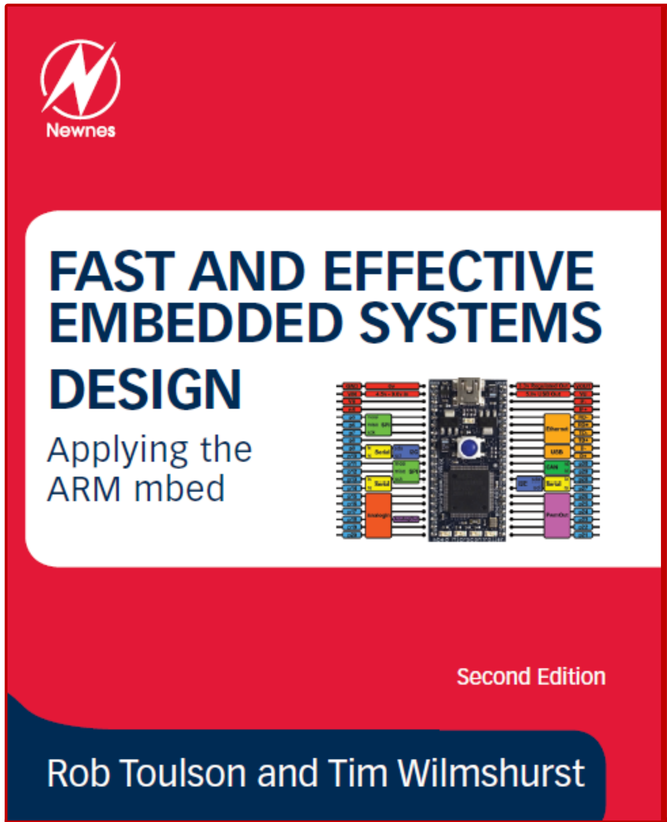


Chapter 5: Analog Input

tw rev. 30.8.16

If you use or reference these slides or the associated textbook, please cite the original authors' work as follows:

Toulson, R. & Wilmshurst, T. (2016). Fast and Effective Embedded Systems Design - Applying the ARM mbed (2nd edition), Newnes, Oxford, ISBN: 978-0-08-100880-5.

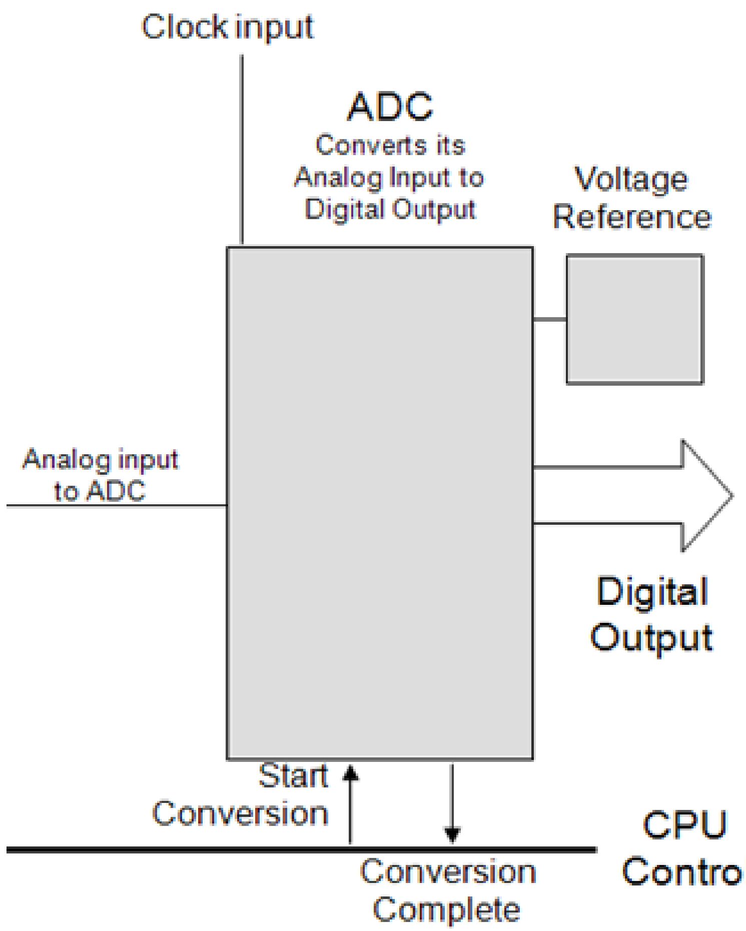
www.embedded-knowhow.co.uk

The Analog-to-Digital Converter (ADC)

An ADC is an electronic circuit whose digital output is proportional to its analog voltage input. Effectively it "measures" the input voltage, and gives a binary output number proportional to its size.

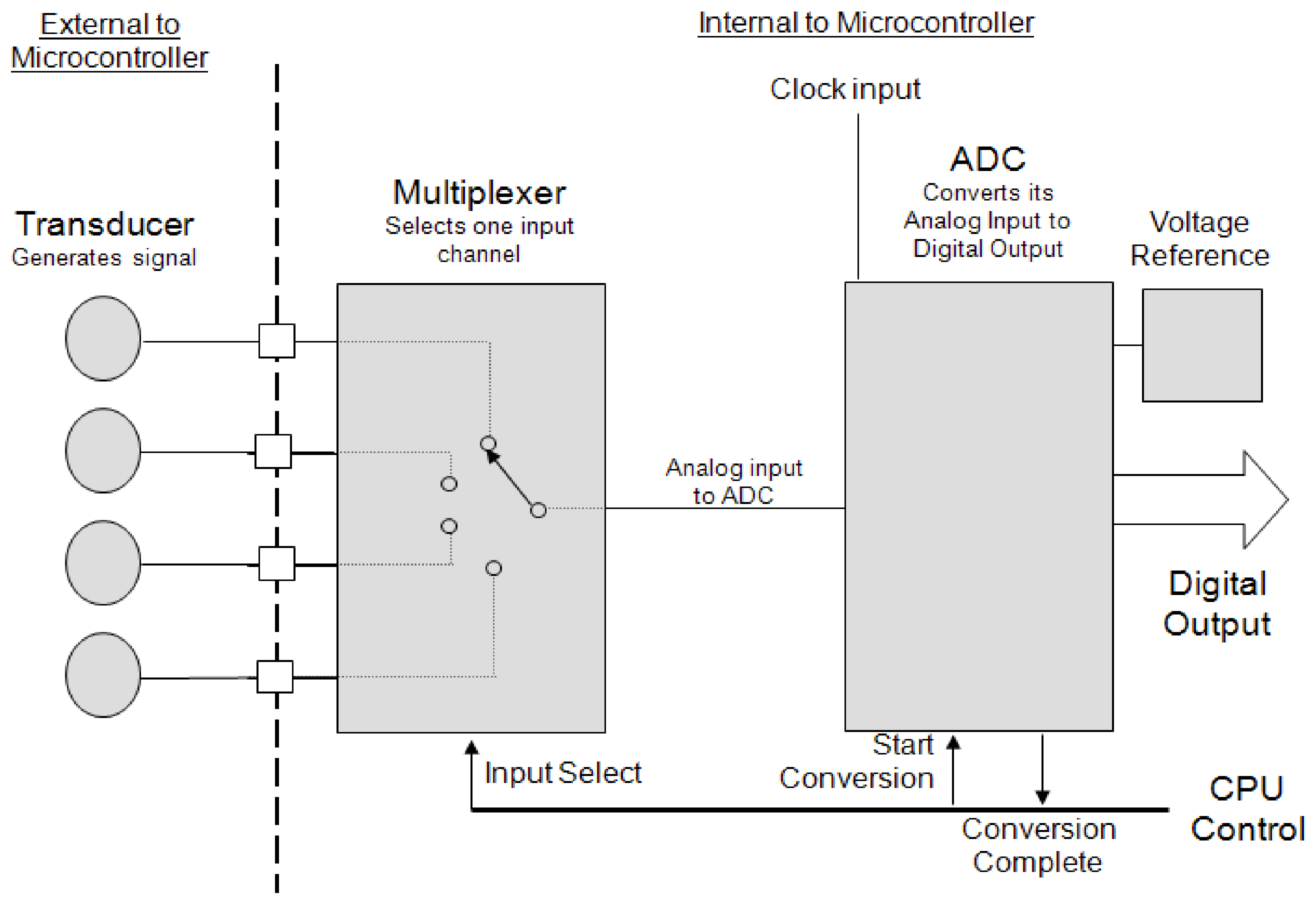
The ADC works with a voltage reference. This is like a ruler or tape measure. In one way or other the ADC compares the input voltage with the voltage reference, and comes up with the output number depending on this comparison.

The conversion takes time, maybe some micro-seconds or more, so the ADC needs to signal when it has finished.



The Data Acquisition System

The ADC almost always operates within a larger environment, often called a data acquisition system. Some features of a general purpose system are shown here.



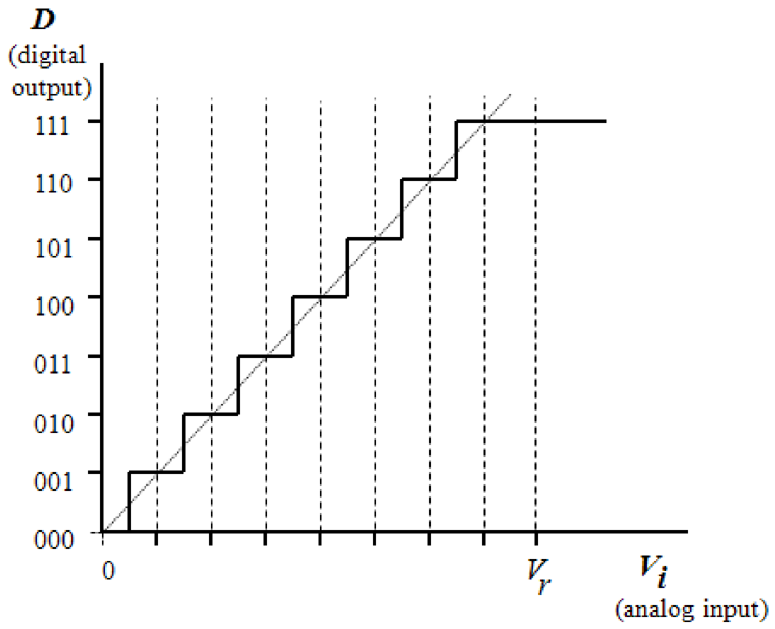
Range, Resolution and Quantisation

The ADC action follows Equation 5.1, and is shown graphically. The output binary number *D* is an integer, and for an n-bit number can take any value from 0 to (2^n – 1).

The difference between the maximum and minimum permissible input values is called the *Range.* Often the minimum value is 0 V, so the range is then just the maximum possible input value.

The range of the ADC is directly linked to the value of the voltage reference; in many ADC circuits the range is equal to the reference voltage.

The resolution is the size of a single step on the staircase characteristic.



Usually

$$D = \frac{V_i}{V_r} \; x \; 2^n \quad \text{(Equation. 5.1)}$$

Questions from the Quiz

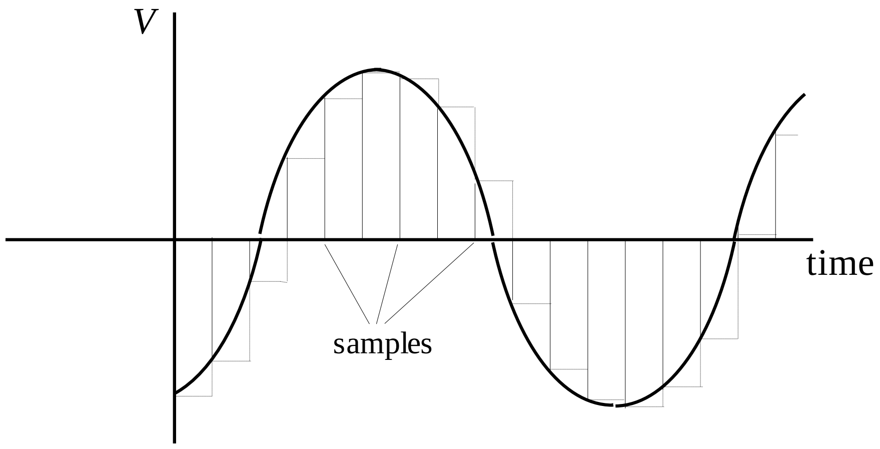
2. An ideal 8-bit ADC has an input range of 5.12 V. What is its resolution, and greatest quantisation error?

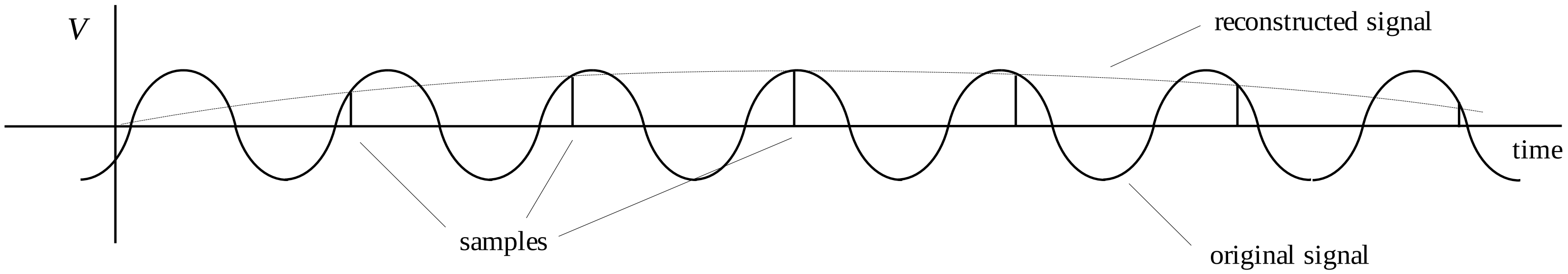
3. An ideal 10-bit ADC has a reference voltage of 2.048 V, and behaves according to Equation 5.1. For a particular input its output reads 10 1110 0001. What is the input voltage?

4. What will be the result if an mbed is required to sample an analog input value of 4.2 V?

Sampling Frequency and Aliasing

When converting an analog signal to digital, we repeatedly take a ‘sample’ and quantise this. The more samples we take, the more accurate the digital data will be. We generally sample at a fixed frequency, called the sampling frequency.





The Nyquist sampling criterion states that the sampling frequency must be at least double that of the highest signal frequency. If the sampling criterion is not satisfied, then *aliasing* occurs – a new lower frequency is generated, as illustrated.

Questions from the Quiz

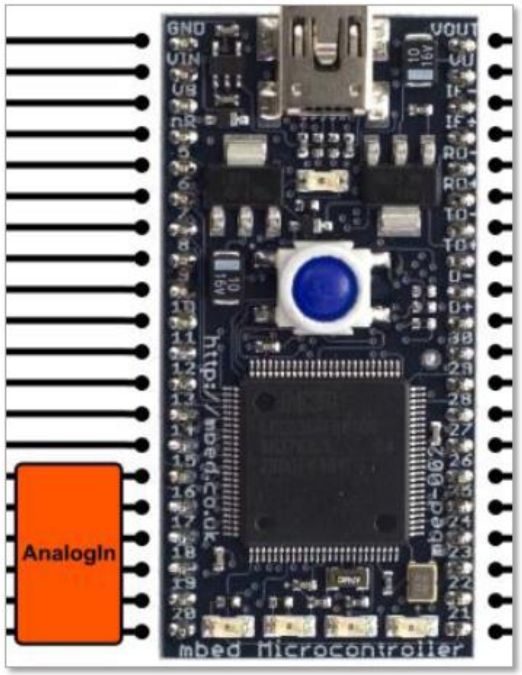
6. An ultrasound signal of 40 kHz is to be digitised. Recommend the minimum sampling frequency.

7. The conversion time of an ADC is found to be 7.5 us. The ADC is set to convert repeatedly, with no other programming requirements. What is the maximum frequency signal it can digitise?

8. The ADC in Question 7 is now used with a multiplexer, so that 4 inputs are repeatedly digitised in turn. A further time of 2500 ns per sample is required, to save the data and switch the input. What is the maximum frequency signal that can now be digitised?

Analog Input with the mbed

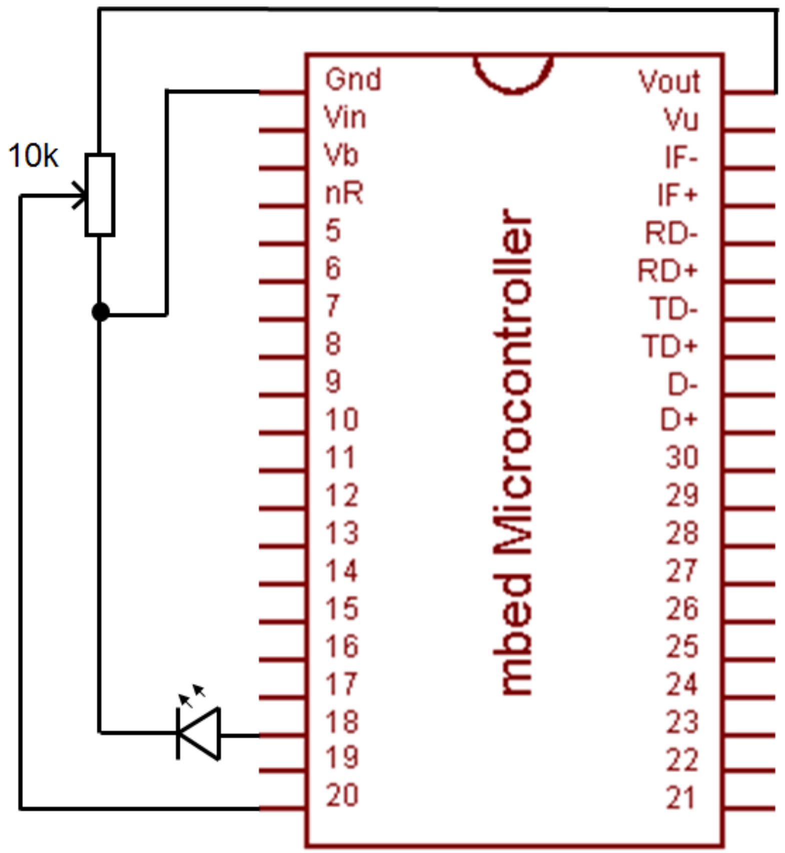
The LPC1768, and hence the mbed, has a single 12-bit ADC, with multiplexer. Its voltage reference is the supply voltage, 3.3 V. The available input pins on the mbed are shown opposite, with API utilities below. The ADC output is available in either unsigned binary (as it would be at the ADC output), or as a floating point number.

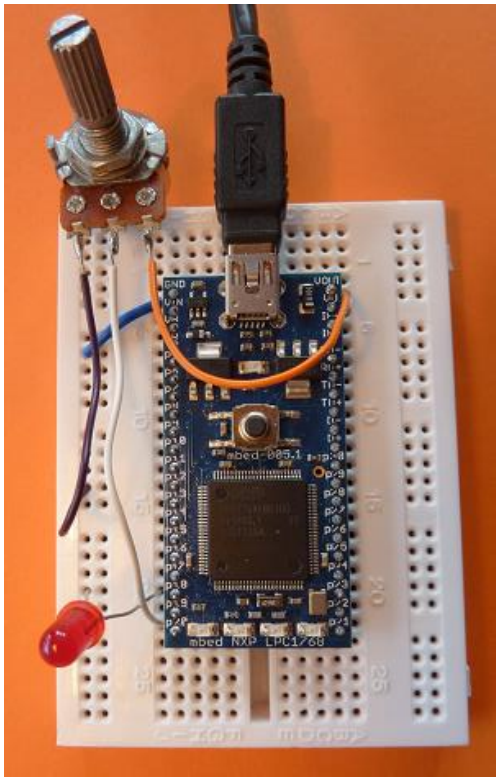


Functions	Usage
AnalogIn	Create an AnalogIn object, connected to the specified pin
read	Read the input voltage, represented as a float in the range (0.0 - 1.0)
read_u16	Read the input voltage, represented as an unsigned short in the range (0x0 - 0xFFFF)

Controlling LED Brightness by Variable Voltage

This simple application reads the analog input, and uses it to control the brightness of an LED by varying the voltage drive to the LED.





Controlling LED Brightness by Variable Voltage

```
/*Program Example 5.1: Uses analog input to control LED brightness,
through DAC output

*/
#include "mbed.h"
AnalogOut Aout(p18);       //defines analog output on Pin 18
AnalogIn Ain(p20)          //defines analog input on Pin 20

int main() {
  while(1) {
    Aout=Ain;     //transfer analog in value to analog out, both
                                                //are type float
  }
}
```

Controlling LED Brightness by PWM

The potentiometer can be used in a similar way to alter the PWM duty cycle. This Program Example will run on the app board, lighting the red LED. Alternatively use the previous circuit, except that the LED should be connected to the PWM output on pin 23. The LED brightness should again be controlled by the potentiometer.

```
/*Program Example 5.2: Uses analog input to control PWM duty
cycle, fixed period

*/
#include "mbed.h"
PwmOut PWM1(p23);
AnalogIn Ain(p20);          //defines analog input on Pin 20

int main() {
  while(1){
    PWM1.period(0.010);  // set PWM period to 10 ms
    PWM1=Ain;            //Analog in value becomes PWM duty, both
are type float
    wait(0.1);
  }
}
```

Controlling PWM Frequency

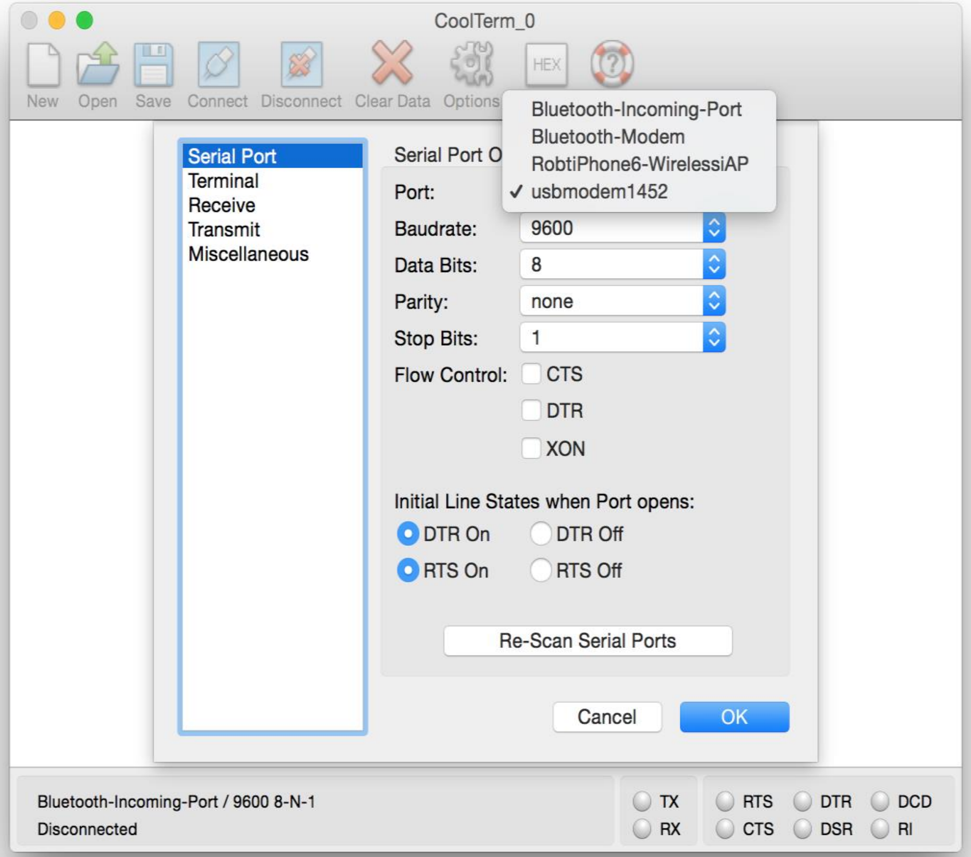
The potentiometer can again be used to alter the PWM frequency, applying this program. This can run on app board or breadboard.

```
/*Program Example 5.3: Uses analog input to control PWM period.
                                                                */

#include "mbed.h"
PwmOut PWM1(p23);
AnalogIn Ain(p20);

int main() {
    while(1){
      PWM1.period(Ain/10+0.001);            // set PWM period
      PWM1=0.5;                             // set duty cycle
      wait(0.5);
    }
}
```

An Interlude - Displaying Values on the Computer Screen

It is possible to print values from the mbed to the PC screen, so any mbed-generated data can be displayed. Both mbed and computer need to be configured to send and receive data. For the computer we need a terminal emulator. The mbed site recommends Tera Term for Windows users, or CoolTerm for Apple OS X developers. Appendix E tells you how to set this up.

The mbed can then be made to appear to the computer as a serial port, communicating through the USB connection. It links up with the USB through one of its own asynchronous serial ports.



Port: usbmodem1452
Baudrate: 9600
Data Bits: 8
Parity: none
Stop Bits: 1
Flow Control: ☐ CTS ☐ DTR ☐ XON

Initial Line States when Port opens:
[✓ DTR On] DTR Off
[✓ RTS On] RTS Off

Bluetooth-Incoming-Port / 9600 8-N-1
Disconnected

Displaying Values on the Computer Screen

```
/*Program Example 5.4: Reads input voltage through the ADC, and transfers to PC
terminal
                                                                     */
#include "mbed.h"
Serial pc(USBTX, USBRX);                        //enable serial port which links to USB
AnalogIn Ain(p20);

float ADCdata;

int main() {
  pc.printf("ADC Data Values...\n\r"); //send an opening text message
  while(1){
    ADCdata=Ain;
    wait(0.5);
    pc.printf("%1.3f \n\r",ADCdata);  //send the data to the terminal
  }
}
```

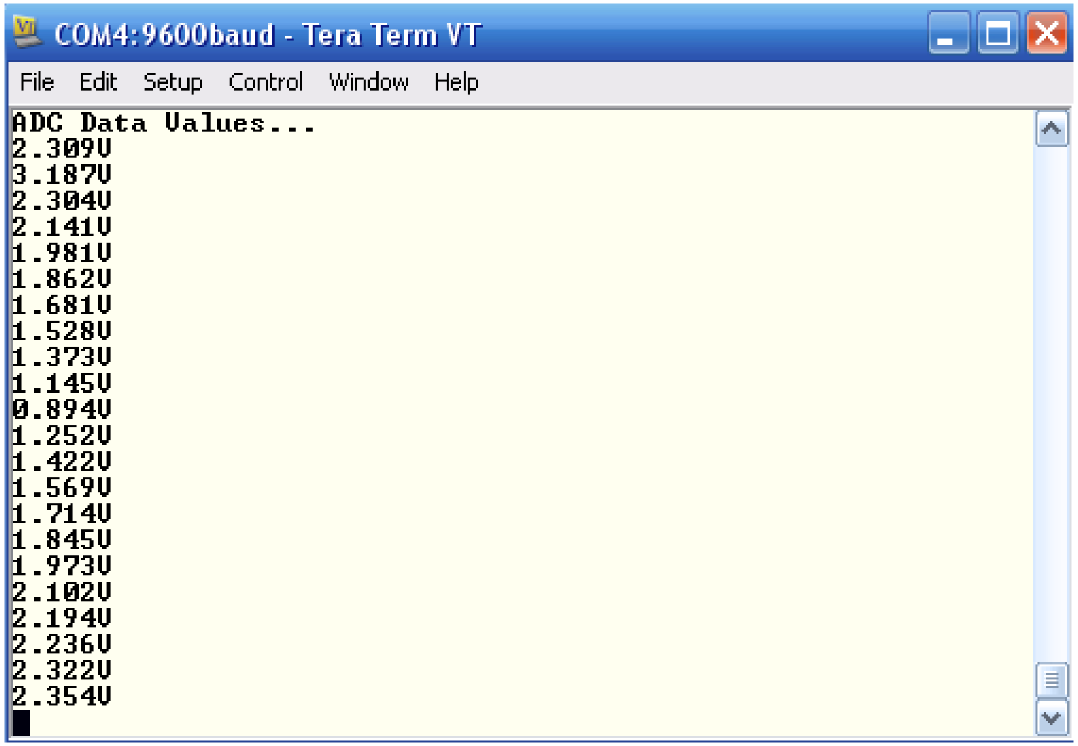
C code feature

This program uses the mbed API to set up the serial link, explained in Chapter 7. It also uses the **printf( )** function for the first time, along with some of its far-from-friendly format specifiers. Check Section B9 (Appendix B) for some background on this.

Scaling ADC Outputs to Recognised Units

Multiplying the float value read from the ADC by 3.3 converts the result into a voltage reading. These lines can be added to the previous program.

```
ADCdata=Ain*3.3;
wait(0.5);
pc.printf("%1.3f",ADCdata);
pc.printf("V\n\r");
```


COM4:9600baud - Tera Term VT
File Edit Setup Control Window Help

```
ADC Data Values...
2.309V
3.187V
2.304V
2.141V
1.981V
1.862V
1.681V
1.528V
1.373V
1.145V
0.894V
1.252V
1.422V
1.569V
1.714V
1.845V
1.973V
2.102V
2.194V
2.236V
2.322V
2.354V
```

The resulting display

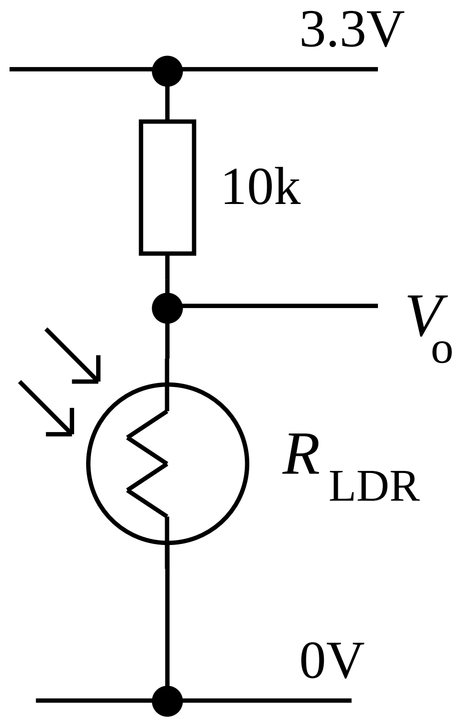
Applying Averaging to Reduce Noise

Incoming analog signals may pick up interference. A very simple first step to improve this situation is to average the incoming signal. This should help to find the underlying average value, and remove any high frequency noise. Try inserting the **for** loop shown below, replacing the ADCdata=Ain; line in Program Example 5.4. This sums 10 ADC values, and takes their average. Note that the overall conversion now takes 10 times as long. This is a very simple example of digital signal processing.

```
for (int i=0;i<=9;i++) {
  ADCdata=ADCdata+Ain*3.3;       //sum 10 samples
}
ADCdata=ADCdata/10;              //divide by 10
```

Simple Analog Sensors - the Light Dependent Resistor

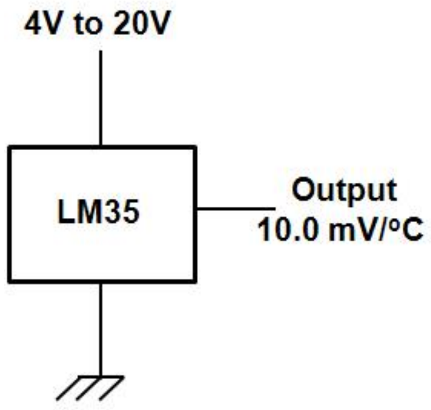
The *light dependent resistor* (LDR) is made from a piece of exposed semiconductor material. When light falls on it, its energy flips some electrons out of the crystalline structure; the brighter the light, the more electrons are released. These electrons are then available to conduct electricity, with the result that the resistance of the material falls. If the light is removed the electrons pop back into their place, and the resistance goes up again. The overall effect is that as illumination increases, the LDR resistance falls.

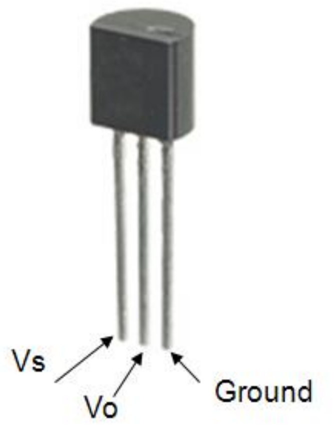


Illumination (lux)	R_{LDR} (Ω)	V_o
Dark	$\geq$ 1.0 M	$\geq$ 3.27 V
10	9k	1.56 V
1,000	400	0.13 V

Integrated Circuit Temperature Sensor

Semiconductor action is highly dependent on temperature, so it’s not surprising that semiconductor temperature sensors are made. A very useful form of sensor is one which is contained in an integrated circuit, such as the LM35. This device has an output of 10 mV/^{o}C, with operating temperature up to 110^{o}C (for the LM35C version). It is thus immediately useful for a range of temperature sensing applications. The simplest connection for the LM35 is shown.





Exploring data Conversion Timing

This program provides a mechanism for measuring conversion times, and then viewing Nyquist's sampling theorem in action. See Exercises 5.7 and 5.8.

```
/*Program Example 5.5: Inputs signal through ADC, and outputs to DAC. View DAC
output on oscilloscope. To demonstrate Nyquist, connect variable frequency signal
generator to ADC input. Allows measurement of conversion times, and explores
Nyquist limit.     */

#include "mbed.h"
AnalogOut Aout(p18);      //defines analog output on Pin 18
AnalogIn Ain(p20);        //defines analog input on Pin 20
DigitalOut test(p5);
float ADCdata;

int main() {
  while(1) {
    ADCdata=Ain;   //starts A-D conversion, and assigns analog value to ADCdata
    test=1;        //switch test output, as time marker
    test=0;
    Aout=ADCdata;  // transfers stored value to DAC, and forces a D-A conversion
    test=1;        //a double pulse, to mark the end of conversion
    test=0;
    test=1;
    test=0;
    //wait(0.001);    //optional wait state, to explore different cycle times
   }
}
```

Chapter Review

- An ADC is available in the mbed; it can be used to digitise analog input signals.
- It is important to understand ADC characteristics, in terms of input range, resolution, and conversion time.
- Nyquist’s sampling theorem must be understood, and applied with care when sampling AC signals. The sampling frequency must be at least twice that of the highest frequency component in the sampled analog signal.
- Aliasing occurs when the Nyquist criterion is not met, this can introduce false frequencies to the data. Aliasing can be avoided by introducing an anti-aliasing filter to the analog signal before it is sampled.
- Data gathered by the ADC can be further processed, and displayed or stored.
- There are numerous sensors available which have an analog output; in many cases this output can be directly connected to the mbed ADC input.