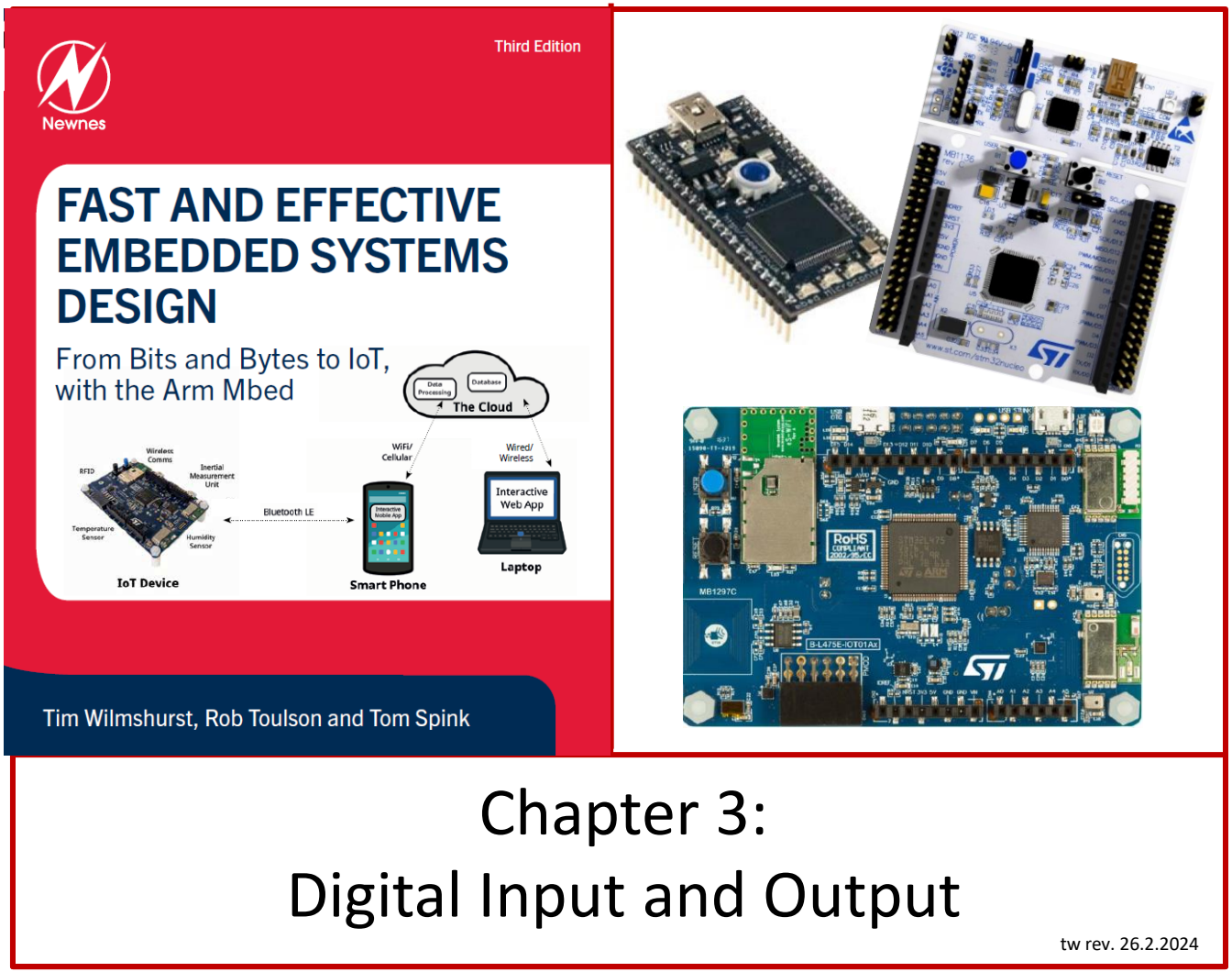


Chapter 3:
Digital Input and Output

tw rev. 26.2.2024

(Note to Instructor: Insert "Essentials of C" (Appendix B) presentation here if required

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

Wilmshurst, T., Spink, T. & Toulson, R. (2024). Fast and Effective Embedded Systems Design – From Bits and Bytes to IoT, with the ARM Mbed (3rd edition), Newnes, Oxford, ISBN: 978-0323951975.

Thinking about the First Program

Our first program is reproduced here, this time with comments inserted.

```
/*Program Example 2.1: A program which flashes target LED1 on and off.
Demonstrates use of digital output and wait functions.
Works with LPC1768, F401RE or STM32 IoT Discovery Kit as target systems*/

#include " mbed.h"    //include the Mbed header file as part of this program

//Blinking rate in milliseconds
#define BLINKING_RATE     500ms

int main() {          //the main function starts here
   // Initialise the digital pin LED1 as an output
   DigitalOut led(LED1);

  while(true) {          //a continuous loop is created
    led = !led;          //toggle led state
    ThisThread::sleep_for(BLINKING_RATE);   //no action for blinking rate duration
  }                      //end of while loop
}                        //end of main function
```

C/C++ code feature

Check your understanding of how this program is structured, its use of **#include, main()** and **while(true)**, and the **DigitalOut** API feature.

A First Word on Waiting and Timing

Timing is of great importance in embedded systems. When required to wait, it reduces power consumption to let the processor enter a “sleep” (more on this later). The **ThisThread::sleep_for(())** function, from the API library, does just this. Further options are shown in the Table. **chrono** is a mechanism used in C++ to define time durations.

API Function	Action
`wait_ns(int ns)`	waits for the number of nanoseconds specified
`wait_us(int us)`	waits for the number of microseconds specified
`thread_sleep_for(int ms)`	sleeps for the number of milliseconds specified
`ThisThread::sleep_for(uint32_t ms)`	sleeps for the number of milliseconds specified
`ThisThread::sleep_for(chrono::microseconds(int us))`	sleeps for the number of microseconds specified
`ThisThread::sleep_for(chrono::milliseconds(int ms))`	sleeps for the number of milliseconds specified
`ThisThread::sleep_for(chrono::seconds(int seconds))`	sleeps for the number of seconds specified
`ThisThread::sleep_for(chrono::minutes(int minutes))`	sleeps for the number of minutes specified

The mbed API, and **DigitalOut**

The Mbed API has a pattern we will see repeated many times. It is accessed through the header file **mbed.h,** included in almost every example program.

The first API component that we look at is **DigitalOut,** summarised in the Table. In a pattern which will become familiar, **DigitalOut** is a C++ *class***.** The class then has a set of member functions, listed below. The first of these is a C++ *constructor,* which must have the same name as the class itself. This can be used to create C++ *objects*.

Functions	Usage
Constructor	
`DigitalOut (PinName pin)`	Create a DigitalOut object of specified name, connected to the specified pin
Member Functions	
`void write (int value)`	Set the output, specified as 0 or 1
`int read()`	Return the output setting, returned as 0 or 1
`Mbed-defined operator: =`	A shorthand for **write()**
`Mbed-defined operator: int()`	A shorthand for **read()**

The mbed API, and **DigitalOut**

In the first example we create the object **myled**. We can then write to it and read from it, using the member functions **write( )** and **read( )**. These are members of the class, so their format is **myled.write( ),** and so on. In the **DigitalOut** class there are also user-defined operators, a feature of C++. These also appear in the Table, for example the assign operator = . Hence when we write

```
myled = 1;
```

the variable value is then not only changed (normal C usage), but it is also written to the digital output. This replaces **myled.write(1);**. We find similar shorthands offered for all peripheral API groups in the mbed API.

Functions	Usage
Constructor	
`DigitalOut (PinName pin)`	Create a DigitalOut object of specified name, connected to the specified pin
Member Functions	
`void write (int value)`	Set the output, specified as 0 or 1
`int read()`	Return the output setting, returned as 0 or 1
`Mbed-defined operator: =`	A shorthand for **write()**
`Mbed-defined operator: int()`	A shorthand for **read()**

(Table repeated for convenience)

Exploring the *while* Loop

```
/*Program Example 3.1: Demonstrates use of while loops. No external connection required.
Works with LPC1768 or F401RE, with output labels as indicated
in Comments
*/

#include " Mbed.h"
DigitalOut myled(LED1);
DigitalOut yourled(LED4); //replace LED4 with A0 for F401RE board

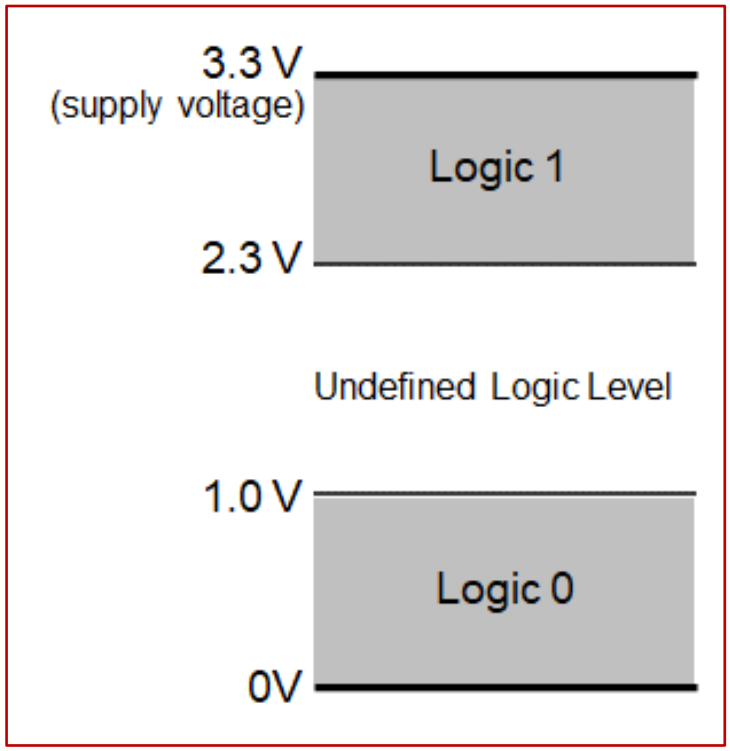
int main() {
  char i=0;          //declare variable i, and set to 0
  while(true){          //start endless loop
  while(i<10) {      //start first conditional while loop
    myled = 1;
    ThisThread::sleep_for(chrono::milliseconds(200));
    myled = 0;
    ThisThread::sleep_for(chrono::milliseconds(200));
    i = i+1;          //increment i
  }                   //end of first conditional while loop
  while(i>0) {        //start second conditional loop
    yourled = 1;
    ThisThread::sleep_for(chrono::milliseconds(200));
    yourled = 0;
    ThisThread::sleep_for(chrono::milliseconds(200));
    i = i-1;
    }
  }                   //end infinite loop block
}                     //end of main
```

C/C++ code feature

Note the declaration of the variable **i**, and the conditional use of **while( ).**

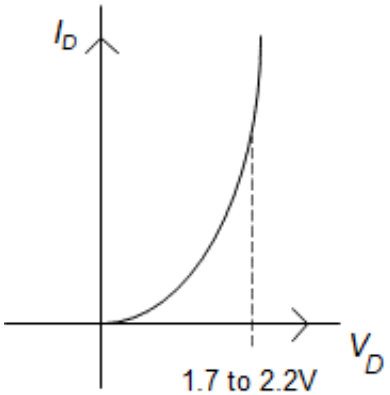
Voltages as Logic Values

In any digital circuit, logic values are represented as electrical voltages. Here now is the BIG benefit of digital electronics: we don't need a precise voltage to represent a logical value. For example, digital inputs of the LPC1678 interpret *any* input voltage below 1.0 V as logic 0, and *any* input voltage above 2.3 V as Logic 1, as illustrated.

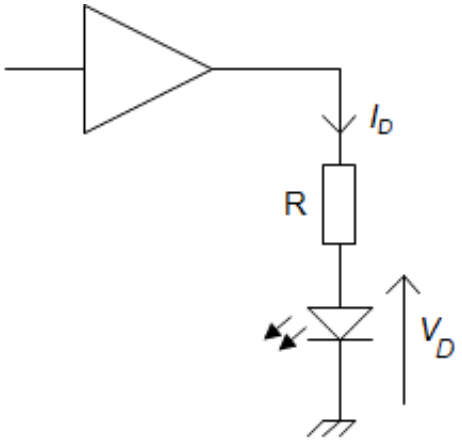


Using LEDs

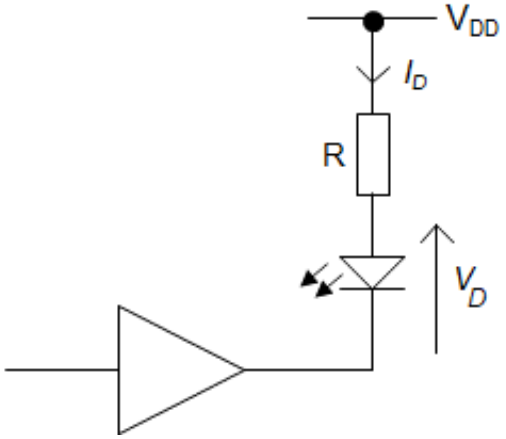
The LED (light emitting diode) has the voltage/current characteristic shown in a). A small forward voltage causes very little current to flow. As the voltage increases there comes a point where the current increases rapidly. Figures b) and c) show circuits used to make direct connections of LEDs to the output of logic gates, for example an Mbed pin configured as an output. Some LEDs, such as the ones recommended for the next program, have the series resistance built into them. They therefore don't need any external resistor connected.



a) Led V-I Characteristic



b) Gate Output Sourcing Current to LED Load



c) Gate Output Sinking Current from LED

Further Example Program using DigitalOut

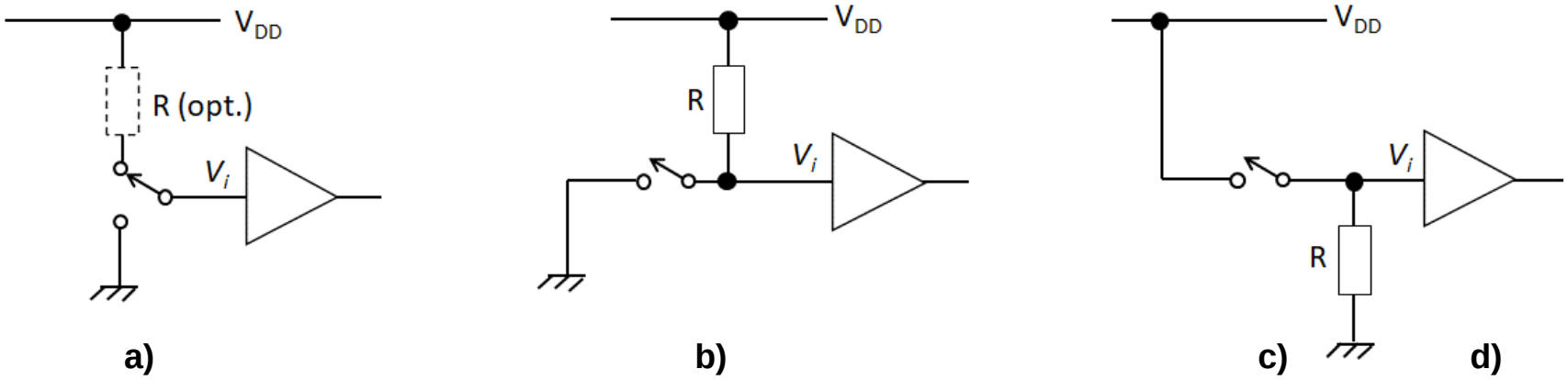
The code uses external LEDs, rather than on-board. It extends ideas already applied, so it should be easy to understand that the green and red LEDs are programmed to flash alternately.

```
/*Program Example 3.2: Flashes red and green LEDs in simple time-based pattern
Works with LPC1768 or F401RE as target systems
    */
#include " Mbed.h"
//define and name digital outputs
DigitalOut redled(p5);     //Replace p5 with A0 for F401RE
DigitalOut greenled(p6);   // Replace p6 with A1 for F401RE

int main() {
  while(true) {
    redled = 1;
    greenled = 0;
    ThisThread::sleep_for(200ms);
    redled = 0;
    greenled = 1;
    ThisThread::sleep_for(200ms);
  }
}
```

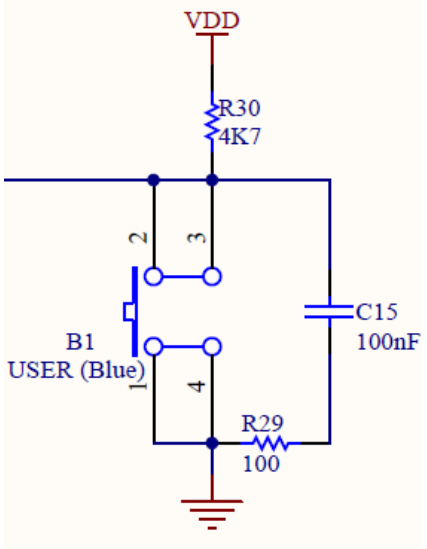
Digital Inputs: Connecting Switches to a Digital System

We can use ordinary electromechanical switches to create logic levels, which will satisfy the logic level requirements seen earlier. Three commonly used ways are shown here.



a) **b)** **c)** **d)**

a) Single-pole double-throw (SPDT) connection.
b) Single-pole single-throw (SPST) with pull-up resistor.
c) SPST with pull-down resistor
Note: the diagrams show a logic buffer, which can in each case be a microcontroller/Mbed input.
d) the user button on the Nucleo F401RE board
(image courtesy of ST)



The **DigitalIn** API

The Mbed API has the digital input functions listed here, with format identical to that of **DigitalOut,** already seen.

Note that use of **DigitalIn** enables by default the internal pull-down resistor, i.e. the input circuit is configured as Figure (c) in previous slide. This can be disabled, or an internal pull-up enabled, using the **mode( )** function.

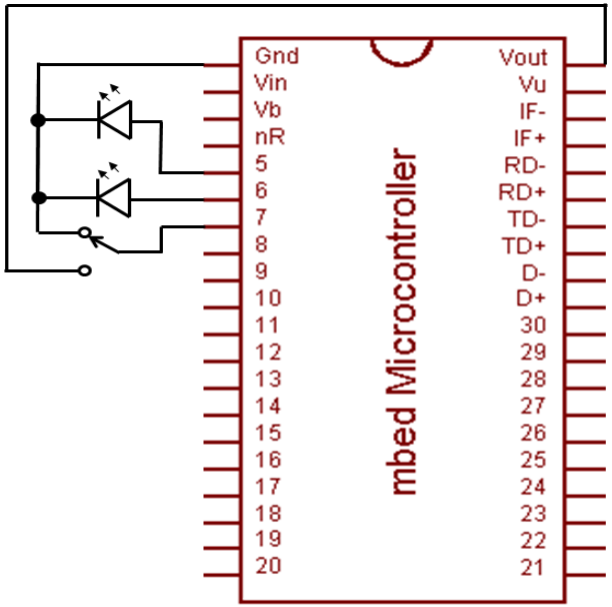
Functions	Usage
Constructor	
`DigitalIn (PinName pin, PinMode mode)`	Create a DigitalIn object of specified name, connected to the specified pin, input mode may also be specified.
Member Functions	
`int read()`	Read the input, returned as 0 or 1.
`void mode(PinMode)`	Set the input pin mode, with parameter chosen from: PullUp, PullDown, PullNone, OpenDrain.
`Mbed-defined operator: int()`	A shorthand for read ()

Using *if* to Respond to a Switch Input

```
/*Program Example 3.3: Flashes one of two LEDs, depending on the state of switch. Works
with either LPC1768 or F401RE using pins shown.
*/

#include " mbed.h"
DigitalOut redled(p5);      //Replace p5 with A0 for F401RE
DigitalOut greenled(p6);    // Replace p6 with A1 for F401RE
DigitalIn  switchinput(p7);  // Replace p7 with BUTTON1 for F401RE

int main() {
  while(true) {
    if (switchinput == 1) {      //test value of switchinput
      //execute following block if switchinput is 1
      greenled = 0;              //green led is off
      redled = 1;                // flash red led
      ThisThread::sleep_for(200ms);
      redled = 0;
      ThisThread::sleep_for(200ms);
    }                            //end of if
    else {                       //here if switchinput is 0
      redled = 0;                //red led is off
      greenled = 1;              // flash green led
      ThisThread::sleep_for(200ms);
      greenled = 0;
      ThisThread::sleep_for(200ms);
    }                            //end of else
  }                              //end of while(true)
}                                //end of main
```



C/C++ code feature

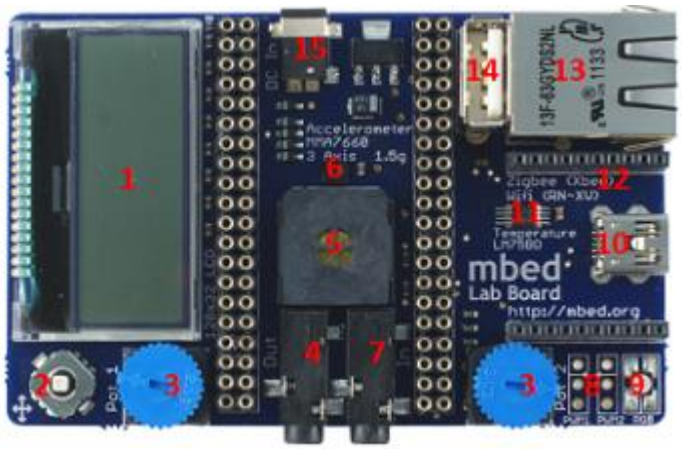
Check carefully the use of **if** and **else**, to allow selection of actions.

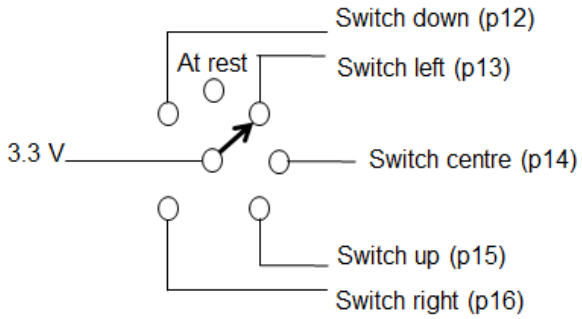
Digital Input and Output with the Application Board

The application board has a Joystick, which is a set of switches, and a tricolour LED. These are both symbolised below, with Mbed LPC1768 pin connections.

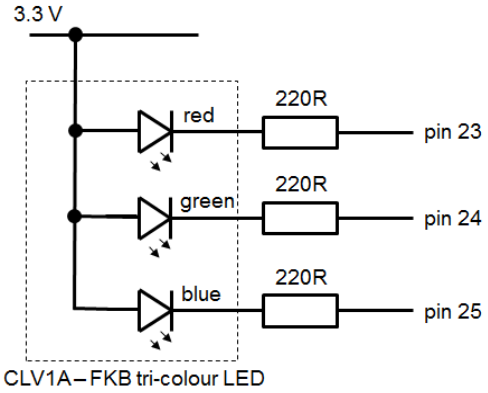
The joystick (item 2 in figure) acts as a 5-position multi-way switch, with the wiper connected to the supply voltage.

The tri-colour LED (item 9 in figure) connects through current-limiting resistors to mbed pins. The LED is a “common anode” type, which means the three internal LED anodes are connected (“commoned”) together, and available externally as a single connection. It requires a logic 0 on the Mbed pin to light the LED.





(a) Joystick



(b) Three-colour LED

Controlling application board LED with the Joystick

```
/*Program Example 3.4. Uses Joystick values to switch tricolor led
Works with LPC1768 or F401RE connected to application board*/

#include "mbed.h"
//set up outputs to leds in tricolor led
DigitalOut redled(p23);    //Replace p23 with D2 for F401RE
DigitalOut greenled(p24);  //Replace p24 with D3 for F401RE
DigitalOut blueled(p25);   //Replace p25 with D4 for F401RE
//set up joystick connections, left and right
DigitalIn  joyleft(p13);  //Replace p13 with D5 for F401RE
DigitalIn  joyright(p16); //Replace p16 with D6 for F401RE

int main() {
  greenled = redled = blueled = 1; //switch all leds off (logic 1 for off)
  redled = 0;         //switch red led on, diagnostic (logic 0 for on)
  ThisThread::sleep_for(1s);       //wait one second
  redled = 1;         //switch red led off
  while(true) {
    if (joyleft == 1) {    //test if the joystick is pushed left
      greenled = 0;        //switch green led on
      ThisThread::sleep_for(1s);       //wait one second
      greenled = 1;        //switch green led off
    }
    if (joyright == 1) {   //test if the joystick is pushed right
      redled = 0;          //switch red led on
      ThisThread::sleep_for(1s);
      redled = 1;          //switch red led off
    }
  }
}
```

Controlling Mbed LEDs from the App Board Joystick

It's useful to be able to group digital inputs or outputs, so that we can read or switch whole groups at the same time. **BusIn** allows you to group a set of digital inputs into one bus, so that you can read a digital word direct from it. The equivalent output is **BusOut**. To apply either simply specify a name, and then list in brackets the pins which will be members of that bus.

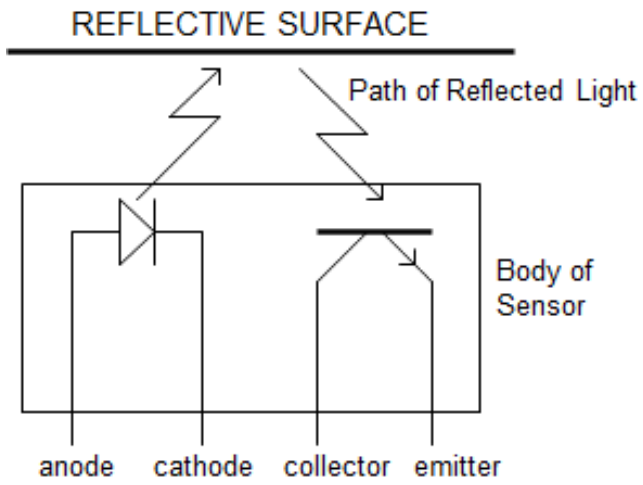
```
/*Program Example 3.5: Transfers the value of the joystick to target LEDs
Works with LPC1768 in application board, or F401RE with appropriate connections*/

#include "mbed.h"

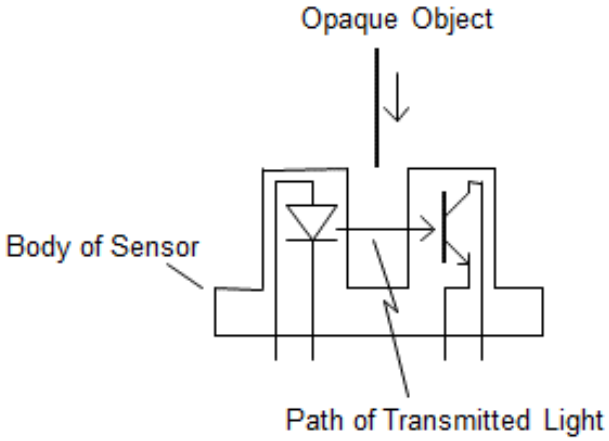
BusIn joystick(p15,p12,p13,p16); //Select own pins for F401RE (any digital I/O)
DigitalIn fire(p14);
BusOut leds(LED1,LED2,LED3,LED4); //Select own pins for F401RE (any digital I/O)

int main(){
  while(true){
    if (fire) {
      leds = 0xf;
      }
    else {
      leds = joystick;
      }
    ThisThread::sleep_for(100ms);
  }
}
```

Interfacing Simple Opto Devices

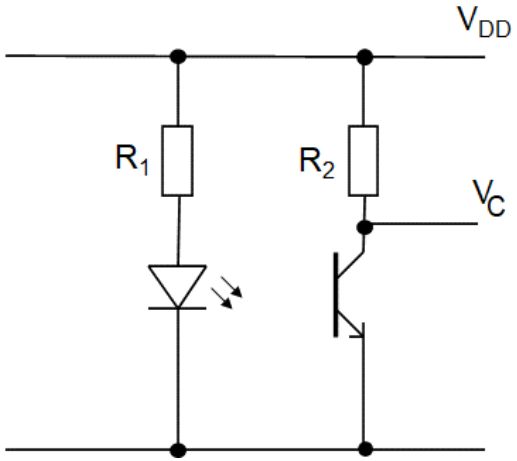


a) The reflective opto-sensor



b) The transmissive opto-sensor

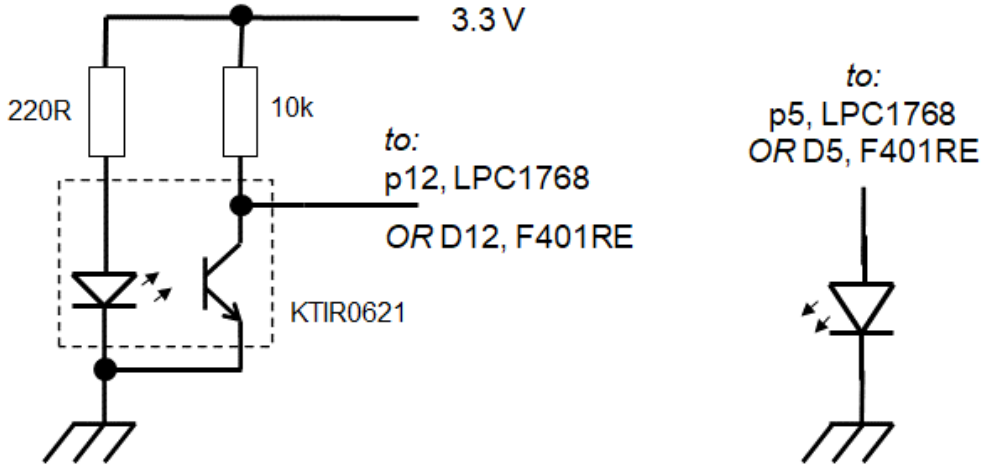
Opto-sensors are simple examples of sensors with "almost" digital outputs. When a light falls on the base of an opto-transistor, it conducts; when there is no light it doesn't. Either of the sensors shown can be connected in the circuit of (c).

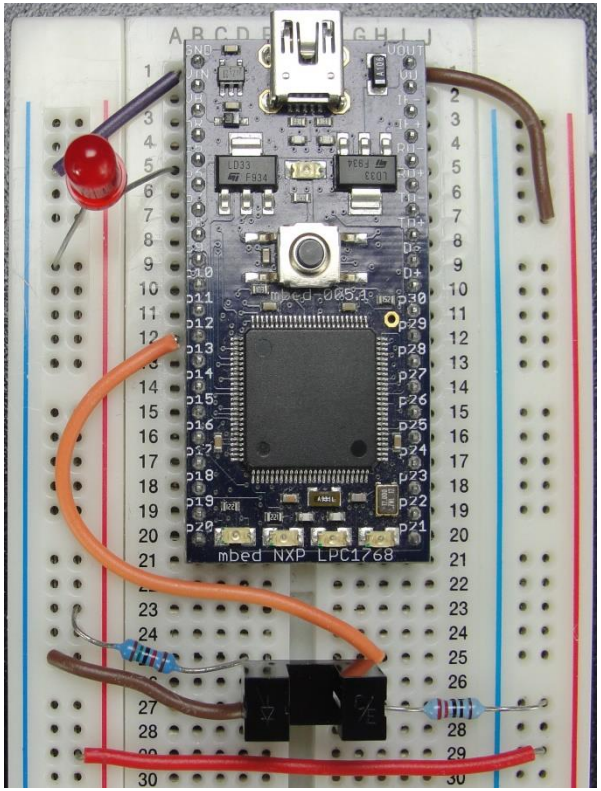


c) Simple drive circuit for opto-sensor

Connecting an Opto-Sensor to an Mbed Target

The figure shows how a transmissive opto-sensor can be connected. The device used is a KTIR0621DS, made by Kingbright; similar devices can be used.





Applying the Photo-Interrupter

This simple program applies connections shown in the previous slide. The program switches the LED on when the beam is interrupted, i.e. an object has been sensed. To make the selection, we use the **if** and **else** keywords, as in the previous example. Now however there is just one line of code to be executed for either state. It is not necessary to use braces to contain these single lines.

```
/*Program Example 3.6: Simple program to test KTIR slotted opto-sensor.
Switches an LED according to state of sensor
Works with LPC1768 or F401RE, using connections indicated*/

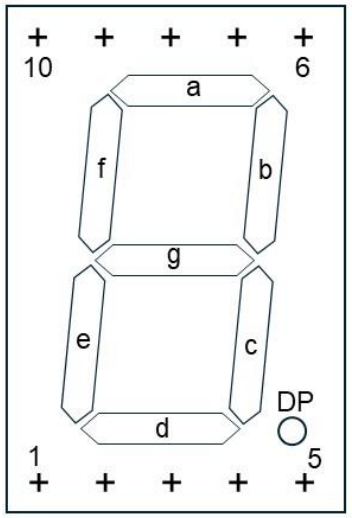
#include "mbed.h"
DigitalOut redled(p5);          //Replace p5 with D5 for F401RE
DigitalIn  opto_switch(p12);    //Replace p12 with D12 for F401RE

int main() {
  while(true) {
    if (opto_switch == 1)       //input = 1 if beam interrupted
      redled = 1;               //switch led on if beam interrupted

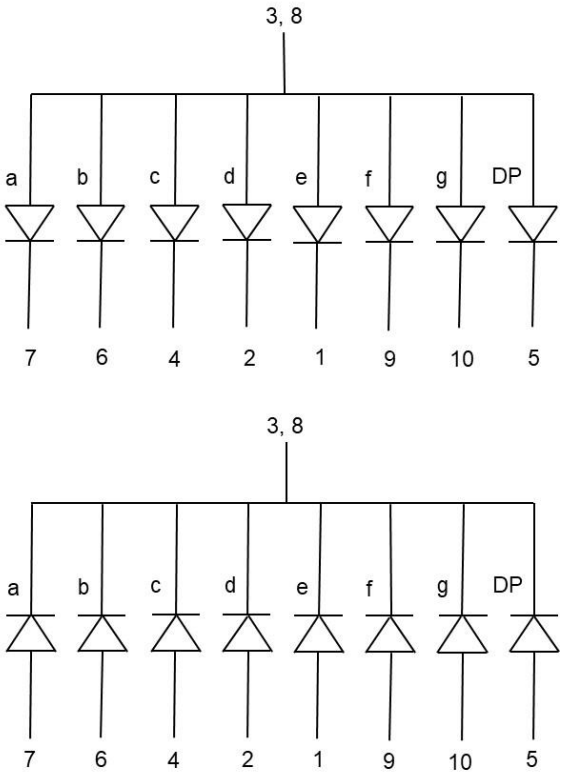
    else
      redled = 0;               //led off if no interruption
  }                             //end of while
}
```

Seven-Segment Displays

The seven-segment display is a versatile configuration. By lighting different combinations of the segments, all numerical digits can be displayed, as well as a surprising number of alphabetic characters. A decimal point is usually included, as shown. This means that there are eight LEDs in the display, needing 16 connections. To simplify matters, either all LED anodes (“common anode”) are connected together, or all LED cathodes (“common cathode”).







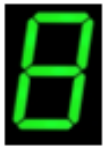
upper: common anode; lower: common cathode

Driving the Display

A small seven-segment display can be driven directly from a microcontroller. In the case of common cathode, the cathode is connected to ground, and each segment is connected to a port pin. If the segments are connected in this sequence to form a byte,

(MSB) DP g f e d c b a (LSB)

then the values shown in the Table apply. For example, if 0 is to be displayed, then all outer segments, i.e. abcdef must be lit, with the corresponding bits from the microcontroller set to 1.

Display Value	0	1	2	3	4	5	6	7	8	9
Segment Drive (B) (MSB) (LSB)	0011 1111	0000 0110	0101 1011	0100 1111	0110 0110	0110 1101	0111 1101	0000 0111	0111 1111	0110 1111
B (hex)	0x3F	0x06	0x5B	0x4F	0x66	0x6D	0x7D	0x07	0x7F	0x6F
Actual Display										

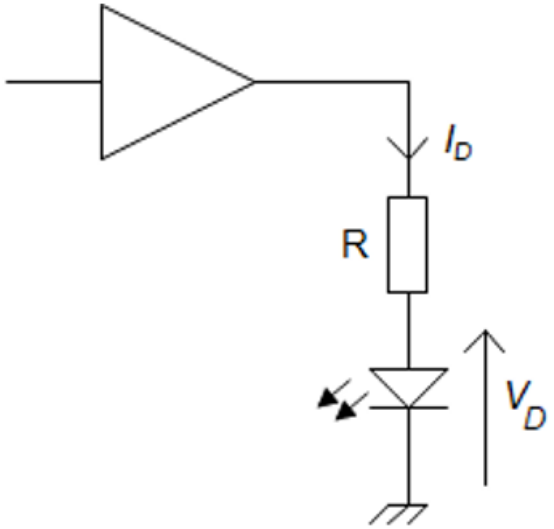
Mbed Targets Connected to a Common Cathode Display

The mbed runs from a 3.3 V supply. Each display LED requires around 1.8 V across it in order to light. This is within the Mbed capability. If there were two LEDs in series in each segment however, the Mbed would barely be able to switch them into conduction.

The internal resistance of the Mbed output pins can be shown to be very approximately 100 Ω.

Calculating as shown, the current per diode is around 15 mA, a reasonable value for an LED.

$$I_D \cong (3.3 - 1.8)/100 = 15 \text{ mA}$$



Driving the Display, and further Programming Features

```
/*Program Example 3.7: Simple demonstration of 7-segment display. Displays digits
0, 1, 2, 3 in turn.
Works for LPC1768 and F401RE, using connections indicated
                                                                        */
#include "mbed.h"
//Configure segments a,b,c,d,e,f,g,dp
//Pin listing for BusOut goes from lsb to msb, so lsb is pin 5
BusOut display(p5,p6,p7,p8,p9,p10,p11,p12);
                                //D13,D12,D11,D10,D9,D8,D7,D6 for F401RE

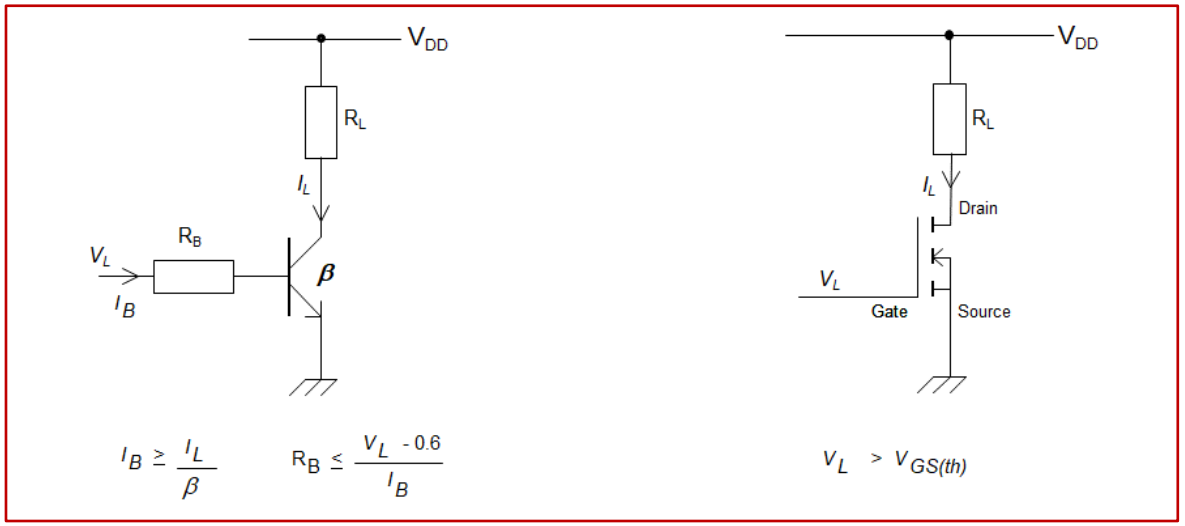
int main() {
  while(true) {
    for(int i = 0; i < 4; i++) {
      switch (i){
        case 0: display = 0x3F; break;       //display 0
        case 1: display = 0x06; break;       //display 1
        case 2: display = 0x5B; break;
        case 3: display = 0x4F; break;
      }                                      //end of switch
      ThisThread::sleep_for(500ms);
    }                                        //end of for
  }                                          //end of while
}                                            //end of main
```

C/C++ code feature

Note the use of the **for( )** loop, an alternative to **while(**) for creating conditional loops. Using **switch( )** allows selection of one of a range of options within a list, identified with the **case** keyword. Each option is terminated with **break.**

Switching Larger DC loads

Mbed target boards can drive simple DC loads directly with its digital I/O pins. If it's necessary to drive a load - say a motor - which needs more current than an port pin can supply, or which needs to run from a higher voltage, then an interface circuit will be needed.



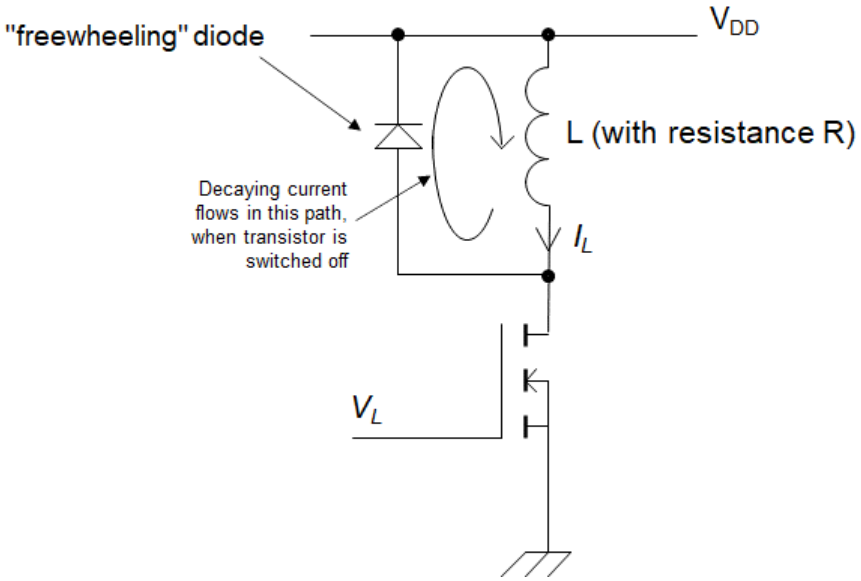
A good switching transistor for small DC loads is the ZVN4206A, with main characteristics shown. An important value is the maximum V_{GS} threshold value, shown as 3 V. This means that the MOSFET will respond, just, to the 3.3V Logic 1 output level of the Mbed.

Characteristic	ZVN4206A
Maximum Drain-Source Voltage V_{DS}	60V
Maximum Gate-Source Threshold $V_{GS(th)}$	3V
Maximum Drain-Source Resistance when 'On'. $R_{DS(on)}$	1.5Ω
Maximum Continuous Drain current I_D	600mA
Maximum Power Dissipation	0.7W
Input Capacitance	100pF

Table 3.5: Characteristics of the ZVN4206A n-channel MOSFET

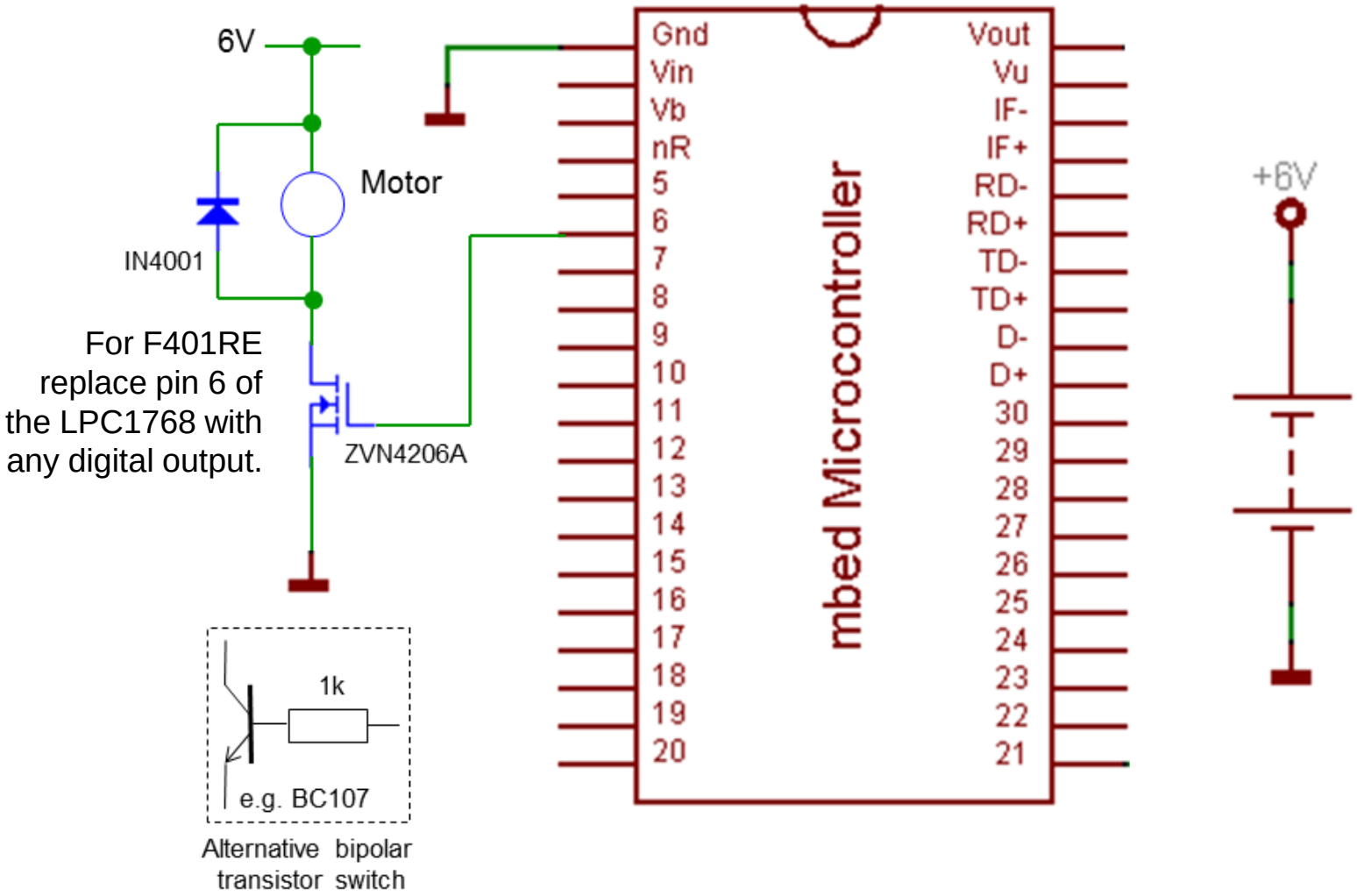
Switching Inductive Loads

A *freewheeling diode* is now needed because any inductance with current flowing in it stores energy in the magnetic field which surrounds it. When that current is interrupted, in this case by the transistor being switched off, the energy has to be returned to the circuit. This happens through the diode, which allows decaying current to continue to circulate.



Switching a DC motor with the Mbed Target

The circuit shown can be applied. Any digital output pin can in fact be used.



Chapter Review

- Logic signals, expressed mathematically as 0 or 1, are represented in digital electronic circuits as voltages. One range of voltages represents 0, another represents 1.
- Mbed-enabled development boards have many digital I/O pins, which can be configured either as input or output.
- LEDs can be driven directly from the microcontroller digital outputs. They are a useful means of displaying a logic value, and of contributing to a simple human interface.
- Electromechanical switches can be connected to provide logic values to digital inputs.
- Multi-coloured LEDs can be made by enclosing several individual LEDs of different colors in the same housing. The Mbed application board contains one of these.
- A range of simple opto-sensors can be configured to connect directly to development board pins.
- Where the microcontroller pin cannot provide enough power to drive an electrical load directly, interface circuits must be used. For simple on/off switching a transistor is often all that is needed.