:4] Push Pull) - 4 LEDs
        // (P5[3:0] Open Drain) - 4 Push-Button Switches (input)
        // (P4 Open Drain) - 8 DIP Switches (input)

        //-- Write a logic 1 to those pins which are to
        //-- be used for input
        P5 |= 0x0F;
        P4 = 0xFF;
}
```

Deleted: //----------
----------¶
void Init_ADC1(void)¶
{¶
.REF0CN = 0x03; .//--
internal bias generat
// internal referen
buffer¶
...// Select ADC1
reference from VREF1
.ADC1CF = 0x81; .//--
conversion clock=941K
// approx., Gain=1¶
.AMX1SL = 0x00; .//--
AIN1.0 input¶
.ADC1CN = 0x82; .//--
ADC1, Continuous Trac
// Mode, Conversion
initiated on Timer ¶
// 3 overflow¶
}¶
//----------

--------¶
¶
¶
//----------

--------¶
// Interrupt Service
¶
void Timer3_ISR (void
interrupt 14¶
{¶
.TMR3CN &= ~(0x80);
clear TF3 flag¶
¶
.//-- wait for ADC1
conversion to be over
.while ((ADC1CN & 0x
0); //-- poll for AD1
.ADC1_reading = ADC1;
read ADC1 data¶
.ADC1CN &= 0xDF; .//--
AD1INT¶
}¶
//----------

Formatted: Font: Bold

Formatted: Indent: Left: 0
Hanging: 1"

Formatted: Indent: Left: 0
Hanging: 0.5"

Formatted: Indent: Left: 0

7.3 Connecting the University DC to a PC

The university daughter card is connected to the personal computer using the USB port through the *ToolStick Base Adapter*. Figure 7.9 shows the connection diagram. The connection between the daughter card and the base adapter uses the Card Edge.

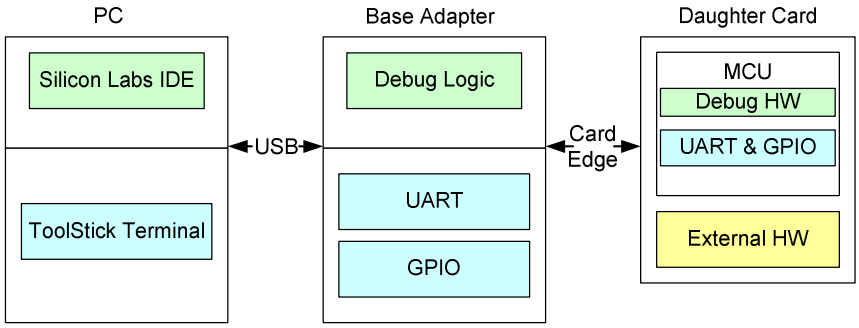

Figure 7.9 Connecting the University Daughter Card to a PC using a ToolStick Base Adapter

The base adaptor provides a USB debug interface to a Windows PC and the firmware for UART serial communication between the PC and the Daughter Card. Figure 7.10 shows the ToolStick University Daughter Card connected to the ToolStick Base Adapter.

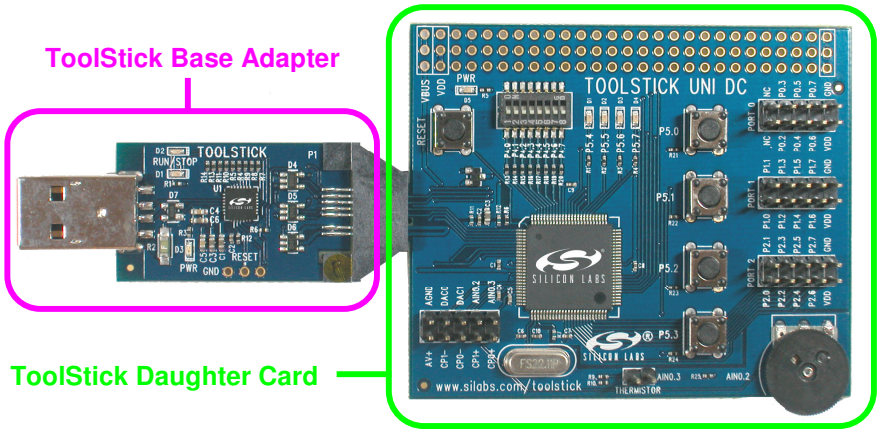

Figure 7.10 University Daughter Card connected to the ToolStick Base Adapter

The Base Adapter may be connected directly to the USB port of the PC, as shown in Figure 7.11 or it can be connected using the USB extension cable as shown in Figure 7.12.

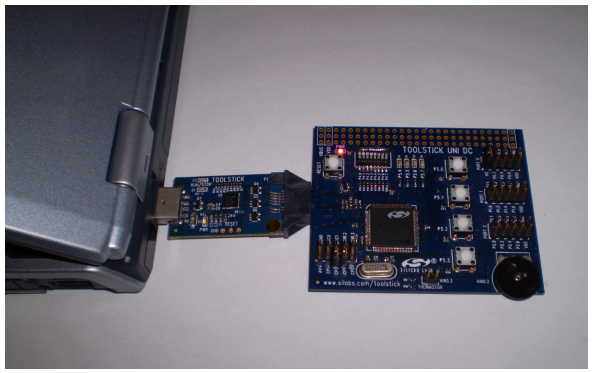

Figure 7.11 Base Adapter connected directly to the USB port of the PC

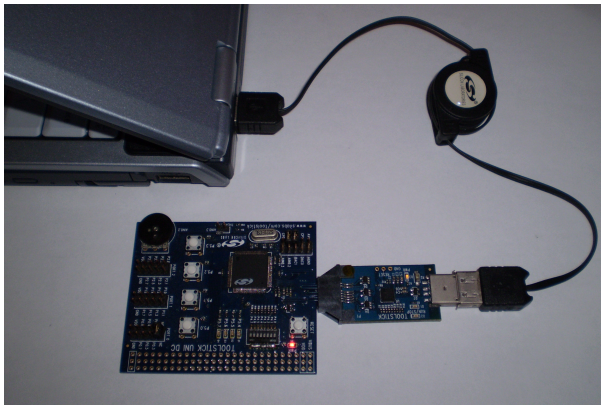

Figure 7.12 Base Adapter connected to PC using the USB extension cable

Debug Adapter Daughter Card

The ToolStick Base Adapter can be connected to the Debug Adapter Daughter Card as shown in Figure 7.13 and Figure 7.14. This provides the 10-pin interface so that the ToolStick Base Adapter can function as a

debug adapter for the standard SiLab microcontroller development boards.

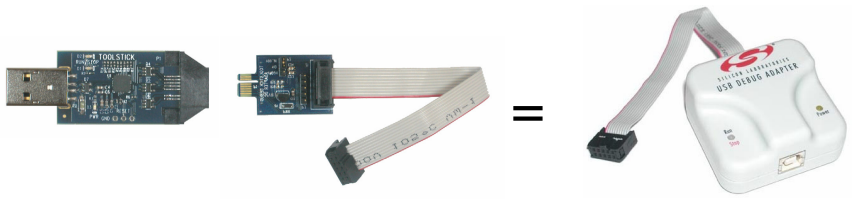

Figure 7.13 Using ToolStick Base Adapter and Debug Adapter Daughter Card

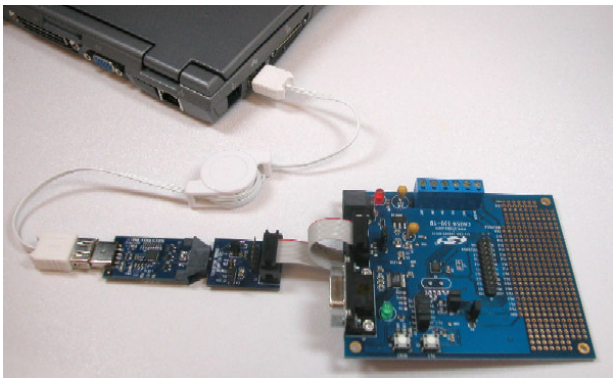

Figure 7.14 Using ToolStick Base Adapter to connect to a standard development board

7.4 Hardware Overview of ToolStick Base Adapter

Figure 7.15 shows the various hardware components of the ToolStick Base Adapter.

Power LED: Indicates that the USB power is on.

Run/Stop LEDs: These LEDs indicate if the target MCU (on the daughter card that is connected to the base adapter) is currently running or halted.

SiLab MCU: This Microcontroller performs the USB debug adapter functions and implements PC communication functions.

Socket Connector: This socket accepts a 14-pin card-edge connector. The MCU daughter card or the Debug Adapter Debug Card may be connected to this socket.

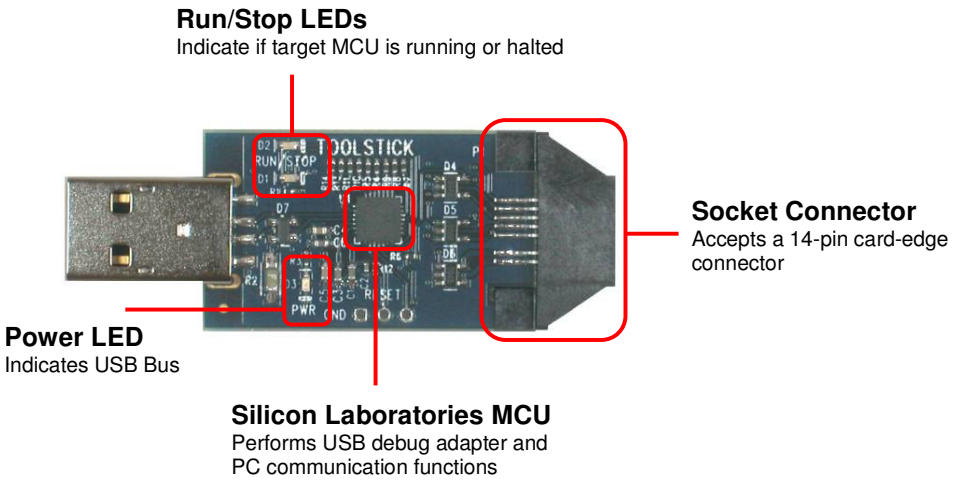

Figure 7.15 Hardware components of the ToolStick Base Adapter

7.5 Hardware Overview of ToolStick University Daughter Card

Figure 7.16 shows the ToolStick University Daughter Card and the following resources and connections.

- LEDs
- Pushbutton Switches
- Toggle Switches
- Potentiometer
- Temperature Sensor Input
- I/O Port Connections
- Analog Connections

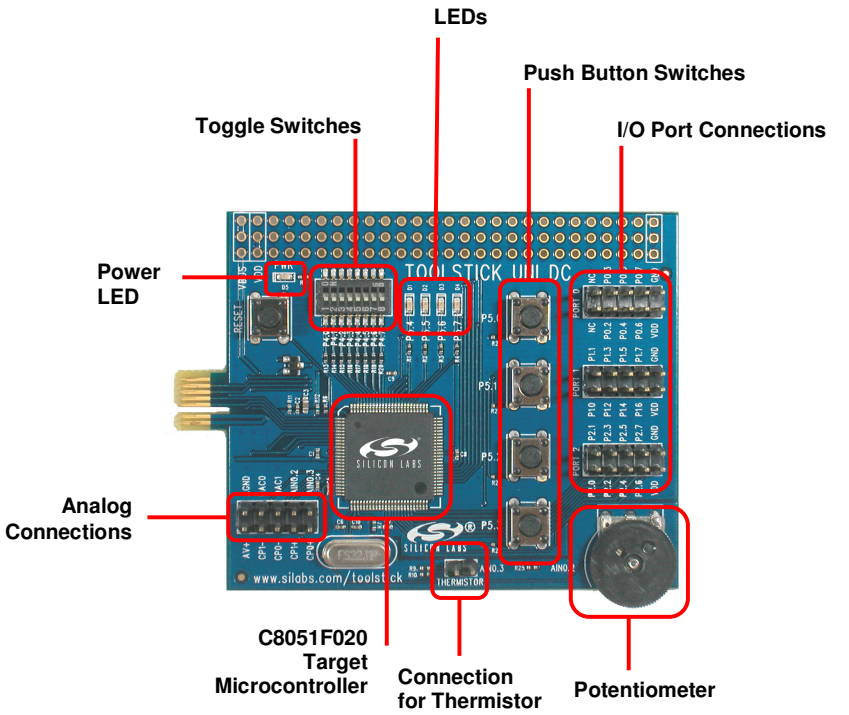

Figure 7.16 Resources and connections on the ToolStick University Daughter Card

7.6 Starting a Project in SiLab IDE

In common with many Windows based software development environments, the SiLab Integrated Development Environment (IDE) uses a project file to specify the actions to be performed on the set of files it is currently working with. With a simple project there will only be a single file involved but larger projects quickly expand to a number of **.c** source files and perhaps assembly files too. The project file stores other information too, including the state of the IDE desktop.

A sensible approach is to start with an empty directory for a new project. Using the menus: Project/New Project followed by Project/Save Project As and navigating to your new directory results in a workspace file (**.wsp**). Similarly, create a new **.c** file and save it as well.

At this point you have a **.wsp** and a **.c** file in your directory, but the **.c** file is not actually part of the project. Project/Add Files to Project will allow you to add the file to the project.

You can now proceed to write your program. It can be compiled with F7, downloaded with Alt-D, and run with F5. To combine the compile and download steps go to Project/Target Build Configuration and check the Enable automatic download/connect after build check box.

7.7 Blinking the LEDs on the Daughter Card Using Software Delays

To come to grips with programming a new microcontroller, it is best to get something working – the simpler the better. A good place to start is blinking a LED. Using software delays, rather than an interrupt, is the simplest approach, although it is generally a poor practice. The program to do this is trivial but there are overheads involved in configuring the SiLab C8051F020 which must be understood.

Because the MCU operates very fast it is necessary to slow it down if a blinking LED is to be observed. The following program makes the 4 green LEDs (on P5.4 to P5.7) blink all together. It uses delay loops to control the blinking on/off period. The internal oscillator at 8 MHz has been used. The same delay functions will implement different amount of delay of the clock speed is changed.

While software delays can be quite accurate, if calibrated, it is difficult to do so and they loose any time which is taken by interrupts. They also tie up the processor while running, so other tasks, such as reading the keyboard or getting data from the serial buffer, may potentially be ignored.

Whereas many C programs run for a time and then exit, a program in a microcontroller normally runs forever. This can be seen in the program where the ***main()*** function has a **while** loop that runs for ever (remember that 1 is Boolean true, 0 is false).

```
//-- This program makes the 4 green LEDs (on P5.4 to P5.7) blink
//-- (all together). Uses delay loops to control the blinking
//-- (on/off) period. Uses internal oscillator at 8 MHz

#include <c8051f020.h>        // SFR declarations

//-----------------------------------------------------------------
// Function PROTOTYPES
//-----------------------------------------------------------------
void init_Clock(void);        // System clock initialisation
void init_Port(void);         // general system initialization
void small_delay(char count);         // small delay loop
void big_delay(char count);           // big delay loop
void huge_delay(char count);          // huge delay loop

void main(void)
{
       //-- disable watchdog timer
       WDTCN = 0xDE;
       WDTCN = 0xAD;

       init_Clock();
       init_Port();

       P5 = P5 & 0xFF;        //-- turn ON the four LEDs

       while (1)      //-- go on forever (endless loop)
       {
              huge_delay(20);
              P5 = P5 ^ 0xF0; //-- toggle the four LEDs using
                              //-- EX-OR with '1'
       }
}

void init_Clock(void)
{
       //-- program the INTERNAL Oscillator Control Register

       OSCICN = 0x86;         //-- 1000 0110b
       //-- Bit 7 : Missing Clock Detector Enabled (MSCLKE = 1)
       //-- Bit 3 : Uses Internal Osc as System Clock CLKSL = 0)
       //-- Bit 2 : Internal Osc. enabled (IOSCEN = 1)
       //-- Bit 1-0 : 8 MHz (IFCN1-IFCN0 set to 10)

       //      OSCICN = 0x84; //-- 2 MHz
       //      OSCICN = 0x85; //-- 4 MHz
       //      OSCICN = 0x87; //-- 16 MHz

       while ( (OSCICN & 0x10) == 0);//-- poll for IFRDY -> 1

}
```

```
void init_Port(void)
{
        //----- Configure the XBRn Registers
        XBR0 = 0x00;
        XBR1 = 0x00;
        XBR2 = 0x40;// Enable the crossbar, weak pull-ups enabled

        //-- Port configuration (0 = Open Drain, 1 = Push Pull)
        P0MDOUT = 0x00;         // Output configuration for P0
        P1MDOUT = 0x00;         // Output configuration for P1
        P2MDOUT = 0x00;         // Output configuration for P2
        P3MDOUT = 0x00;         // Output configuration for P3

        //-- Port 7-4 I/O Lines
        P74OUT = 0x08; // Output configuration for P4-7
                       // (P5[7:4] Push Pull) - 4 LEDs
        // (P5[3:0] Open Drain) - 4 Push-Button Switches (input)
        // (P4 Open Drain) - 8 DIP Switches (input)

        //-- Write a logic 1 to those pins which are to be used
        //-- for input
        P5 |= 0x0F;
        P4 = 0xFF;
}

//=========================================
//         delay loops
//=========================================
void small_delay(char count)
{
        while(count--);
}

void big_delay(char count)
{
        while(count--) small_delay(255);
}

void huge_delay(char count)
{
        while(count--) big_delay(255);
}
```

```
//-----------------------------------------------------------------
void Init_ADC1(void)
{
       REF0CN = 0x03; //-- Enable internal bias generator and
//   internal reference buffer
                      //   Select ADC1 reference from VREF1 pin
       ADC1CF = 0x81; //-- SAR1 conversion clock=941KHz
//   approx., Gain=1
       AMX1SL = 0x00; //-- Select AIN1.0 input
       ADC1CN = 0x82; //-- enable ADC1, Continuous Tracking
//   Mode, Conversion initiated on Timer
//   3 overflow
}
//-----------------------------------------------------------------


//-----------------------------------------------------------------
// Interrupt Service Routine

void Timer3_ISR (void) interrupt 14
{
       TMR3CN &= ~(0x80);     //-- clear TF3 flag

       //-- wait for ADC1 conversion to be over
       while ( (ADC1CN & 0x20) == 0); //-- poll for AD1INT-->1
       ADC1_reading = ADC1;   //-- read ADC1 data
       ADC1CN &= 0xDF;        //-- clear AD1INT
}
//-----------------------------------------------------------------
```