***Rapid Embedded Systems***

***Design and Programming Course***

**LAB 3**

**Analog I/O**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc35526552)

[1.1 Lab overview 1](#_Toc35526553)

[2 Requirements 1](#_Toc35526554)

[2.1 Software and Hardware 1](#_Toc35526555)

[3 Hardware Setup 2](#_Toc35526556)

[3.1 Pin Layout 2](#_Toc35526557)

[4 Technical background 3](#_Toc35526558)

[4.1 PwmOut Interface 3](#_Toc35526559)

[4.2 Wave Generation 4](#_Toc35526560)

[4.3 Analog I/O Interfaces 4](#_Toc35526561)

[5 Generating audio waves using PWM 5](#_Toc35526562)

[5.1 Audio waves 5](#_Toc35526563)

[5.2 Your application code 5](#_Toc35526564)

[6 Appendix 6](#_Toc35526565)

[7 Additional references 6](#_Toc35526566)

# Introduction

## Lab overview

In this lab, we will implement an audio wave generator using PWM. The volume and the pitch can be tuned using potentiometers.

By the end of this lab you will get some insight and practical experience with the Mbed API for Analog I/O and PWM.

# Requirements

## Software and Hardware

In this lab, we will be using the following hardware and software:

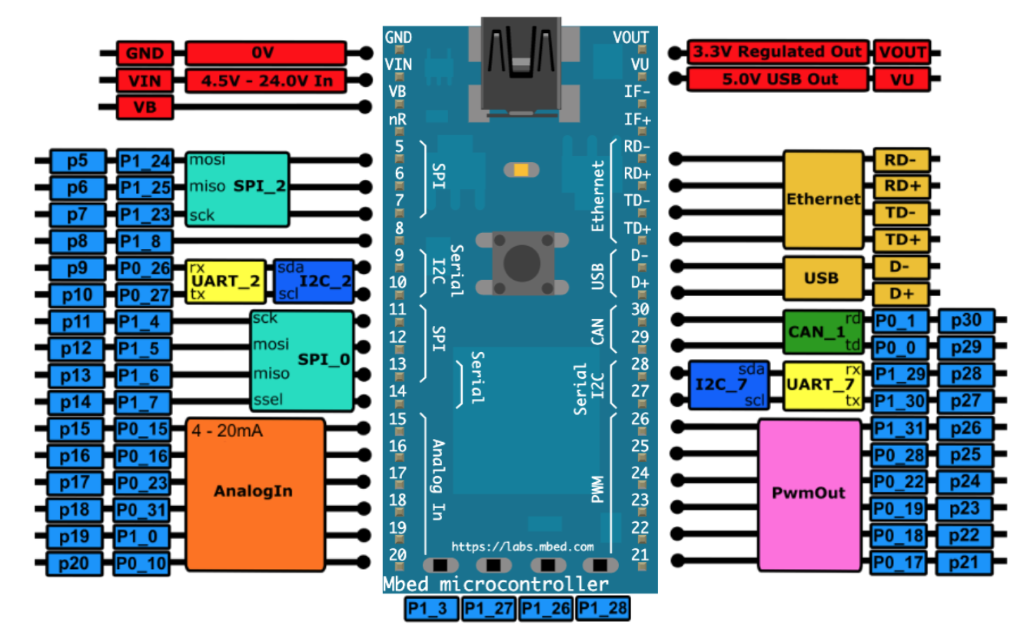
* **Visual Studio Code,** or any other text editor.
* **Mbed Simulator**, an offline simulator of an mbed microcontroller and hardware components.

The code skeleton, which includes some support for implementing the task in 5.2 should be found in the same folder as this manual.

# Hardware Setup

## Pin Layout

In this experiment, we are going to use the Mbed Simulator target, which pin descriptions can be found below:



*Figure 1: The Mbed Simulator board pin descriptions*

Create and add the PWM Speaker the component of the simulator as explained in the appendix.

In the Simconfig.json file, connect the thermistors and the PWM Speaker to their respective pin as defined in the table below.

Pin on the board as defined in the table below.

|  |  |
| --- | --- |
| **Pin** | **Pin name in Mbed API** |
| Potentiometer 1 | p15 |
| Potentiometer 2 | p16 |
| PWM speaker | p21 |

# Technical background

## PwmOut Interface

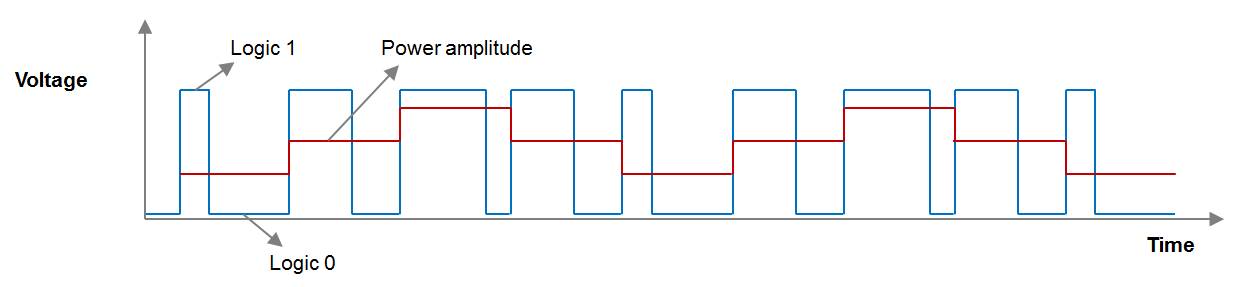
Pulse width modulation (PWM) is a simple method of using a rectangular digital waveform to create an analog output. PWM uses the width of the pulse to represent an amplitude.

The period of the wave is usually kept constant, and the pulse width, or ON time is varied.

The duty cycle is the proportion of time that the pulse is ON or HIGH, and is expressed as a percentage:

Duty cycle = 100% \* (pulse ON time)/(pulse period)

Whatever duty cycle a PWM stream has, there is an average value. If the ON time is small, the average value is low; if the ON time is large, the average value is high. Therefore, by controlling the duty cycle, we control the average output value (represented as the red line below).



We can use the PwmOut Interface to control the frequency and duty cycle of a PWM signal. Here are some examples showing how to use the PWM:

PwmOut led(D5); //define the PWM output

led = 0.5; //set the duty cycle of PWM output to 50%

led.period(0.02) //set the output frequency to 50Hz ;

PwmOut Interface has a set of member functions which can be found in figure 3:

|  |  |
| --- | --- |
| **Function name** | **Description** |
| **Void write (float value)** | Set the output duty-cycle, specified as a percentage (float) |
| **float read ()** | Return the current output duty-cycle setting, measured as a percentage (float) |
| **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 milli-seconds (int), keeping the duty cycle the same |
| **void period\_us (int us)** | Set the PWM period, specified in micro-seconds (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 |

Figure 3: Member function of PwmOut API

Note that not all the Digital pins can be used as PWM outputs.

## Wave Generation

We can use for loops to generate simple waveforms without using lookup tables. For example, for a saw-tooth wave we could use:

for(i=0; i<1; i+=0.05)

out = i;

or, for a triangle wave use two for loops:

for(i=0; i<1; i+=0.05)

out = i;

for(i=1; i>0; i-=0.025)

out = i;

## Analog I/O Interfaces

The AnalogIn interface is used to read an external voltage applied to an analog input pin. AnalogIn() reads the voltage as a fraction of the system voltage. The value is a floating point from 0.0(VSS) to 1.0(VCC). For example, we can test the voltage on the initialized analog pin and compare it to a threshold voltage.

AnalogIn ain(A0);

DigitalOut dout(LED1);

int main(void)

{

while (1) {

if(ain > 0.3f) {

dout = 1;

} else {

dout = 0;

}

wait(0.2f);

}

}

AnalogIn Interface has a set of member functions which can be found in figure 4:

|  |  |
| --- | --- |
| **Function name** | **Description** |
| **float read ()** | Read the input voltage, represented as a float in the range [0.0, 1.0] |
| **unsigned short read\_u16 ()** | Read the input voltage, represented as an unsigned short in the range [0x0, 0xFFFF] |
| **operator float ()** | An operator shorthand for read () |

Similarly, we use AnalogOut interface to set the output voltage of an analog output pin specified as a percentage or as an unsigned short. It can be used to generate waveforms, for example, for outputting a saw-tooth waveform:

AnalogOut aout(A5);

DigitalOut dout(LED1);

int main(void)

{

while (1) {

for (float i = 0.0f; i < 1.0f; i += 0.1f) {

aout = i;

wait(1.0f);

}

}

}

# Generating audio waves using PWM

In this lab, you will have to use the PWM output to generate electrical waves which can be turned into sound by the on-board speaker. Be aware that the frequency of the wave must be in a specific range so that humans can hear it.

## Audio waves

A sound is essentially an air wave. The amplitude and the frequency of the wave decide the volume and the pitch of the sound respectively. The speaker (or headphone) inputs electrical signals (voltages) and use them to turn a coil into an electromagnet, which can either attract or repel the magnet that moves back and forth. The motion of the magnet will further push and pull a diaphragm and create air waves (just like a drum).

## Your application code

In this lab exercise you will need to:

* Define analog input, PWM and serial output ports
  + Two analog inputs for two potentiometers
  + One PWM output for the speaker
* Generate waves to the PWM output
  + Generate a saw-tooth wave (design the range to be from 320Hz to 8kHz)
  + Use potentiometer 1 to tune the pitch and the potentiometer 2 adjust the volume.
* Test the code, use the LEDs to track both input voltages.

What changes when you adjust the amount by which variable *i* is incremented/decremented?

# Appendix

We are going to create a new component and add it the Mbed Simulator as it was explained in the Getting Started Lab. Before adding the component, please check that you have made the changes explained in section “5.3.3 Modify mbed simulator-hal” of the Getting Started document.

1. Find a picture of a speaker and save it as *viewer/img* as *pwm\_speaker.png*.
2. Add in *viewer/js-ui/viewer.js*:

{

component: 'PwmSpeaker',

name: 'Pwm Speaker',

pins: [ { name: 'Speaker', value: [ 'p21', 'p22', 'p23',

'p24', 'p25', 'p26' ] } ]

},

inside var components = [ … ]

1. Add the *pwm\_speaker.js*provided file to *viewer/js-ui/components*.
2. Add in viewer/viewer.html:

<script ="text/javascript" src="/js-ui/components/pwm\_speaker.js">.

</script>

under   <!-- UI handling -->

# Additional references

**Documentation for AnalogIn API**

<https://os.mbed.com/docs/mbed-os/v5.13/apis/analogin.html>

**Documentation for PwmOut API**

<https://os.mbed.com/docs/mbed-os/v5.13/apis/pwmout.html>

**Audio frequencies**

<http://en.wikipedia.org/wiki/Audio_frequency>