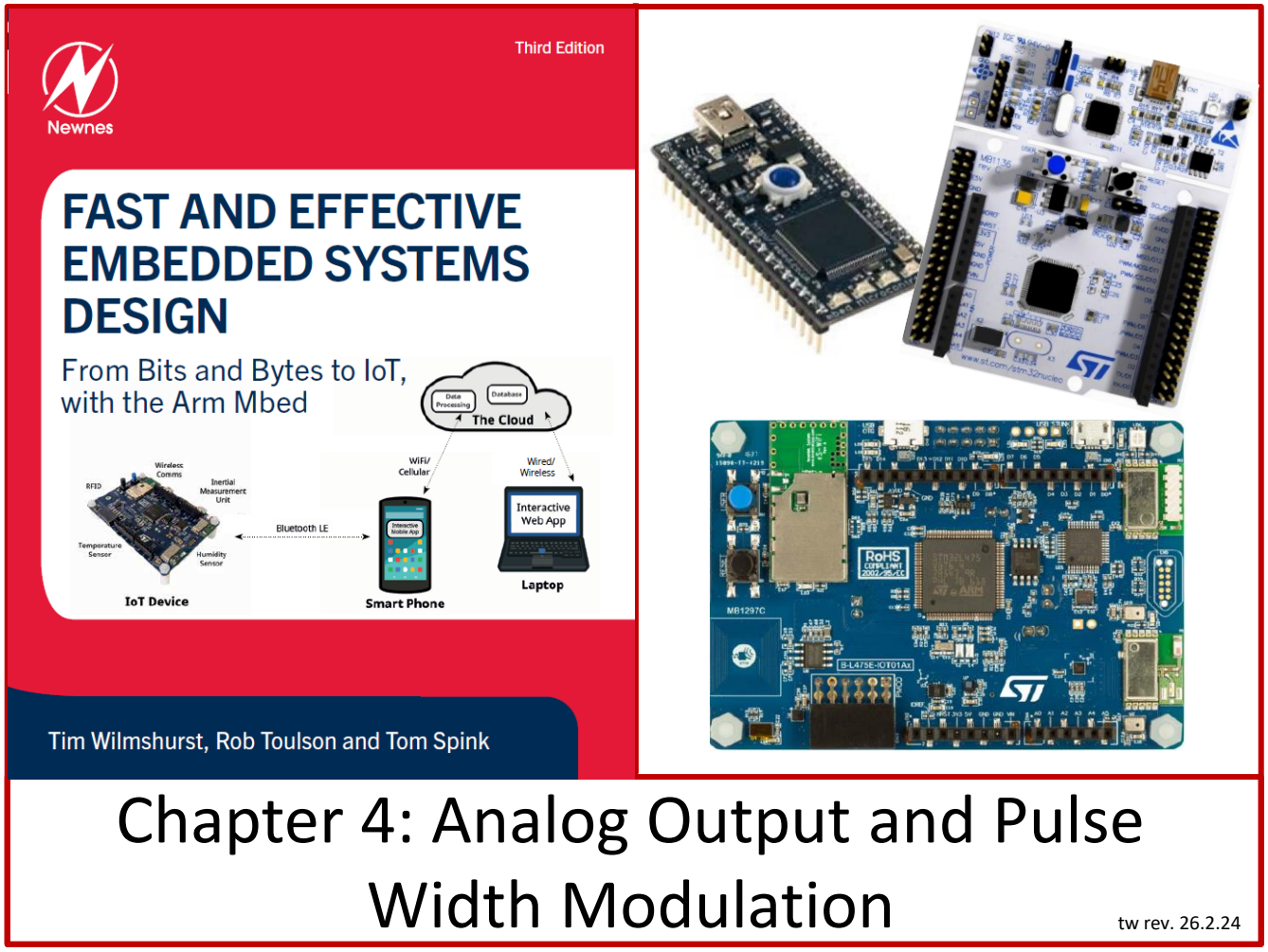


Chapter 4: Analog Output and Pulse Width Modulation

tw rev. 26.2.24

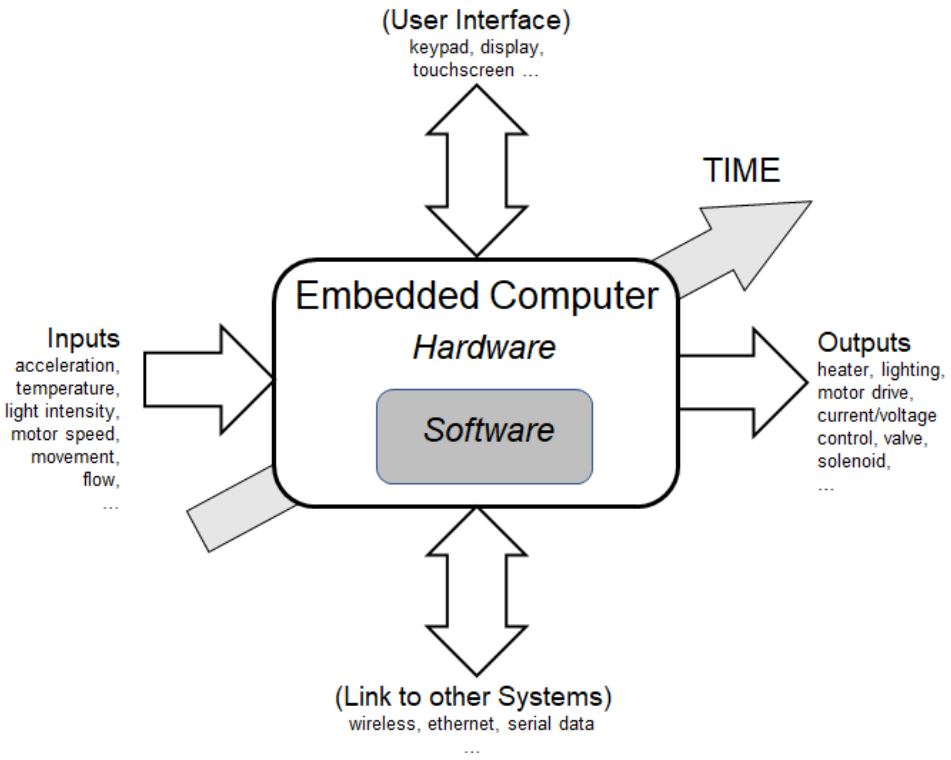
If you use or reference these slides or the associated textbook, please cite the original authors' work as follows:

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.

www.embedded-knowhow.co.uk

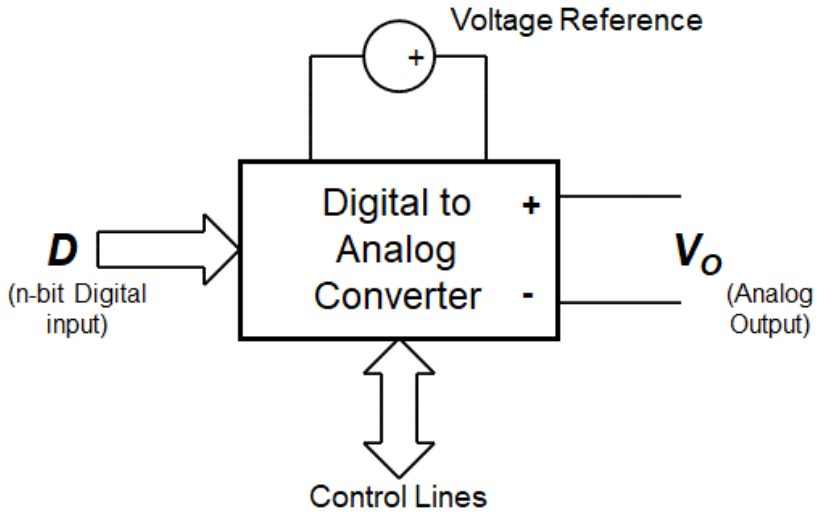
Introducing Data Conversion

Microcontrollers are digital devices; but they spend most of their time dealing with a world which is analog. To make sense of incoming analog signals, for example from a microphone or temperature sensor, they must be able to convert them into digital form. After processing the data, they may then need to convert digital data back to analog form, for example to drive a loudspeaker or dc motor. Here we consider *digital to analog conversion.*



The Digital-to-Analog Converter 1

A digital-to-analog converter (DAC) converts a binary input number into an analog output. The actual circuitry inside the DAC need not concern us. We can however represent the DAC as a block diagram, as shown. This has a digital input, represented by *D*, and an analog output, represented by V_o. To work, the DAC needs a *voltage reference* - a precise, stable and known voltage.

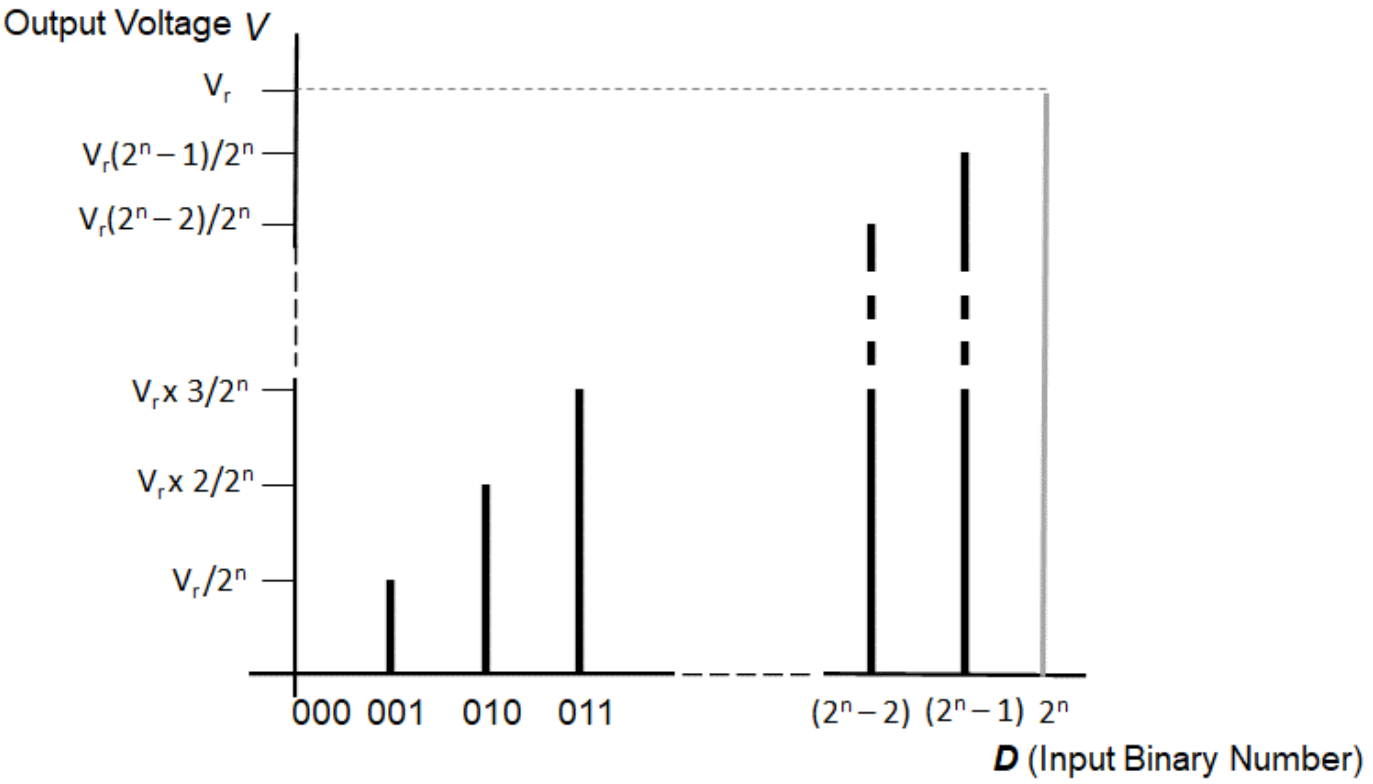


Most DACs have a simple relationship between their digital input and analog output, as shown here. V_r is the value of the voltage reference.

$$v_0 = \frac{D}{2^n} v_r$$

The Digital-to-Analog Converter 2

The Figure shows this equation represented graphically. For each input digital value, there is a corresponding analog output. The number of possible output values is given by 2^n, and the step size by $V_r / 2^n$; this is called the *resolution.* The maximum possible output value occurs when *D* reaches its maximum value, of $(2^n - 1)$.

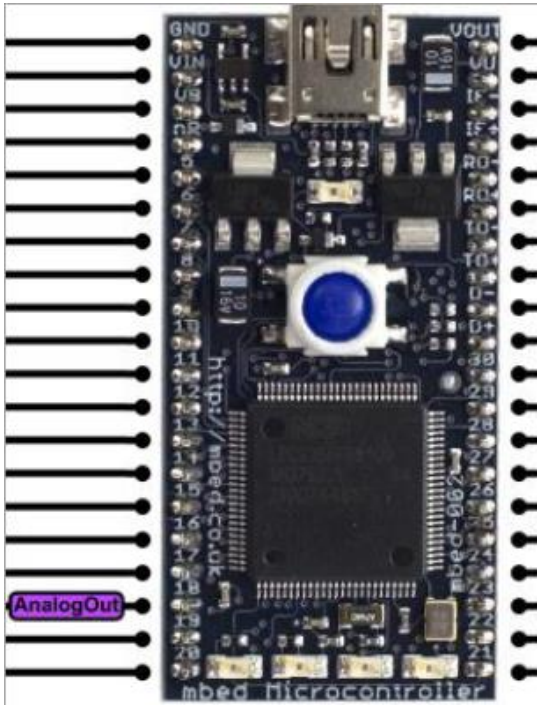


Mbed LPC1768 DAC Characteristics

The LPC1768 microcontroller has a 10-bit DAC; hence with 2^{10} steps in its output characteristic, i.e. 1024. It uses its own power supply voltage, 3.3 V, as voltage reference. The step size, or resolution, is therefore 3.3/1024, i.e. 3.22 mV.

The Mbed LPC17768 has a single DAC output on pin 18.

Note that the F401RE does not have a DAC, so the sections which follow can only relate to the Mbed LPC1768.



Analog Output on the Mbed LPC1768

The Table summarises the **AnalogOut** API.

Functions	Usage
Constructor	
`AnalogOut (PinName pin)`	Create an AnalogOut object connected to the specified pin.
Member Functions	
`void write(float value)`	Set the output voltage, specified as a percentage (float).
`void write_u16(unsigned short value)`	Set the output voltage, represented as an unsigned short in the range 0x0 to 0xFFFF, producing respectively 0.0 V and 3.3 V.
`float read()`	Return the current output voltage setting, measured as a percentage (float).
`Mbed-defined operator: =`	An operator shorthand for write().

C/C++ code feature

Check your understanding of keywords **float, short** and **unsigned.**

Setting Simple Analog Outputs

In this program we create an analog output labelled **Aout**, using **AnalogOut**. **Aout** can be set to any floating point number between 0.0 and 1.0, which then determines the analog output. This is between 0 V and 3.3 V; the floating point number is scaled to this.

```
/*Program Example 4.1: Three values of DAC are output in turn on
Pin 18. Read the output on a DVM.
Works only on Mbed LPC1768.
*/

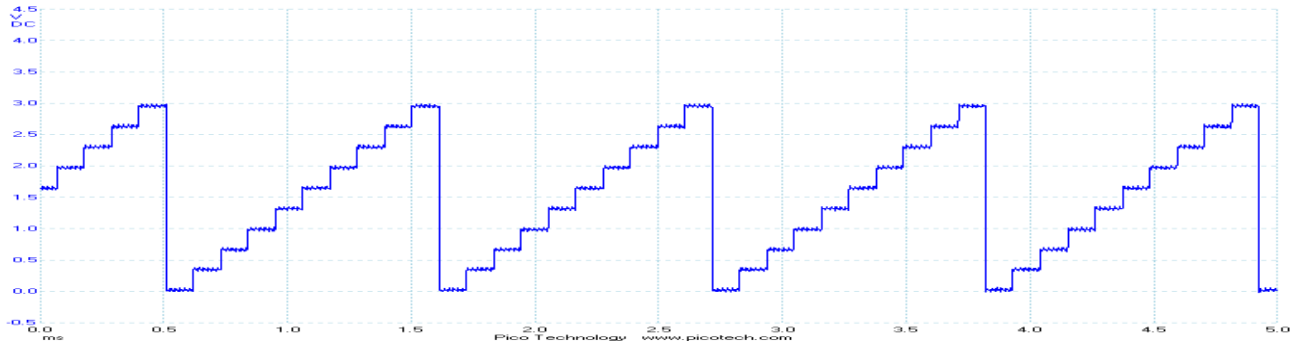
#include "mbed.h"
AnalogOut Aout(p18);   //create an analog output on pin 18
int main() {
  while(true) {
    Aout = 0.25;           // 0.25*3.3V = 0.825V
    ThisThread::sleep_for(2s);  //wait 2 seconds
    Aout = 0.5;            // 0.5*3.3V = 1.65V
    ThisThread::sleep_for(2s);
    Aout = 0.75;           // 0.75*3.3V = 2.475V
    ThisThread::sleep_for(2s);
  }
}
```

Generating a sawtooth waveform

```
/*Program Example 4.2: Saw tooth waveform on DAC output. View on
oscilloscope.
Only runs on LPC1768, as F401RE does not have a DAC.

*/
#include "mbed.h"
AnalogOut Aout(p18);
float i;

int main() {
  while(true){
    for (i = 0;i < 1;i = i+0.1){     // i is incremented in steps of 0.1
      Aout = i;
      ThisThread::sleep_for(1s);    // wait 1 millisecond
    }
  }
}
```



Generating a sine wave

C/C++ code feature

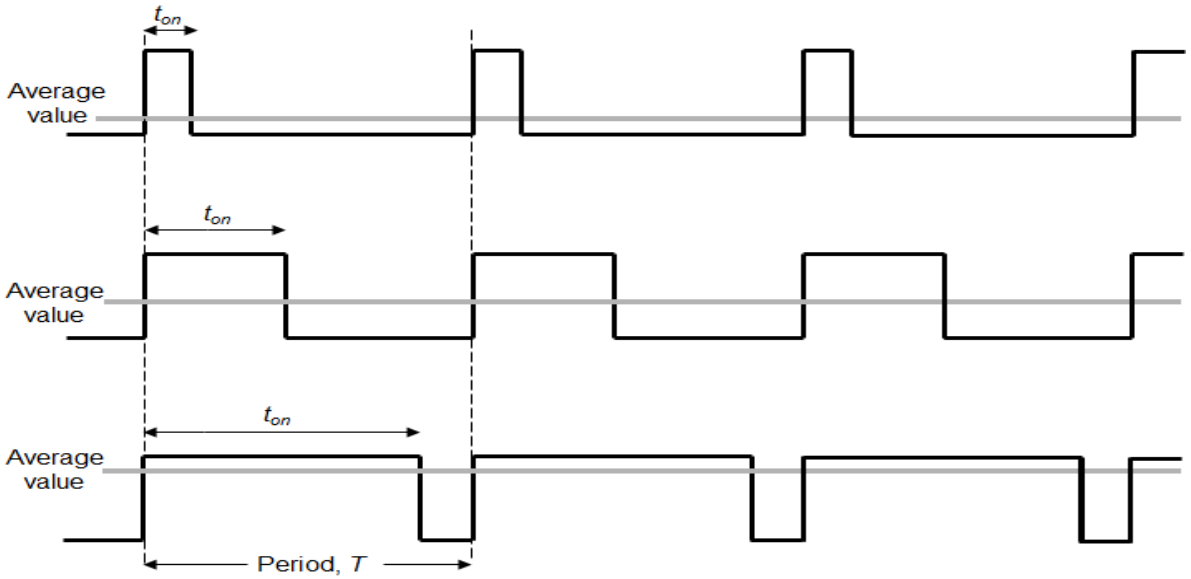
The **sin( )** function, part of the C standard library, is applied. To produce one cycle of the sine wave, a **for** loop is used to increment variable **i** from 0 to 2 in small steps, then multiplying that number by π to calculate the sine value. A further challenge is that the DAC cannot output negative values. Therefore, the amplitude is halved and an offset of 0.5 is added to the number sent to the DAC. The multiply operator, *, is used for the first time.

```
/*Program Example 4.3: Sine wave on DAC output. View on oscilloscope
Only runs on LPC1768, as F401RE does not have a DAC.
                                                                    */
#include "mbed.h"
AnalogOut Aout(p18);
float i;
int main() {
  while(true)  {
    for (i = 0;i < 2;i = i + 0.05) {
      // Compute the sine value, + halve the range
      Aout = 0.5 + (0.5 * sin(i * 3.14159));
      ThisThread::sleep_for(1ms);        // Controls the sine wave period
    }
  }
}
```

Another Form of Analog Output: Pulse Width Modulation

Pulse width modulation (PWM) can be an alternative to the DAC; PWM represents a neat and simple way of getting a rectangular digital waveform to control an analog variable, usually voltage or current. PWM is used in a variety of applications, ranging from telecommunications to robotic control.

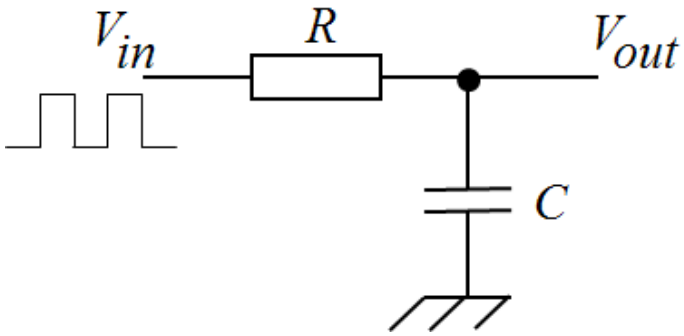
Whatever duty cycle a PWM stream has, there is an average value, as shown. By controlling the duty cycle, we control this average value. When using PWM, it is this average that we're usually interested in.



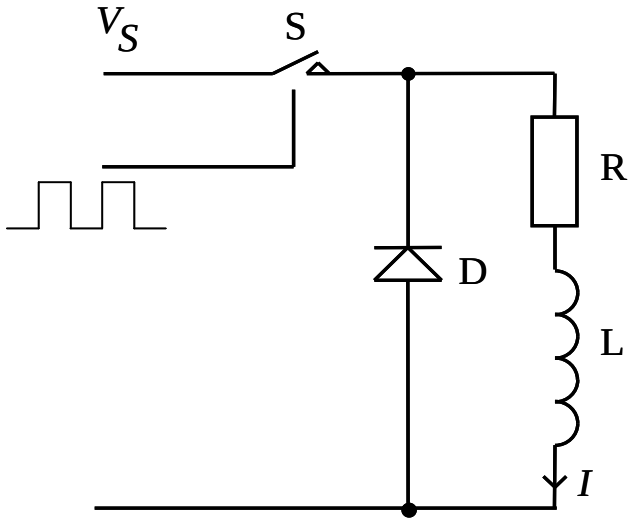
$$duty\ cycle = \frac{pulse\ on\ time}{pulse\ period} * 100\%$$

Simple Averaging Circuits

The average PWM value can be extracted from the PWM stream in a number of ways. We can use a low-pass filter, like the resistor-capacitor combination below left. In this case, and as long as PWM frequency and values of R and C are well-chosen, V_{out} becomes an analog output, with a bit of ripple, and the combination of PWM and filter acts just like a DAC. Alternatively, if we switch the current flowing in an inductive load, below right, then the inductance has an averaging effect on the current flowing through it. This is important, as the windings of any motor are inductive, so we can use this technique for motor control.



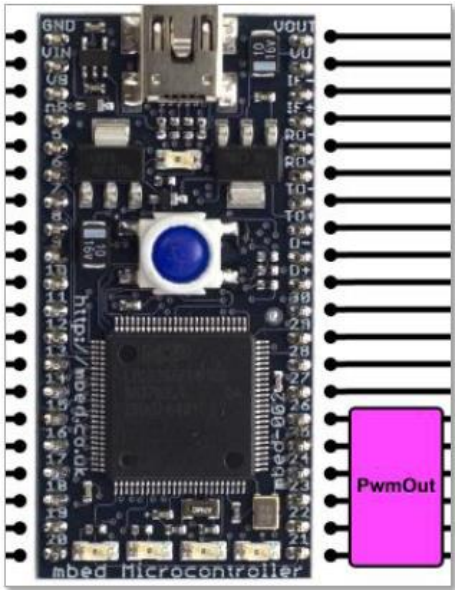
Resistor-capacitor low-pass filter



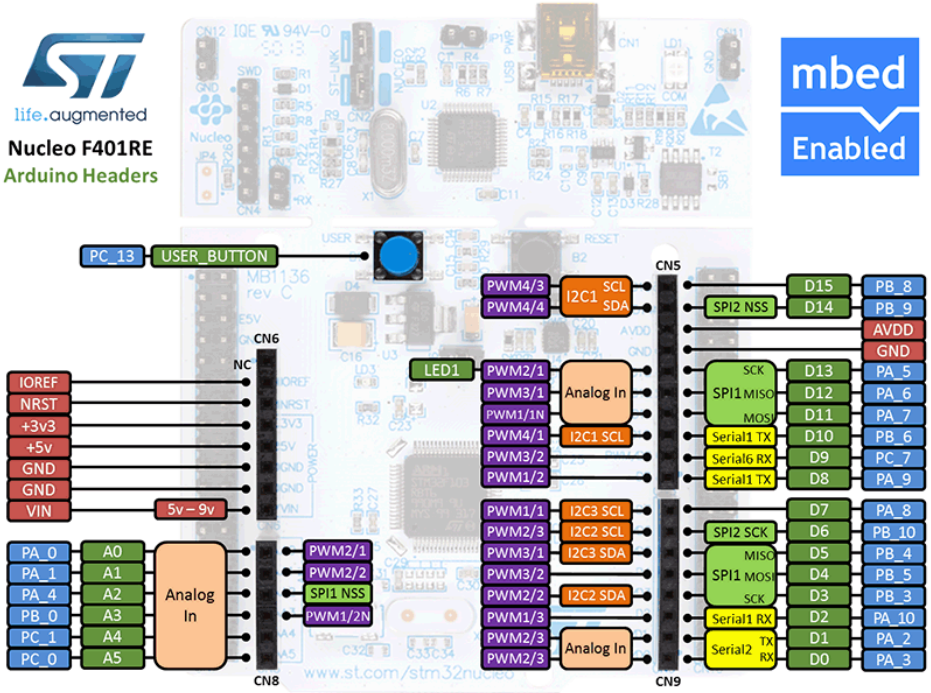
An inductive load

PWM in the Mbed Environment

the Mbed LPC1768 has six PWM outputs available, from pin 21 to 26 inclusive. These all share the same period/ frequency; if the period is changed for one, then it is changed for all.



The Nucleo F401RE appears to have 19 PWM outputs. They come from *four* sources. Hence the F401RE can generate PWM signals of up to four different frequencies. Thus the PWM sources on the F401RE are labelled for example PWM2/1, indicating output 1 of PWM source 2.



PWM in the Mbed Environment

The PWM API is summarised here.

Functions	Usage
Constructor	
`PwmOut(PinName pin)`	Create a PwmOut object connected to the specified pin
Member Functions	
`void write(float value)`	Set the output duty-cycle, in range 0.0 (0% on) to 1.0 (100% on).
`float read()`	Return the current output duty-cycle setting, measured as a normalised float (0.0 – 1.0)
`void period(float seconds)`	Set the PWM period, specified in seconds (float), keeping the duty cycle the same.
`void period_ms(int ms)`	Set the PWM period, specified in milliseconds (int), keeping the duty cycle the same.
`void period_us(int us)`	Set the PWM period, specified in microseconds (int), keeping the duty cycle the same.
`void pulsewidth(float seconds)`	Set the PWM pulse width, specified in seconds (float), keeping the period the same.
`void pulsewidth_ms(int ms)`	Set the PWM pulse width, specified in milliseconds (int), keeping the period the same.
`void pulsewidth_us(int us)`	Set the PWM pulse width, specified microseconds (int), keeping the period the same.
`mbed-defined operator: =`	An operator shorthand for write()

A Trial PWM Output

This program generates a 100 Hz pulse with 50% duty cycle, i.e. a perfect square wave, appearing on pin 21 (LPC1768) or D2 (F401RE).

```
/*Sets PWM source to fixed frequency and duty cycle. Observe output on
oscilloscope.
Runs on LPC1768 or F401RE, using pins shown.                          */

#include "mbed.h"

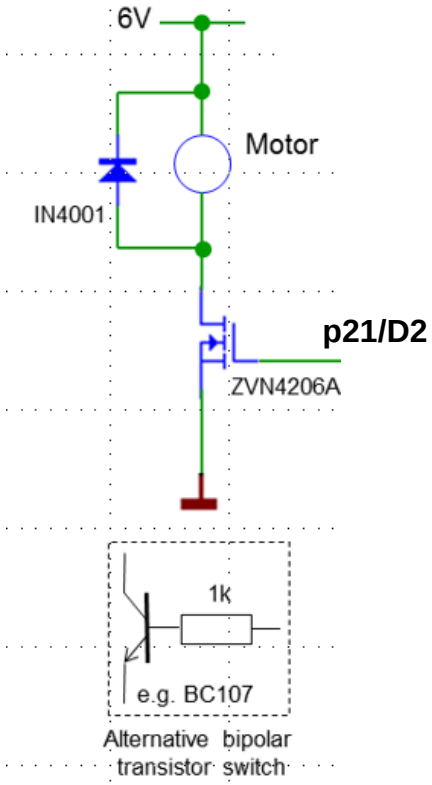
//create a PWM output called PWM1
PwmOut PWM1(p21);               //Use D2 for F401RE
int main() {
  PWM1.period(0.010);           // set PWM period to 10 ms
  PWM1 = 0.5;                   // set duty cycle to 50%
}
```

Speed Control of a Small Motor

This program ramps the PWM up from a duty cycle of 0% to 100%. Motor speed should ramp up accordingly.

```
/*Program Example 4.5: PWM control to DC motor is repeatedly ramped
Runs on LPC1768 or F401RE, using pins shown.
*/

#include "mbed.h"
PwmOut PWM1(p21);  //Use D2 for F401RE
float i;
int main() {
  PWM1.period(0.010);           //set PWM period to 10 ms
  while(true) {
    for (i = 0;i < 1;i = i + 0.01) {
      PWM1 = i;                 // update PWM duty cycle
      ThisThread::sleep_for(1ms);
      }
  }
}
```



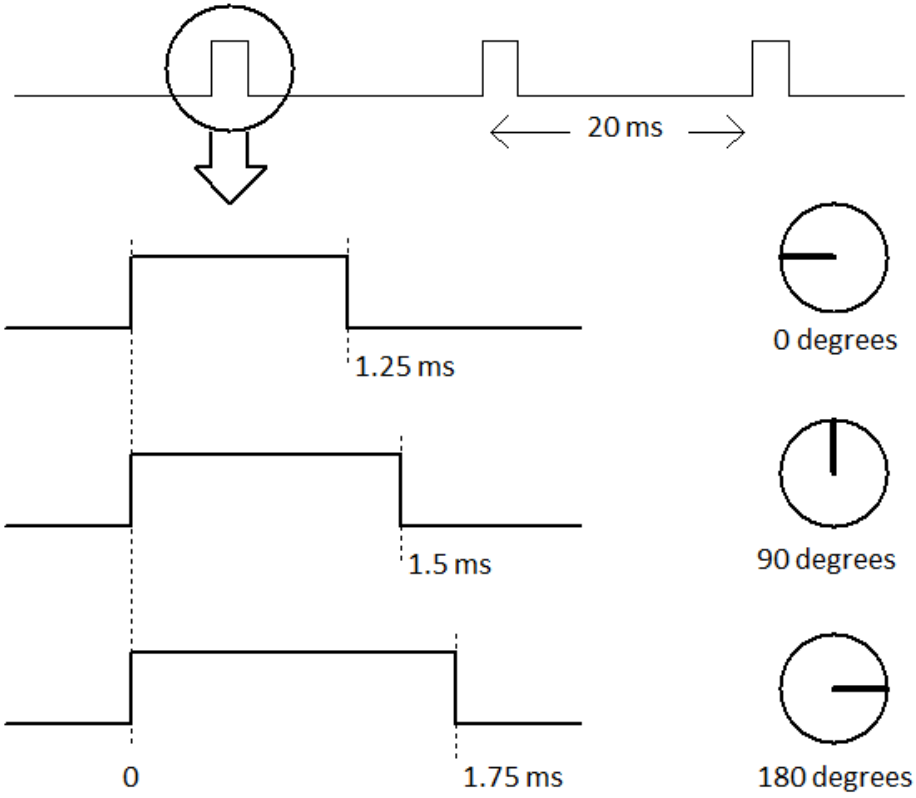
Generating PWM in Software

It is useful to realise hardware PWM sources aren't essential for creating PWM; it can be done just with a digital output, and some timing, as shown here.

```
/*Program Example 4.6: Software generated PWM. 2 PWM values generated in turn,
with full on and off included for comparison.
Runs on LPC1768 or F401RE, using pins shown.
*/
#include "mbed.h"
DigitalOut motor(p6);  //use any digital output for F401RE
int i;
int main() {
  while(true) {
    motor = 0;                      //motor switched off for 5 secs
    ThisThread::sleep_for(5s);
    for (i = 0; i < 5000;i = i + 1) {  //5000 PWM cycles, low duty cycle
      motor = 1;
      wait_us(400);                 //output high for 400us
      motor = 0;
      wait_us(600);                 //output low for 600us
    }
    for (i = 0; i < 5000;i = i + 1){   //5000 PWM cycles, high duty cycle
      motor = 1;
      wait_us(800);                 //output high for 800us
      motor = 0;
      wait_us(200);                 //output low for 200us
    }
    motor = 1;                      //motor switched fully on for 5 secs
    ThisThread::sleep_for(5s);
  }  //end of while loop
}
```

Servo Control

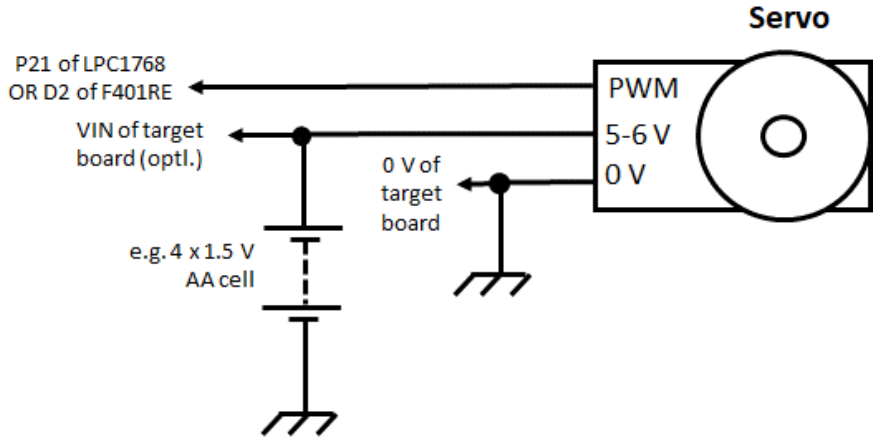
A servo is a small rotary position control device, often used in radio-controlled cars and aircraft to control angular position of variables such as steering, elevators and rudders. Servos are now popular in certain robotic applications. The servo shaft can be positioned to specific angular positions by sending the servo a PWM signal, as shown.

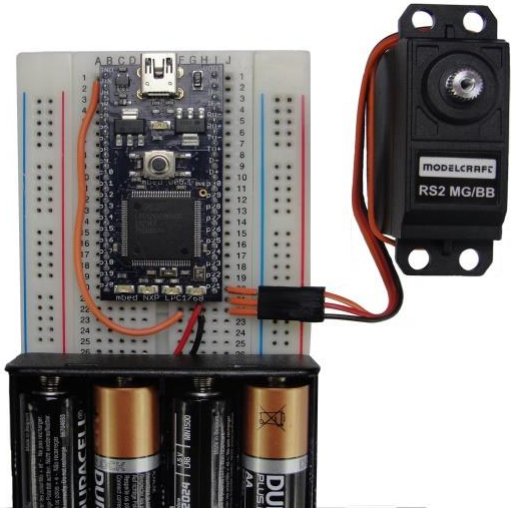


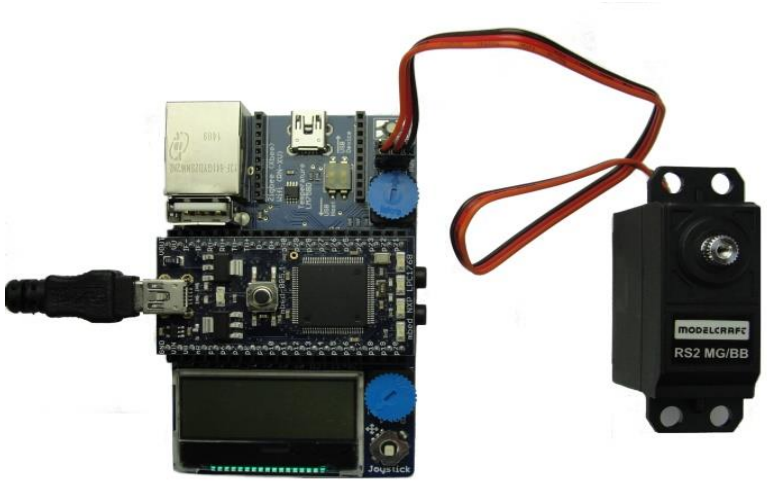


Servo Control with the mbed

A servo can readily be connected to an Mbed target. It requires a higher current than the USB standard can provide, and so must be powered using an external supply, e.g. 4xAA (6 V) battery pack.
The app board has PWM output connectors, so both breadboard or app board builds are possible.







Producing Audio Output

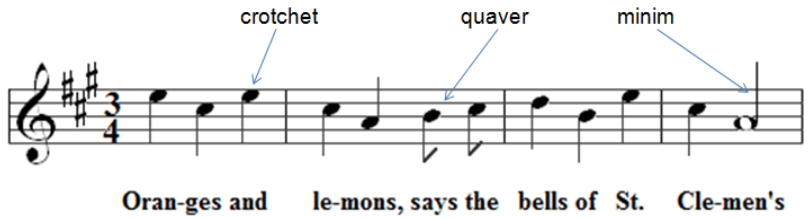
We can use PWM as a signal generator. Here it sounds a piezo transducer, playing the start of an old London folk song.

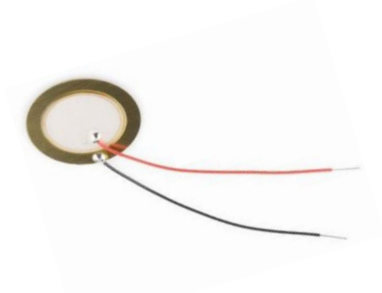
C/C++ code feature

The program uses two C arrays, one for frequency data, the other for beat length. There are 12 values in each, for each of the 12 notes in the tune.

```
/*Program Example 4.7: Plays the tune "Oranges and Lemons" on a piezo buzzer,
using PWM.
Runs on LPC1768 or F401RE, using pins shown.                    */

#include "mbed.h"
PwmOut buzzer(p26);  //Use D2, or PWM output of choice, for F401RE
                                                        //frequency array
float frequency[]={659,554,659,554,440,494,554,587,494,659,554,440};
float beat[]={1,1,1,1,1,0.5,0.5,1,1,1,1,2};              //beat array
int main() {
  while (true) {
    for (int i = 0;i <= 11;i++) {
      buzzer.period(1/(2 * frequency[i]));          // set PWM period
      buzzer = 0.5;                                 // set duty cycle
      wait_us(400000 * beat[i]);                      // hold for beat period
    }
  }
}
```





Chapter review

- A digital–to-analog converter (DAC) converts an input binary number to an output analog voltage, which is proportional to that input number.
- DACs are widely used to create continuously varying voltages, for example to generate analog waveforms.
- The Mbed LPC1768 has a single DAC, and associated set of library functions. The Nucleo F401RE does not have a DAC.
- Pulse Width Modulation (PWM) provides a way of controlling certain analog quantities, by varying the pulse width of a fixed frequency rectangular waveform.
- PWM is widely used for controlling flow of electrical power, for example led brightness or motor control.
- The Mbed LPC1768 has six possible PWM outputs. They can all be individually controlled, but must all share the same frequency. The Nucleo F401RE shows 19 possible PWM outputs, these are from four different sources, so four different frequencies are possible.