# **Embedded Systems Essentials with Arm: Get Practical with Hardware**

## Module 4

## SV1: Lab 4 – Audio Player Project

In this lab, we will apply knowledge gained throughout the course to build a simple audio player, with user control. While you are led towards a proposed design, you shouldn’t feel constrained by this. Feel free to change or adapt hardware or software designs, in order to explore your own interests creatively.

Start by connecting up the circuit as shown in this diagram. If you have your build from the previous lab, you can adapt it fairly easily. You can simply remove the temperature sensor circuit, and add the potentiometer circuit, LEDs and piezoelectric sounder. The connection details for the Nucleo board are also displayed here for convenience.

Here we have a complete build. You can see that the one breadboard has become quite crowded with this circuit, with the buttons and LEDs being partially obscured by the wiring. You may wish to move to two breadboards, a larger breadboard, or try a different layout.

We have linked the piezo sounder to D9 via a yellow wire and placed it next to the LCD. It lets out a very weak sound if it’s just hanging freely, but this is greatly increased when it’s held or mounted, even with Blu Tack or similar, onto a hard surface. The sound can become quite irritating, so be ready to detach or disconnect it while you work! The green LED at the top centre of the image is connected directly across 3.3 V and ground as a power indicator. This was not shown in the circuit diagram shown earlier.

As we’re planning to develop a moderately complex program, it’s worth standing back to consider what the program needs to do, and how it should be structured. Flow or state diagrams, or other diagramming tools can help with this.

The informal state diagram shown here has been used as a simplified structure of the program development described. The diagram shows four main states that the Audio Player can move through. Between each state we can see the action that causes the transition. The two main states are Ready and Playing, each indicated by an LED. Within the program, each state is indicated by a Boolean variable, as shown in the diagram. An RTOS-based approach is adopted to implement this state diagram.

When writing an RTOS-based program, an early requirement is to define the tasks or threads. Looking at the diagram, and the intended functionality outlined in the Introduction, the following tasks were chosen, with the Play button triggering an interrupt:

1. Updating the LCD with the Welcome message.
2. Testing the select buttons and configuring the new song.
3. Playing the song.

Other functions of the Audio Player are merged into these threads. The division used is not definitive, and threads could be defined in different ways.

Now let's look at the program for the basic audio player. Here’s the code from the file “main”. This is accompanied by three other files, whose listings are given in the Appendices. The complete file list is therefore:

* Main dot c p p
* Tunes dot h
* 4bit\_LCD dot h
* 4bit\_LCD dot c p p

The last two of these are simply the LCD driver functions that we have been using in the past few labs, now transferred to separate files. The Tunes.h file contains data for 8 songs, taken from 8 different countries. As we can see, each song is formatted into a C or C++ structure, with information on name, length, note frequency and note duration. The song is played by cycling through each value in turn of the frequency and duration arrays.

Focusing on the “main” file; We start by declaring the relevant variables and objects we will need for the project. This includes a “P W M out” object for the piezo buzzer, a “bus in” object for the select buttons, “digital out” objects for the LEDs, an “analog in” object for the potentiometer, and an “interrupt in” object for the play button. After this, we create a mutex object that will help us protect certain blocks of code in the program from concurrency issues.

We then define a number of flags that will indicate the current state of the player. There are flags that indicate the “welcome”, “waiting”, “primed”, and “playing” states.

Then we declare some of the functions that will be used later on in the different threads.

The primary action of the program lies in these three threads.

Thread 1, LCD\_cont(), displays the welcome message when enabled. That is, when the welcome bool variable is set high. It ends by clearing welcome, and switching on the ready LED.

Thread 2, Tune\_select(), is there to make the choice of song, if the Play button has been pressed and no song is currently playing. Selection is made with a switch statement, testing the three bits from the push buttons, which make up the input bus labelled choose. It then displays the chosen song on the LCD, implementing a mutex to ensure safe access.

Thread 3, Play\_Tune(), does exactly that! PWM operation is suspended at the end, as otherwise it may hold the last note indefinitely. Consequently, the PWM is resumed at the beginning of the thread. The song pointer has been preloaded with the address of the song to be played, and the frequency is read and transferred to the PWM. A simple volume

control is implemented by reading the potentiometer value and using it to set the PWM duty cycle. The thread then sleeps for the duration indicated by the chosen beat element. This for() loop iterates for the number of times indicated by the length element in the song structure, moving in turn through the frequency and beat pairs. When the song is completed the playing LED is switched off, and the welcome variable set to I, indicating the welcome message needs to be displayed and that the Audio Player will be ready to play another song.

The main() function is very simple, consisting essentially of the launch of the Interrupt and the three threads, followed by a while loop awaiting calls to the threads, and/or the interrupt.

Create this program in Mbed Studio. Each of the program files - provided with this lab should be created as separate files and copied into the same program folder under their given names. The result should look like this.

Build and download the program to your previously constructed hardware.

If you encounter issues, check the troubleshooting section of the lab document for potential solutions.

Now that you have completed this lab, you should have considerably enhanced your level of confidence in designing, building and commissioning embedded systems of moderate challenge and complexity.

If you can take the basic design described and find ways of enhancing both the hardware and software, you will have succeeded in climbing the steepest part of the embedded learning curve, and you can look forward to considerable further progress.