Embedded Systems Essentials with Arm: Getting Started

Module 6

SV2 (6): Module 6 Lab Project: Part 1

Now let's start looking at the exercise. For this lab, you will be using the Mbed Simulator, a simulation of an Mbed microcontroller and hardware peripherals/components.

Here you can see a diagram mapping out the design of the simulated microcontroller. The pin descriptions can also be seen.

As previously mentioned, in this lab we will create an audio player that uses a timer and PWM.

Begin by opening up the simulator and loading the Module 6 – PWM Skeleton Lab demo.

We will now take a look at the code required for completing this project.

In order to play music, it is easier to define a list of notes that may be used to play it. This includes the music notes, meaning the frequency of the sound, e.g. do, re, mi, fa, so, la, or ti, and the beat length, or the time that the current note lasts for, such as whole note, half note. To do that, you need to adjust the period of the PWM so that it can produce a correct frequency for each music note, and the period of play to produce the desired beat.

After defining the music note and beat length, you can then make a note array which will produce a simple piece of music, for example, the following sheet gives the notes and beats of "Jingle bells".

The skeleton code provides this for you: the frequency of basic music notes such as the natural 4th octave, as well as a silent note called "No", are defined in the highlighted area.

In this next highlighted area, we define the length of each note. As we want to be able to modify the speed of playback, we define them as a portion of the whole note. We define a half, quarter, eighth and a sixteenth of a note.

These component parts are then formed into a musical piece for you. This is in the form of two arrays, one containing a sequence of notes, and another containing their lengths. Notice we add in some silence between each note.

To build upon the skeleton code, the first thing to do is define the analog inputs, using "AnalogIn", which are buttons, and then the PWM outputs, using "PwmOut", which are a speaker and RGB LEDs. This can be seen in the highlighted area. The idea behind this is so that we can reflect the melody on the LEDs by modifying the intensity of each colour accordingly to the song.

Following this, we have to set up a timer to periodically generate an interrupt after a specific time. For our project it would be possible to initialise a timer and monitor the counting value to determine when a note has to be played. However, this is a very inefficient method. Instead we can use Ticker. A ticker is used to call a function at a recurring interval as described in the theory review. A ticker object is initialized with a name. Then we use "attach" to specify a function to be called and at a specified interval of time. Once attached, the function with the specified name will be called repeatedly at each interval of time. For now we just define the name of the ticker object.

Then before writing the main body of code, some variables need to be declared that will be used later on. The static 'int "k"' will be used to control at which point of the song we are. The other two float variables will be used to adjust the volume and speed of the melody.

In the main function, you will need to initialize the previously defined ticker. We use "attach" to specify a function to be called, in this case "timer_ISR", and at a specified interval of time. Once attached, the function with the specified name will be called repeatedly at each interval of time. The main function will also loop and update the volume and speed of the song – controlled by the potentiometers. This will utilize the read function.