Chapter 6

Analogue to Digital Conversion

Up to now we have considered inputs as being digital in operation, i.e., the input is either a 0 or 1. But suppose we wish to make temperature measurements, but not just hot or cold (1 or 0). We may for example require to:

- Sound a buzzer if the temperature drops below freezing.
- Turn a heater on if the temperature is below 18°C.
- Turn on a fan if the temperature goes above 25°C.
- Turn on an alarm if the temperature goes above 30°C.

We could of course have separate digital inputs, coming from comparator circuits for each setting. But a better solution is to use 1 input connected to an analogue to digital converter and measure the temperature with that.

Before we consider the application we need to look at how we configure the A/D convertor.

CONFIGURING THE A/D DEVICE

In order to make an analogue measurement we have to configure the device.

To configure the 18F1220 for A–D measurements five registers are used.

- ADCON0
- ADCON1
- ADCON2
- ADRESH
- ADRESL

ADCON0 A–D CONTROL REGISTER 0

ADCON0 is used to:

- Switch the A/D converter on with ADON, bit0. This bit turns the A/D on when set and off when clear. The A/D once it is turned on can be left on all of the time but it does draw a current of 90 μA, compared to the rest of the microcontroller which draws a current of 15 μA.

R/W-0	R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
VCFG1	VCFG0	—	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	ADON
bit 7							bit 0

bit 7-6 **VCFG<1:0>:** Voltage Reference Configuration bits

	A/D V_{REF+}	**A/D V_{REF-}**
00	AV_{DD}	AV_{SS}
01	External V_{REF+}	AV_{SS}
10	AV_{DD}	External V_{REF-}
11	External V_{REF+}	External V_{REF-}

bit 5 **Unimplemented:** Read as '0'

bit 4-2 **CHS2:CHS0:** Analog Channel Select bits
000 = Channel 0 (AN0)
001 = Channel 1 (AN1)
010 = Channel 2 (AN2)
011 = Channel 3 (AN3)
100 = Channel 4 (AN4)
101 = Channel 5 (AN5)
110 = Channel 6 (AN6)
111 = Unimplemented(1)

bit 1 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit
When ADON = 1:
1 = A/D conversion in progress
0 = A/D Idle

bit 0 **ADON:** A/D On bit
1 = A/D converter module is enabled
0 = A/D converter module is disabled

FIGURE 6.1 ADCON0 register.

- Instruct the microcontroller to execute a conversion by setting the GO/DONE bit, bit1. When the GO/DONE bit is set the micro does an A/D conversion. When the conversion is complete the hardware clears the GO/DONE bit. This bit can be read to determine when the result is ready.
- Set the particular channel (input) to make the measurement from. This is done with three Channel Select bits, CHS0, CHS1, and CHS2, bits 2, 3, and 4.
- Determine the +ve and −ve voltage references for the A/D convertor using bits 6 and 7. 00 here will select the supply rails as the references.
- The Register ADCON0 is shown in Figure 6.1.

ADCON1 A–D CONTROL REGISTER 1

The 18F1220 has 7 I/O that can be configured as analogue inputs. They are found on pins 1, 2, 6, 7, 8, 9, and 10 which also correspond to RA0, RA1, RA2, RA3, RB0, RB1, and RB4. Shown in Figure 2.1. The outputs can be switched as analogue or digital individually by the program configuration bits PCFG in ADCON1.

The ADCON1 register is shown in Figure 6.2.

The diagram shows the 8 bit register, bit 7 is unused. The bits can be written to or read and power up as 0, all analogue.

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
——	PCFG6	PCFG5	PCFG4	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

FIGURE 6.2 ADCON1.

R/W-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	–	ACQT2	ACQT1	ACQT0	ADCS2	ADCS1	ADCS0
bit 7							bit 0

bit 7 ADFM: A/D Result Format Select bit
1 = Right justified
0 = Left justified

bit 6 Unimplemented: Read as "0"

bit 5-3 **ACQT2: ACQT0:** A/D Acquisition Time Select bits
000 = 0 TAD(1)
001 = 2 TAD
010 = 4 TAD
011 = 6 TAD
100 = 8 TAD
101 = 12 TAD
110 = 16 TAD
111 = 20 TAD

bit 2-0 **ADCS2: ADCS0:** A/D Conversion Clock Select bits
000 = FOSC/2
001 = FOSC/8
010 = FOSC/32
011 = FRC (clock derived from A/D RC oscillator)(1)
100 = FOSC/4
101 = FOSC/16
110 = FOSC/64
111 = FRC (clock derived from A/D RC oscillator)(1)

FIGURE 6.3 ADCON2.

A logic 1 sets the I/O pin as digital a logic 0 sets the pin as analogue. In our header we have used:

ADCON1=0x7F; //all IO are digital or 0b01111111 in binary.

ADCON2 A–D CONTROL REGISTER 2

ADCON2 controls the Conversion Clock Speed, the Acquisition Time, and the Result Format and is shown in Figure 6.3. The programs used in this book use all 10 bits of the result rather than 8 and right justification is used. So bit 7, ADFM in ADCON2 requires setting, i.e., to a 1. Because the A/D section runs off an internal clock of 31.25 kHz and we are using right justification, then ADCON2 is set to 0b10000000.

When making an A/D reading there are two timing components that need to be considered to ensure enough time is allowed to make the measurement.

AD clock source TAD		
Operation	ADCS2:ADCS0	Clock Frequency
2TOSC	000	1.25 MHz
4TOSC	100	2.50 MHz
8TOSC	001	5.00 MHz
16TOSC	101	10.0 MHz
32TOSC	010	20.0 MHz
64TOSC	110	40.0 MHz

FIGURE 6.4 TAD versus clock frequencies.

NB. For clock frequencies below 100 kHz the clock is slow enough, so we do not need to slow it down and can ignore these two timing requirements.

1. The T_{AD} time which is the minimum time required to convert 1 bit which is 1.6 µs. So if the clock frequency is 10 MHz, i.e., A period, T_{OSC}, of 0.1 µs we require 16 of these clock periods. So T_{AD} has to be $16T_{OSC}$, i.e., $16 \times 0.1\,\mu s = 1.6\,\mu s$.
   Figure 6.4 shows the relationship between the clock frequencies and the T_{AD} and the selection using the A/D clock source select bits ADCS2, ADCS1, and ADCS0 in ADCON2. In this case 101.
2. The Acquisition Time, T_{ACQ}, is the time required for the A/D convertor to be ready to take a measurement after the channel has been selected. It is given as about 12.86 µs. This is $8 \times T_{AD}$ (8v× 1.6 µs = 12.8 µs. The T_{ACQ} is set by bits ACQT2, ACQT1, and ACQT0 in ADCON2—so use setting 100.

With a conversion time for 12 bits × 1.6 µs = 19.2 µs and an acquisition time of 12.86 µs that gives a total time of 32.06 µs which means that the maximum theoretical frequency of A/D readings is about 30 kHz (31.25 kHz).

ADRESH AND ADRESL: A/D RESULT REGISTERS HIGH AND LOW BYTE

The fourth and fifth registers are the A to D RESult registers which are the files where the result of the A/D conversion is stored. The A/D result is stored in 10 bits which requires 2, 8 bit registers. If several measurements require storing then the number in ADRES needs to be transferred to a user file before it is overwritten with the next measurement.

The A/D result can be left justified or right justified as shown in Figures 6.5 and 6.6. The justification is set by bit 7, ADFM in ADCON2. In this book the condition right justified is used.

With right justification the 10 bit RESULT = ADRESL + (ADRESH × 256).

The low byte ADRESL counts up to 256 and overflows. The high byte ADRESH is incremented every time ADRESL overflows, i.e., ADRESH is

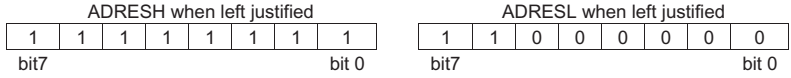


FIGURE 6.5 ADRES left justified.

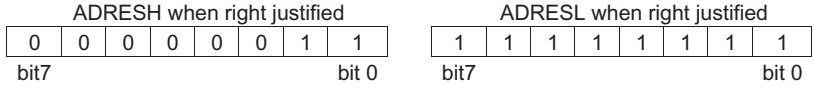
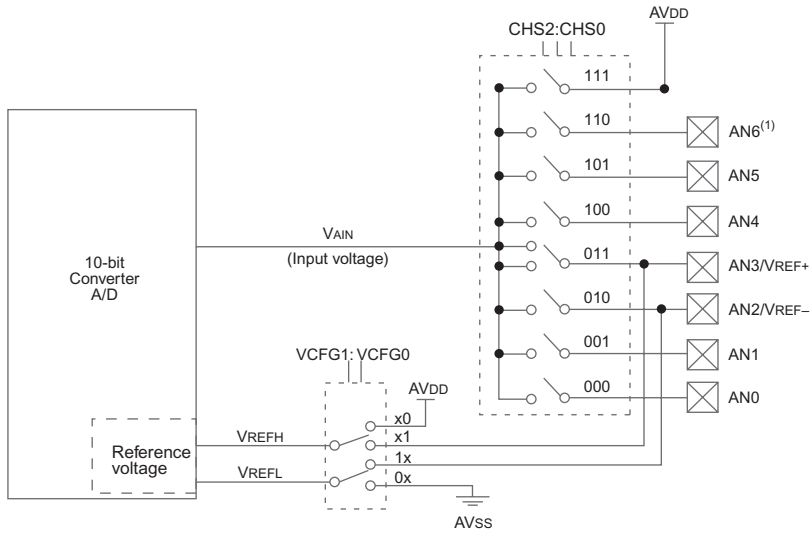


FIGURE 6.6 ADRES right justified.



Note 1: I/O pins have diode protection to VDD and VSS.

FIGURE 6.7 The block diagram of the A/D convertor.

counting how many 256s there have been. (This is similar in operation to an hours and minutes file. The minutes file overflows at 60 and the hours file counts the number of these 60 minutes.)

The block diagram of the A/D convertor is shown in Figure 6.7.

Now that we have seen how to configure the device let us look at that temperature measurement.

THE THERMISTOR

Figure 6.8 shows a basic circuit for measuring temperature. It consists of a fixed resistor in series with a thermistor (a temperature sensitive resistor).

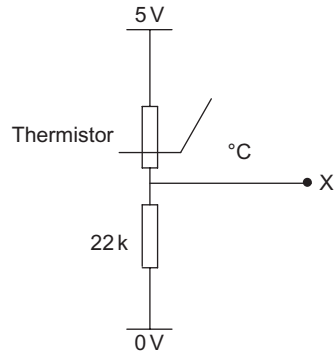


FIGURE 6.8 Thermistor temperature measuring circuit.

The resistance of the thermistor changes with temperature causing a change in the voltage at point X shown in Figure 6.8.

As the temperature rises the voltage at X rises.

As the temperature decreases the voltage at X reduces.

We need to know the relationship between the temperature of the thermistor and the voltage at X. A simple way of doing this would be to place the thermistor in a cup of boiling water (100°C) and measure the voltage at X. As the water cools corresponding readings of temperature and voltage can be taken. If needed a graph of these temperature and voltage readings could be plotted.

MAKING AN A/D READING WITH THE THERMISTOR

In the initial example let us suppose:

- 0°C gave a voltage reading of 0.6 V
- 18°C gave a reading of 1.4 V
- 25°C gave a reading of 2.4 V
- 30°C gave a reading of 3.6 V

The microcontroller would read these voltages and convert them to a 10-bit number where 0 V is 0 and 5 V is 1023, i.e., a reading of 204.6 per volt or a resolution of 1/204.6 V, i.e., 1 bit is 4.89 mV.

So 0°C = 0.6 V = reading of 122 (0.6 × 204.6 = 122.76)

18°C = 1.4 V = 286 (1.4 × 204.6)

25°C = 2.4 V = 491 (2.4 × 204.6)

30°C = 3.6 V = 736 (3.6 × 204.6)

If we want to know when the temperature is above 30°C the microcontroller looks to see if the A/D reading is above 736. If it is, switch on the alarm, if not keep the alarm off. In a similar way any other temperature can be investigated—not just the ones listed. With our 10 bits we have 1023 different temperatures we can choose from.

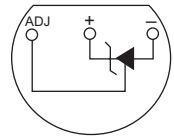


FIGURE 6.9 The temperature sensing IC.

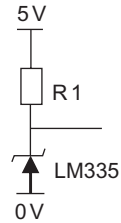


FIGURE 6.10 Temperature measuring circuit.

THE LM335 TEMPERATURE SENSING IC

Rather than use the thermistor a better solution would be to use the temperature sensing IC the LM335 which has a temperature range from −40°C to +100°C. The LM135 has a range from −55°C to +150°C and the LM235 ranges from −40°C to +125°C.

The pin connection for the IC is shown in Figure 6.9.

OPERATION OF THE LM335

The LM335 gives an output voltage of 10 mV/K, where 0K is −273°C.

So 0°C is 273K equivalent to an output voltage of $273 \times 10\,\text{mV} = 2.73\,\text{V}$.

−40°C is 233K (40 below zero of 273) equivalent to an output voltage of $233 \times 10\,\text{mV} = 2.33\,\text{V}$.

100°C is 373K (100 above zero of 273) equivalent to an output voltage of $373 \times 10\,\text{mV} = 3.73\,\text{V}$.

The circuit diagram of the IC is shown in Figure 6.10.

The IC requires a bias current between 0.4 and 5 mA. So using a supply of 5 V and at 0°C the voltage across the IC is 2.73 V so the voltage across R1 is 2.27 V (5 − 2.73). If we require a current of say 1 mA then the value of R1 would be $2.27\,\text{V}/1\,\text{mA} = 2.27\,\text{k}\Omega$, i.e., a value of 2.2 k would be required. NB. the value of the bias current does not affect the voltage reading across the device.

A/D APPLICATION

Let us now consider the application to:

Sound a buzzer if the temperature drops below freezing.
Turn a heater on if the temperature is below 18°C.

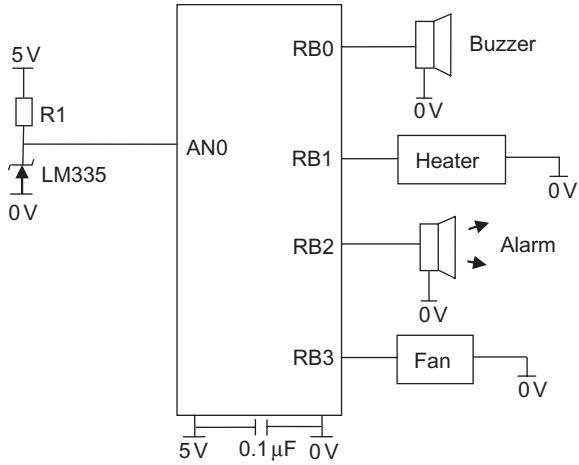


FIGURE 6.11 The temperature control circuit.

Turn on a fan if the temperature goes above 25°C.
Turn on an alarm if the temperature goes above 30°C.
The first thing we need to do before we start programming is to sort out where our inputs and outputs are going. So draw the circuit diagram as shown in Figure 6.11.

For the LM335:

0°C is 2.73 V an A/D reading of $204.6 \times 2.73 = 559$
18°C is $2.73 + (18 \times 10\,\text{mV}) = 2.91\,\text{V}$ an A/D reading of $204.6 \times 2.91 = 595$
25°C is $2.73 + (25 \times 10\,\text{mV}) = 2.98\,\text{V}$ an A/D reading of $204.6 \times 2.98 = 610$
30°C is $2.73 + (30 \times 10\,\text{mV}) = 3.03\,\text{V}$ an A/D reading of $204.6 \times 3.03 = 620$

So READING / 204.6=voltage
Voltage = 2.73 + (TEMPERATURE x 0.01)

TEMPERATURE = (voltage – 2.73) / 0.01
= (READING / 204.6 – 2.73)/0.01
= READING / 2.046 – 273

Open MPLAB and OPEN the file Header.c Figure 3.5 to save some initial typing then modify the code to look like the Temperature.C file below.

```
1   //Temperature.C, DW Smith, 21st November 2011
2
3   #include <p18f1220.h>
4   #pragma config WDT=OFF, OSC=INTIO2, PWRT=ON, LVP=OFF, MCLRE=OFF
5   #include <delays.h>
6
```

```
7       int READING;
8       float TEMPERATURE;          //Temperature is a decimal number not an integer.
9
10      void main (void)
11      {
12      //SET UP
13      // OSCCON defaults to 31 kHz. So no need to alter it.
14      ADCON1=0x7E;                //AN0 is analogue or 0b01111110 in binary
15      TRISA=0b11111111;           //sets PORTA as all inputs, bit0 is AN0
16      PORTA=0b00000000;       //turns off PORTA outputs, not required, no outputs
17      TRISB=0b00000000;           //sets PORTB as all outputs
18      PORTB=0b00000000;           //turns off PORTB outputs, good start position
19      ADCON0bits.ADON=1;          //turn on A/D
20      ADCON2=0b10000000;    //right justified, acquisition times are ok at 0 with 31 kHz
21
22      while (1)
23        {
24      ADCON0bits.GO_DONE=1;                   // do A/D measurement
25      while (ADCON0bits.GO_DONE== 1);         //wait until bit=0 for measurement
                                                completed
26      READING=ADRESL+(ADRESH * 256);
27      TEMPERATURE=READING / 2.046–273.0;
28
29      if (TEMPERATURE <= 0)    PORTB=0b00000011;   //buzzer and heater on
30      if (TEMPERATURE>0 && TEMPERATURE <=18)    PORTB=0b00000010;
                                                                        //heater on
31      if (TEMPERATURE>18 && TEMPERATURE <= 25)    PORTB=0b00000000;
                                                                    //no output required
32      if (TEMPERATURE>25 && TEMPERATURE <= 30)    PORTB=0b00001000;
                                                                        //fan on
33      if (TEMPERATURE>30)                 PORTB=0b00001100;          //fan and alarm on
34        }
35      }
```

Explanation of the Code

The previous header has been loaded and modified to provide the code for the Temperature.C program.

- Lines 7 and 8 have declared the variables we are using in our code. READING is the value of the A/D reading and is an integer. The TEMPERATURE is computed from the READING and is a decimal (floating point) number (not an integer).
- Line 14. ADCON1 has bit 0 cleared so that AN0 is an analogue input.
- Line 19. The ADON bit in ADCON0 turns the A/D convertor on.
- Line 20. ADCON2 bit 7 sets the result to be right justified and uses defaults for the A/D timing.
- Line 22. The program is run in the continuous while loop between lines 23 and 34.
- Line 24 setting the GO_DONE bit in ADCON0 tells the A/D to make a measurement from the selected channel,0.

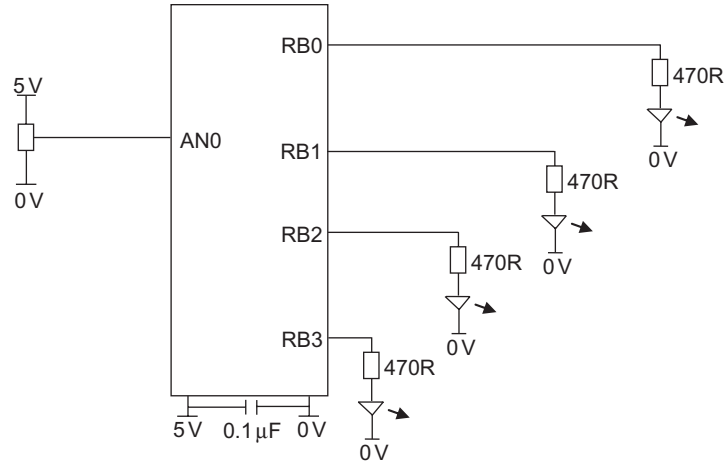


FIGURE 6.12 Voltage indicator.

- Line 25 waits while the GO_DONE bit is equal to 1 (= =), i.e., until it is cleared when the measurement is made.
- Line 26 stores the A/D reading in READING.
- Line 27 computes the TEMPERATURE from the READING.
- Line 29 asks if the TEMPERATURE is less than or equal to 0°C then make PORTB=0b00000011.
- Line 30 asks if the TEMPERATURE is greater than 0°C or less than or equal to 18°C then make PORTB=0b00000010.
- Line 31 asks if the TEMPERATURE is greater than 18°C or less than or equal to 25°C then make PORTB=0b00000000.
- Line 32 asks if the TEMPERATURE is greater than 25°C or less than or equal to 30°C then make PORTB=0b00001000.
- Line 33 asks if the TEMPERATURE is greater than 30°C then make PORTB=0b00001100.

QUESTION

Figure 6.12 shows a Microcontroller measuring a voltage ranging from 0 to 5 V. Write a program that will switch on the LEDs for every volt input.

The solution to this is shown below in Voltage.c. The lines of code required from you are lines 26–32. Copy and paste lines 1–25. (alter line 7), from Temperature.C.

```
1    //Voltage.c, DW Smith, 1st December 2011
2    #include <p18f1220.h>
3    #pragma config WDT=OFF, OSC=INTIO2, PWRT=ON, LVP=OFF, MCLRE=OFF
4    #include <delays.h>
5
```

```
6   int READING;
7   float VOLTAGE;              //Voltage is a decimal number not an integer.
8
9   void main (void)
10  {
11  //SET UP
12  // OSCCON defaults to 31 kHz. So no need to alter it.
13  ADCON1=0x7E;                //AN0 is analogue or 0b01111110 in binary
14  TRISA=0b11111111;           //sets PORTA as all inputs, bit0 is AN0
15  PORTA=0b00000000;           //turns off PORTA outputs, not required, no outputs
16  TRISB=0b00000000;           //sets PORTB as all outputs
17  PORTB=0b00000000;           //turns off PORTB outputs, good start position
18  ADCON0bits.ADON=1;          //turn on A/D
19  ADCON2=0b10000000;          //right justified, acquisition times are ok at 0 with 31 kHz
20
21  while (1)
22    {
23  ADCON0bits.GO_DONE=1;               // do A/D measurement
24  while (ADCON0bits.GO_DONE==1); //wait until bit=0 measurement completed
25  READING=ADRESL+(ADRESH * 256);
26  VOLTAGE=READING/ 204.6;
27
28  if (VOLTAGE>0 && VOLTAGE <= 1)        PORTB=0b00000000;
29  if (VOLTAGE>1 && VOLTAGE <= 2)        PORTB=0b00000001;
30  if (VOLTAGE>2 && VOLTAGE <= 3)        PORTB=0b00000011;
31  if (VOLTAGE>3 && VOLTAGE <= 4)        PORTB=0b00000111;
32  if (VOLTAGE>4 && VOLTAGE <= 5)        PORTB=0b00001111;
33    }
34  }
```

USING SEVERAL A–D INPUTS

Suppose we wish to control the temperature and soil moisture content of a greenhouse.

The temperature sensor is connected to AN0 and the soil moisture probe to AN1 as shown in Figure 6.13.

Suppose the heater is required to come on when the temperature is <15°C and the water valve opens when the soil dries out so the voltage on the sensor is >2V.

The code for this Greenhouse routine is shown:

```
1       //Greenhouse.C, DW Smith, 30th January 2012
2       #include <p18f1220.h>
3       #pragma config WDT=OFF, OSC=INTIO2, PWRT=ON, LVP=OFF, MCLRE=OFF
4       #include <delays.h>
5       int READING;
6       float TEMPERATURE;          //Temperature is a decimal number not an integer.
7       void main (void)
8       {
9       //SET UP
10      // OSCCON defaults to 31 kHz. So no need to alter it.
```

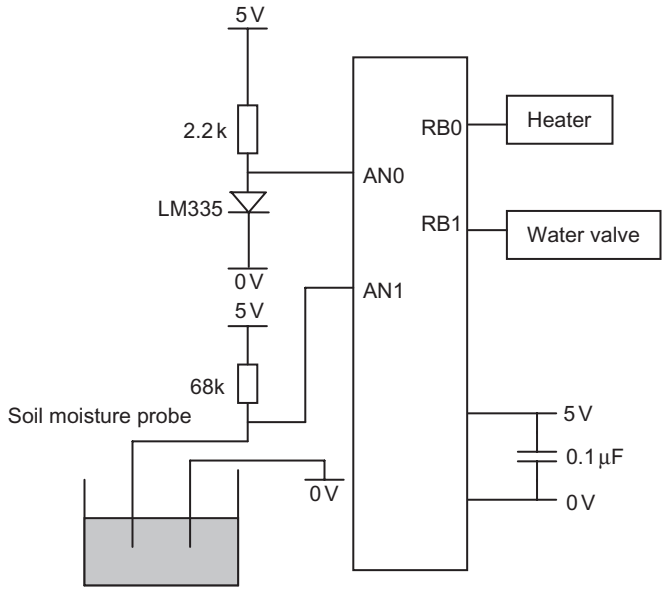


FIGURE 6.13 The greenhouse control.

```
11    ADCON1=0x7C;                //AN0 and AN1 are analogue or 0b01111100 in binary
12    TRISA=0b11111111;           //sets PORTA as all inputs, bit0 is AN0
13    PORTA=0b00000000;           //turns off PORTA outputs, not required, no outputs
14    TRISB=0b00000000;           //sets PORTB as all outputs
15    PORTB=0b00000000;           //turns off PORTB outputs, good start position
16    ADCON0bits.ADON=1;          //turn on A/D
17    ADCON2=0b10000000;          //right justified, acquisition times are OK at 0 with 31 kHz
18    while (1)
19    {     ADCON0bits.CHS0=0;        //selects CH.0
20
21          ADCON0bits.GO_DONE=1;             // do A/D measurement
22    while (ADCON0bits.GO_DONE==1);   //wait until bit=0 for measurement completed
23          READING=ADRESL+(ADRESH * 256);
24          TEMPERATURE=READING/2.046–273.0;
25          ADCON0bits.CHS0=1;                //selects CH.1
26
27          ADCON0bits.GO_DONE=1;          // do A/D measurement
28    while (ADCON0bits.GO_DONE ==1);      //wait until bit=0 for measurement completed
29          READING=ADRESL+(ADRESH * 256);
30    if (TEMPERATURE >=15)    PORTBbits.RB0=0;          //turn heater off
31    if (TEMPERATURE<15)      PORTBbits.RB0=1;          //turn heater on
32    if (READING >= 409)      PORTBbits.RB1=1;          //turn water valve on, 2V=409
33    if (READING<409)         PORTBbits.RB1=0;          //turn water valve off
34    }
35    }
```

Explanation of the Code

- Lines 1–18 are as before.
- Line 19–20 use CHS0 to select CH0 with 0.
- Lines 21–24 take the temperature measurement.
- Lines 25–26 use CHS0 to select CH1 with 1.
- Lines 27–29 take the soil moisture measurement.
- Lines 30–33 deal with the measurements.

SOLAR PANEL HEATING SYSTEM

The solar panel heating system uses a solar panel to heat the water and a pump to pump the water to a tank when the water in the panel is a certain temperature above the tank.

The block diagram of the system is shown in Figure 6.14.

The temperature of the solar panel is measured on AN0.

The temperature of the tank is measured on AN1.

When the temperature of the solar panel is 5°C higher than the tank temperature then the pump is turned on to circulate the water for 1 min.

The temperature measurement is then repeated.

The code for the panel is shown below in **SolarPanel.C**

```
1    //SolarPanel.C, DW Smith, 21st November 2011
2    #include <p18f1220.h>
3    #pragma config WDT=OFF, OSC=INTIO2, PWRT=ON, LVP=OFF, MCLRE=OFF
4    #include <delays.h>
5    int READING;
```

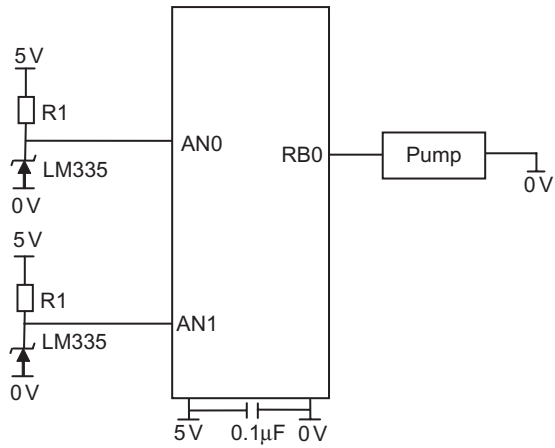


FIGURE 6.14 Solar panel heating system.

```
6   float TempTank;      //Temperature is a decimal number not an integer.
7   float TempPanel;
8   void main (void)
9   {
10      //SET UP
11      // OSCCON defaults to 31 kHz. So no need to alter it.
12      ADCON1=0x7C;            //AN0 and AN1 are analogue or 0b01111100 in binary
13      TRISA=0b11111111;       //sets PORTA as all inputs, bit0 is AN0, bit1 is AN1
14      PORTA=0b00000000;       //turns off PORTA outputs, not required, no outputs
15      TRISB=0b00000000;       //sets PORTB as all outputs
16      PORTB=0b00000000;       //turns off PORTB outputs, good start position
17      ADCON0bits.ADON=1;      //turn on A/D
18      ADCON2=0b10000000;      //right justified, acquisition times are ok at 0 with 31 kHz
19      while (1)
20      {
21      ADCON0bits.CHS0=0;                  // selects CH.0
22      ADCON0bits.GO_DONE=1;               // do A/D measurement
23      while (ADCON0bits.GO_DONE == 1);    // wait until bit=0 for measurement completed
24      READING=ADRESL+(ADRESH * 256);
25      TempPanel=READING/ 2.046–273.0;
26      ADCON0bits.CHS0=1;                  //selects CH.1
27      ADCON0bits.GO_DONE=1;               //do A/D measurement
28      while (ADCON0bits.GO_DONE == 1);    //wait until bit=0 for measurement completed
29      READING=ADRESL+(ADRESH * 256);
30      TempTank=READING/ 2.046–273.0;
31          if (TempPanel—TempTank >= 5)
32          {
33          PORTB=0b00000001;       //pump on
34          Delay10KTCYx(47);       //60 s.
35          }
36          else
37          {
38          PORTB=0b00000000;       //pump off
39          }
40      }
41  }
```

Explanation of the Code

- Lines 1–30 should be obvious from what we did before. The TempPanel reading is taken on channel 0, selected with line 21, and stored in line 25. The TempTank reading is taken on channel 1, selected with line 26, and stored in line 30.
- Lines 31–39 execute code depending on if the panel is 5°C higher than the tank or (else) not.
- Line 40 closes the while(1) loop.
- Line 41 closes the main program.

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-q(1)	R/W-q(1)	R/W-q(1)
—	—	VCFG1	VCFG0	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7-6 **Unimplemented:** Read as '0'

bit 5 **VCFG1:** Voltage Reference Configuration bit ($V_{REF\text{-}source}$)
0 = V_{REF} - (AN2)
0 = Vss

bit 4 **VCFG0:** Voltage Reference Configuration bit ($V_{REF\text{-}source}$)
1 = V_{REF} + (AN3)
1 = V_{DD}

bit 3-0 **PCFG<3:0>:** A/D Port Configuration Control bits:

PCFG3: PCFG0	AN12	AN11	AN10	AN9	AN8	AN7(2)	AN6(2)	AN5(2)	AN4	AN3	AN2	AN1	AN0
0000(1)	A	A	A	A	A	A	A	A	A	A	A	A	A
0001	A	A	A	A	A	A	A	A	A	A	A	A	A
0010	A	A	A	A	A	A	A	A	A	A	A	A	A
0011	D	A	A	A	A	A	A	A	A	A	A	A	A
0100	D	D	A	A	A	A	A	A	A	A	A	A	A
0101	D	D	D	A	A	A	A	A	A	A	A	A	A
0110	D	D	D	D	A	A	A	A	A	A	A	A	A
0111(1)	D	D	D	D	D	A	A	A	A	A	A	A	A
1000	D	D	D	D	D	D	A	A	A	A	A	A	A
1001	D	D	D	D	D	D	D	A	A	A	A	A	A
1010	D	D	D	D	D	D	D	D	A	A	A	A	A
1011	D	D	D	D	D	D	D	D	D	A	A	A	A
1100	D	D	D	D	D	D	D	D	D	D	A	A	A
1101	D	D	D	D	D	D	D	D	D	D	D	A	A
1110	D	D	D	D	D	D	D	D	D	D	D	D	A
1111	D	D	D	D	D	D	D	D	D	D	D	D	D

A = Analog input D = Digital I/O

Note 1: The POR value of the PCFG bits depends on the value of the PBADEN Configuratin bit. When PBADEN = 1, PCFG<2:0> = 000; when PBADEN = 0, PCFG<2:0> = 111.
2: AN5 through AN7 are available only on 40/44-pin devices.

FIGURE 6.15 ADCON1 for the 18F2420/2520 and PIC18F4420/4520.

MICROCONTROLLERS WITH MORE A/D INPUTS

Larger microcontrollers can have more than 7A/D inputs such as the PIC18F2420 and PIC18F2520 that have 10A/D inputs. The PIC18F4420 and PIC18F4520 have 13A/D inputs while the PIC18(L)F43K22 has 28A/D inputs.

For the 18F2420/2520 and PIC18F4420/4520 the analogue inputs are selected by ADCON1 as in the 18F1220, but the 10 and 13A/D inputs, respectively, cannot be individually selected by 8 bits, so only certain combinations are allowable.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ANSELA	-	-	ANSA5	-	ANSA3	ANSA2	ANSA1	ANSA0
ANSELB	-	-	ANSB5	ANSB4	ANSB3	ANSB2	ANSB1	ANSB0
ANSELC	ANSC7	ANSC6	ANSC5	ANSC4	ANSC3	ANSC2	-	-
ANSELD	ANSD7	ANSD6	ANSD5	ANSD4	ANSD3	ANSD2	ANSD1	ANSD0
ANSELE	-	-	-	-	-	ANSE2	ANSE1	ANSE0

FIGURE 6.16 Selecting the 28A/D inputs in the PIC18(L)F43K22.

These 16 allowable combinations in ADCON1 are shown in Figure 6.15 selected by 4 bits.

THE PIC18(L)F43K22 28A/D INPUTS

The PIC18(L)F43K22 has all 28A/D inputs individually selectable, not of course, by 8 bits in ADCON1 but with 5 ANalogue SELect registers—ANSELA, ANSELB, ANSELC, ANSELD, and ANSELE as shown in Figure 6.16.

ADCON1 in the PIC18(L)F43K22 is concerned with selecting the voltage references.