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

**LAB 4**

**Audio Player Mini-Project**

Contents

[1 Introduction 1](#_Toc67645293)

[2 Resources 1](#_Toc67645294)

[3 Configuring the Circuit 2](#_Toc67645295)

[4 Developing the Program 4](#_Toc67645296)

[4.1 Planning the Program Structure 4](#_Toc67645297)

[4.2 Determining the Tasks 4](#_Toc67645298)

[4.3 The Program 5](#_Toc67645299)

[4.4 Troubleshooting 9](#_Toc67645300)

[5 Taking Things Further 9](#_Toc67645301)

[6 Conclusion 10](#_Toc67645302)

[7 References 10](#_Toc67645303)

[Appendix 1: Tunes.h 11](#_Toc67645304)

[Appendix 2: 4bit\_LCD.h 12](#_Toc67645305)

[Appendix 3: 4bit\_LCD.cpp 12](#_Toc67645306)

# Introduction

This Laboratory (Lab) aims to guide you through the development of a project, applying and extending knowledge gained in this and previous modules, including of course use of the Mbed Real Time Operating System (RTOS).

The project aims to design and build a simple audio player, with user control. While you are led towards a proposed basic design, you should not feel committed to this. Be ready if you wish to change or adapt hardware or software designs, in order to explore creatively your own interests.

The basic Audio Player will be able to:

- play 8 different songs selected by three push buttons;

- display on the LCD the name of the song that is being played;

- show the status using LEDs;

- display instructions for use;

- adjust the volume of the song.

# Resources

In this lab you can use either the Mbed Studio, or the on-line compiler (or both). Both were introduced in Lab 0 and used in previous labs. The hardware elements needed are listed in Table 1. Items which are in grey were used in Labs 0 to 3, and continue to be used. Those in black are new for this lab.

|  |  |
| --- | --- |
| **Item** | **Qty.** |
| STM32F401 Nucleo-64 Development Board | 1 |
| Bread Board | 2 |
| Jumper Wires (kit) | 1 |
| LED with internal current-limiting resistor | 3 |
| 74HC595N Shift Register | 1 |
| Newhaven LCD. NHD-0420H1Z-FSW-GBW-33V3 | 1 |
| 10kΩ potentiometer | 1 |
| 100 Ω resistor | 4 |
| Push-button switches | 4 |
| 10 kΩ resistors | 4 |
| Piezo Transducer (Speaker) | 1 |

*Table 1: List of Required Parts*

# Configuring the Circuit

Connect up the circuit of Figure 1. If you have your build from Lab 3, you can adapt it fairly easily, by removing the temperature sensor circuit, and adding the potentiometer circuit, LEDs and piezoelectric sounder. The connection details of the Nucleo board are included in Figure 2 for convenience.

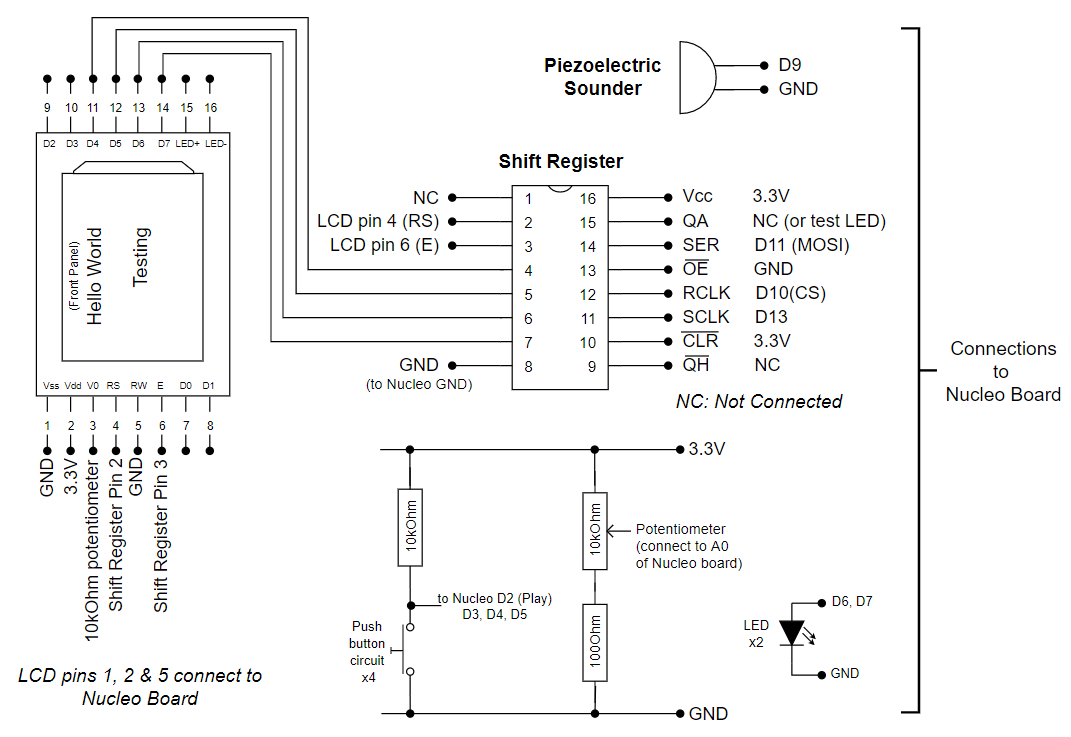


Figure 1: Circuit Layout

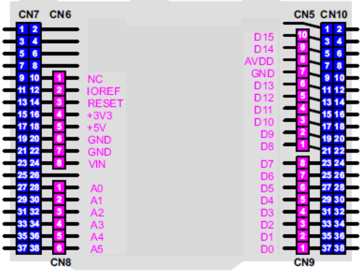


Figure 2: Connections on Nucleo Board, Arduino Connectors Only

A complete example build is shown in Figure 3. The piezo sounder (linked to D9 via a yellow wire) is here placed beside next to the four buttons; you may of course choose a different location. The sound can become quite irritating quite quickly, so be ready to detach or disconnect it when you’re continuing your development! The green LED at the top centre of the image is connected directly across 3.3 V and ground as a power indicator, it is not shown in the circuit of Figure 1.

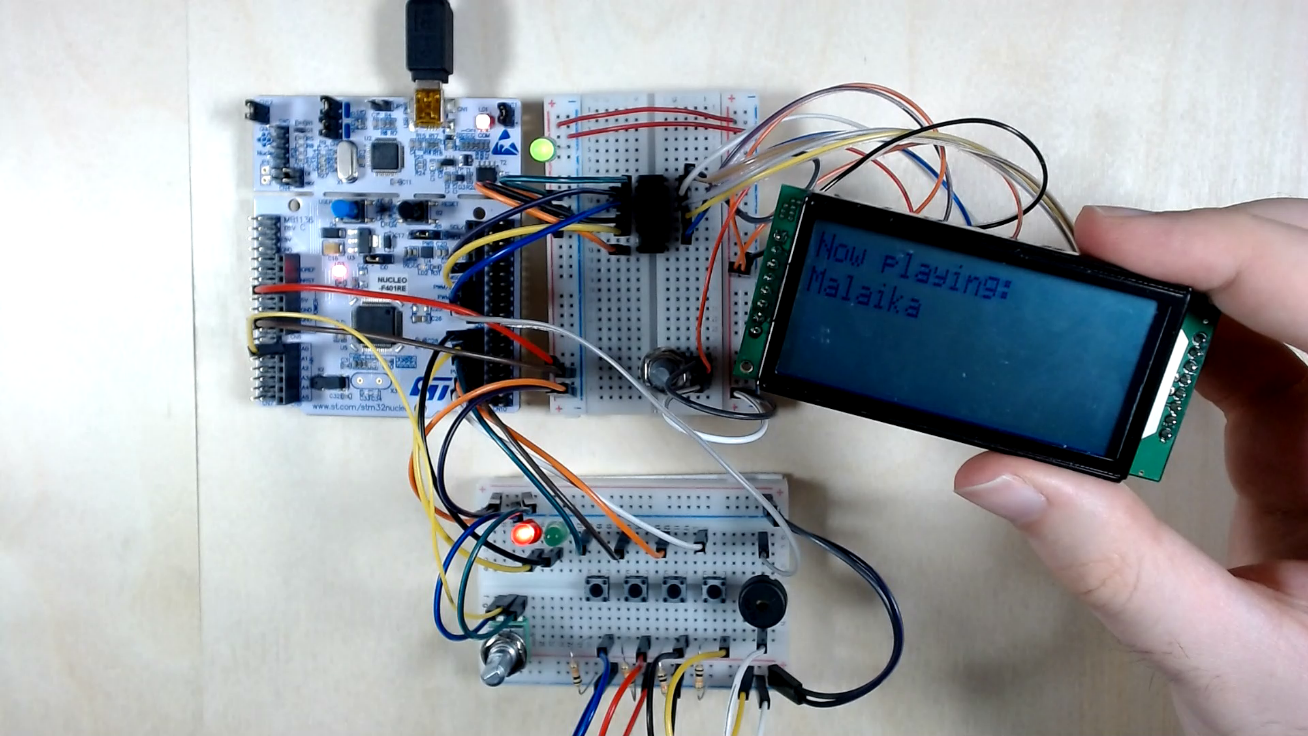
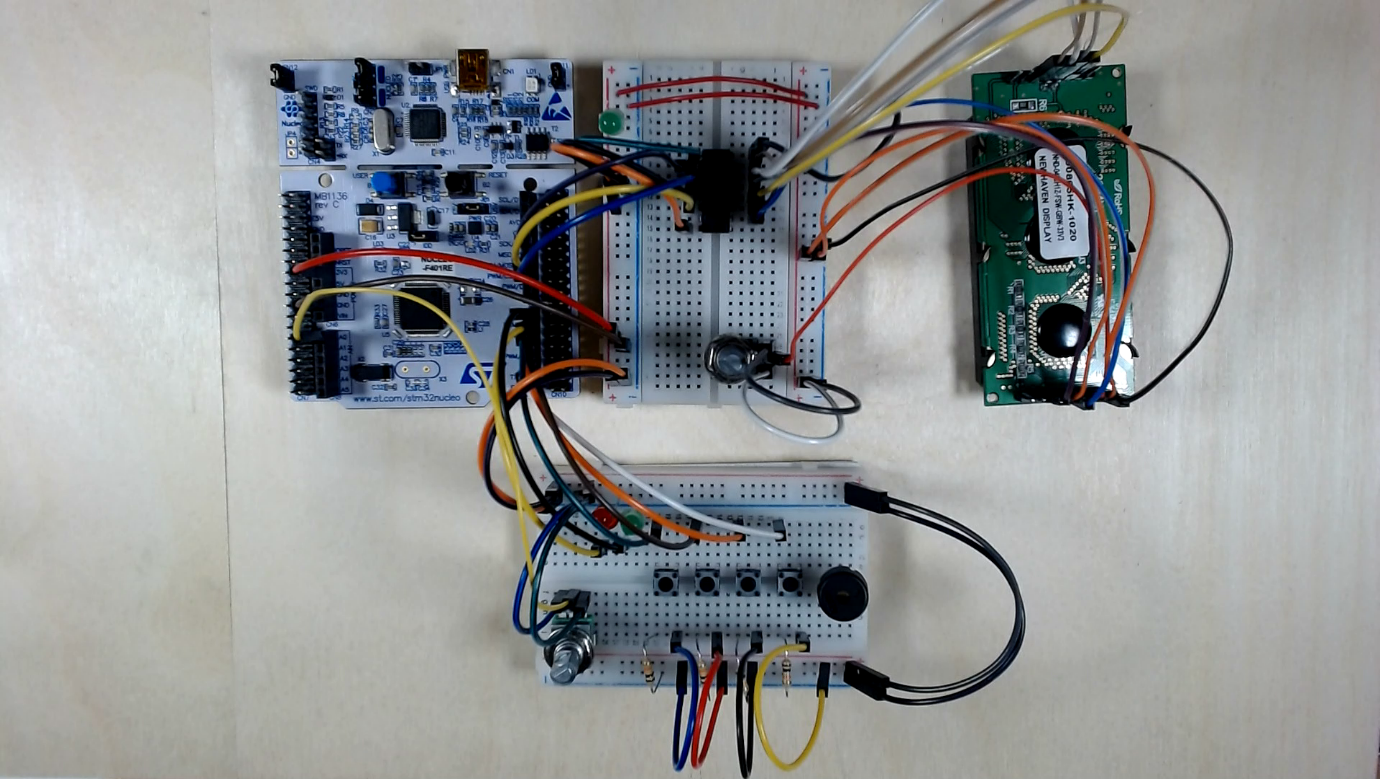


Figure 3: A Complete Build

# Developing the Program

## Planning the Program Structure

As we’re planning to develop a program of moderate complexity, 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 of Figure 4 has been used as simplified structure of the program development described. The Figure shows four main states, through which the Audio Player can move. Between each state is shown the action which 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 Figure. An RTOS-based approach is adopted to implement this state diagram.

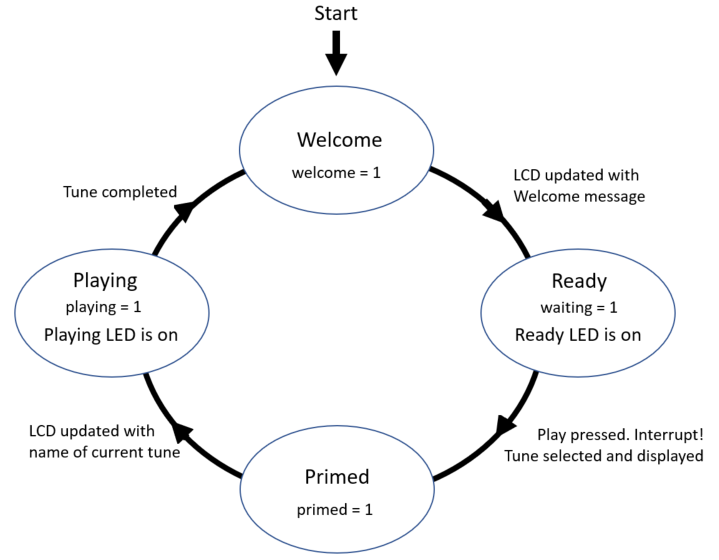


Figure 4: Possible Informal State Diagram for the Audio Player

## Determining the Tasks

When writing an RTOS-based program, an early requirement is to define the tasks or threads. Viewing the diagram of Figure 4, and the intended functionality outlined in the Introduction, the following tasks were chosen, with the Play button triggering an interrupt:

* 1. Update the LCD with Welcome message;
  2. Test the select buttons and configure the new song;
  3. Play the tune.

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

## The Program

A full program listing of the “basic” Audio Player is shown in Program Example 1. This is accompanied by three other files, whose listings are given in the Appendices. The complete file list is therefore:

* main.cpp (Program Example 1);
* Tunes.h (Program Example 2, Appendix 1);
* 4bit\_LCD.h (Program Example 3, Appendix 2);
* 4bit\_LCD.cpp (Program Example 4, Appendix 3).

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 can be seen, each song is formatted into a C/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.

The main action of the program lies in its three threads. Thread 1, **LCD\_cont()**, simply displays the Welcome message when enabled, i.e. 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. Hence 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, indicatingthe 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(1)** loop awaiting calls to the threads, and/or the interrupt.

/\*Final program for Lab 4, with RTOS.

Selects and plays one of 8 tunes, with name displayed on LCD \*/

#include "mbed.h"

#include "tunes.h"

#include "4bit\_LCD.h"

PwmOut speaker(D9); //for piezo sounder

BusIn choose(D3,D4,D5); //for select buttons

DigitalOut playing\_led (D6); //Led indicator

DigitalOut ready\_led (D7); //Led indicator

AnalogIn volume (A0); //for potentiometer

InterruptIn play(D2); //for Play button

Mutex lcd\_mutex;

//These flags indicate state of player

bool welcome = 1; //Indicates welcome state

bool waiting = 0; //waiting for tune selection

bool primed = 0; //tune is selected but hasn't started yet

bool playing = 0; //tune is playing

void LCD\_cont(void);

void Tune\_select(void);

void Play\_tune(void);

//-------------- Threads ----------------//

Thread thread1 ; //Displays welcome message on LCD

void LCD\_cont(void const \*args){

init\_lcd(); //initialise the LCD

//Waiting for song

while(1){

if (welcome){

lcd\_mutex.lock();

clr\_lcd(); //Clear the LCD

print\_lcd("Choose song");

write\_cmd(0xc0);

print\_lcd("Then press Play");

lcd\_mutex.unlock();

thread\_sleep\_for (500);

welcome=0;

ready\_led=1;

} //end of if

} //end of while(1)

}

Thread thread2; //Reads the select buttons and readies the song

void Tune\_select(void const \*args){

while(1){

if(primed){ //"primed" is set by Interrupt

switch(choose){ //read song selection and load song pointer

case 0x00:song\_ptr=&Oranges;break;

case 0x01:song\_ptr=&Cielito;break;

case 0x02:song\_ptr=&Malaika;break;

case 0x03:song\_ptr=&Guten\_Abend;break;

case 0x04:song\_ptr=&Yankee;break;

case 0x05:song\_ptr=&Rasa;break;

case 0x06:song\_ptr=&Matilda;break;

case 0x07:song\_ptr=&Alouetta;break;

} //end of switch

lcd\_mutex.lock();

clr\_lcd(); //Clear the LCD

print\_lcd("Now playing:");

write\_cmd(0xc0);

print\_lcd(song\_ptr->name); //display song name

lcd\_mutex.unlock();

primed=ready\_led=0;

playing=playing\_led=1;

} //end of if

}

}

Thread thread3; //plays the chosen tune

void Play\_tune(void const \*args){

while(1){

if(playing){

//resume PWM operation, stopped after last song

speaker.resume();

for (int i=0;i<=(song\_ptr->length);i++) {

speaker.period(1/(2\*(\*song\_ptr).freq[i])); // set PWM period

speaker=volume; // set duty cycle, hence volume control

thread\_sleep\_for(200\*(\*song\_ptr).beat[i]); //hold for beat prd

}

//indicate end of song

playing\_led=playing=false;

welcome=1;

speaker.suspend();

} //end of if

} //end of while(1)

}

void ISR(){ //responds to press of Play button, and sets "primed"

if(playing==0) //only set "primed" if song not playing

primed=1;

}

//--------------- Main ------------------//

int main() {

ready\_led=1;

play.fall(&ISR); //Int. when Play is pressed, ie on falling edge

//Launch the threads

thread1.start(callback(LCD\_cont,&playing\_led));

thread2.start(callback(Tune\_select,&choose));

thread3.start(callback(Play\_tune,&play));

while (1) {

\_\_wfi(); //wait for interrupt

}

}

*Program Example 1: A Complete “Basic” Program*

Create this program in Studio or the on-line compiler. Each of Program Examples 1-4 should be created as separate files, copied into the same program folder under their given names. The result should appear similar to Figure 5. Build and download the program to your previously constructed hardware.

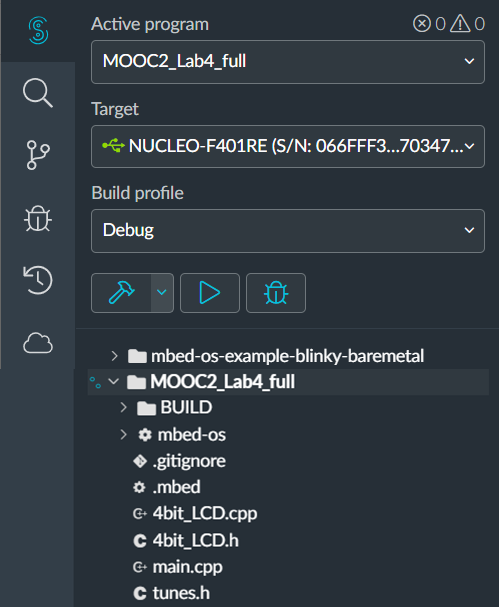


Figure 5: Complete File Structure in Mbed Studio, with Program Name MOOC2\_Lab4\_full

## Troubleshooting

If you have worked carefully and consistently, and with a tiny bit of good fortune, your Audio Player will spring into life first time, and play each tune as you select it. If, along with most other normal human beings, you find your build doesn’t work first time, then try the following for hardware:

* Check all connections, and power supply distribution;
* Devise simple programs which test:
* the LCD circuit alone, writing simple text messages;
* the piezo sounder alone, sending it a PWM signal of period 1 ms;
* the switches and LEDs alone, getting the switches to switch the LEDs.

If you have reasonable confidence in the hardware, but operation is incomplete, try these options:

* LCD remains blank, but rest of operation seems OK, i.e. tunes can be selected and played, and LEDs seem to operate correctly: go back to Lab 1 and work through those introductory programs; it is the same LCD circuit, so should operate with Lab 1 programs.
* Welcome screen is showing but there’s no response to Play: check the Play button is properly and fully connected, and with voltmeter or oscilloscope check that the voltage at D2 switches from 3.3 V to 0 V when Play is pressed.
* Most things seem OK, but the same tune is played (probably “Alouetta”, which selects when no button is pressed) whatever combination is pressed on the select switches: check the select buttons are properly and fully connected, and with voltmeter or oscilloscope that the voltage at D3, D4 and D5 switch from 3.3 V to 0 V when the related switch is pressed.
* When Play is pressed a tune name shows on the LCD, but there is no sound: check your piezo sounder connections, and ensure that the volume control is not set to a minimum.

# Taking Things Further

Explore options to develop the Audio Player further. Some ideas are shown here:

* Replicate and then extend communication with the user, through the PC screen.
* Show the country of origin of the song being played on the LCD, as well as the song name.
* Scroll the song names and numbers across the LCD while in Ready state, giving the user information about available songs.
* Add a new song, replacing one of those that are there, for example if your country is not represented in the song selection. Some musical skill, from you or a friend, is needed here of course!
* Add options for continuous play, for example running through all the songs; the addition of another push button could be helpful here.
* Implement a counter, displayed on the LCD, while a tune is playing. This could for example display which note is playing for any particular song. Hence for the tune “Alouetta” for example it would count from 1 to 19. Choose whether to create a new thread for this, or to embed it into one of those already in place.

Try at least two from this list, or better still, try ideas of your own.

# Conclusion

When 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. You should have taken the basic design described, and found ways of enhancing both the hardware and software. You are over the steepest part of the embedded learning curve, and should be able to look forward to much enjoyable further progress.

# References

*The five web pages below are significant landmarks of the on-line Mbed manual*

1. Introduction to ARM Mbed OS6

[Introduction - Introduction to Mbed OS 6 | Mbed OS 6 Documentation](https://os.mbed.com/docs/mbed-os/v6.6/introduction/index.html)

1. Full Mbed API listing

[Full API list - API references and tutorials | Mbed OS 6 Documentation](https://os.mbed.com/docs/mbed-os/v6.6/apis/index.html)

1. Mbed Tutorials and Examples

[Tutorials and official examples - Tutorials and examples | Mbed OS 6 Documentation](https://os.mbed.com/docs/mbed-os/v6.6/tutorials/index.html)

1. Mbed Components

[Components | Mbed](https://os.mbed.com/components/)

1. Mbed Forums

[Arm Mbed OS support forum - Get support for Arm Mbed OS from our community and support team](https://forums.mbed.com/)

*These two books are excellent reference points while programming in C and/or C++*

1. Peter Prinz and Ulla Kirch-Prinz. (2002). *C Pocket Reference*. O’Reilly. ISBN 0-596-00436-2.
2. Kyle Loudon. (2003). *C++ Pocket Reference*. O’Reilly. ISBN 978-0-596-00496-5.

# Appendix 1: Tunes.h

/\*A set of snips from international songs, mainly of around 8 bars.

Each song is stored as a C structure. \*/

//Define structure to hold songs

struct Songs{

char name[17]; //the song name

int length; //no of notes in song

float freq[30]; //frequency array

float beat[30]; //beat array

};

struct Songs \*song\_ptr; //points to selected song

/\*Note frequencies, for reference.

mid C=262, D=294, E=330, F=349, F#=370, G=392, A=440, B=494,

C=523, C#=554, D=587, E=659, F=698 \*/

struct Songs Oranges={

.name="Oranges & Lemons", //England

.length=11, //enter number one less than no of notes

.freq=659,554,659,554,440,494,554,587,494,659,554,440, //frequencies

.beat=2,2,2,2,2,1,1,2,2,2,2,4 //beat durations

};

struct Songs Cielito={

.name="Cielito Lindo", //Mexico

.length=24,

.freq=659,587,523,440,587,587,523,659,523,392,440,440,392,440,392,698,587,494,392,440,392,349,330,294,262,

.beat=3,2,1,6,3,2,1,1,4,1,1,2,1,2,1,1,2,1,1,2,2,1,1,1,5,

};

struct Songs Malaika={

.name="Malaika", //Tanzania

.length=19,

.freq=294,494,494,440,494,523,440,370,392,392,294,494,494,440,494,523,440,370,392,392,

.beat=2,4,6,1,1,1,2,1,4,6,2,4,6,1,1,1,2,1,4,6,

};

struct Songs Guten\_Abend={

.name="Guten Abend", //Germany

.length=26,

.freq=370,370,440,370,370,440,370,440,588,554,494,494,440,330,370,392,330,330,370,392,330,392,554,494,440,554,587,

.beat=1,1,3,1,1,4,1,1,2,3,1,2,2,1,1,2,2,1,1,4,1,1,1,1,2,2,4,

};

struct Songs Yankee={

.name="Yankee Doodle", //USA

.length=28,

.freq=392,392,440,494,392,494,440,294,392,392,440,494,392,370,294,392,392,440,494,523,494,440,392,370,294,330,370,392,392,

.beat=1.0,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,

};

struct Songs Rasa={

.name="Rasa Sayang", //Malaysia

.length=26,

.freq=370,392,440,440,587,554,494,440,494,392,440,370,587,554,494,494,440,392,370,440,294,370,330,392,277,330,294,//frequency array

.beat=1,1,2,2,2,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2

};

struct Songs Matilda={

.name="Waltzing Matilda", //Australia

.length=20,

.freq=554,554,494,494,440,494,554,440,370,415,440,330,440,554,660,660,660,660,660,660,660,

.beat=2,2,2,2,1.5,0.5,1.5,0.5,1.5,0.5,2,2,1.5,0.5,2,1.5,0.5,2,1.5,0.5,4,

};

struct Songs Alouetta={

.name="Alouetta", //France

.length=18,

.freq=392,440,494,494,440,392,440,494,392,294,392,440,494,494,440,392,440,494,392,

.beat=3,1,2,2,1.5,0.5,1.5,0.5,2,2,3,1,2,2,1.5,0.5,1.5,0.5,2,

};

/\*A spare song!

struct Songs Twinkle={

.name="Twinkle",

.length=13,

.freq=440,440,659,659,740,740,659,587,587,554,554,494,494,440, //frequency array

.beat=2,2,2,2,2,2,4,2,2,2,2,2,2,4, //beat array

};\*/

*Program Example 2: tunes.h*

# Appendix 2: 4bit\_LCD.h

/\*Header file for a simple set of functions to write to 2x16 LCD,

operating in 4-bit mode. \*/

#ifndef FOUR\_BIT\_LCD\_H

#define FOUR\_BIT\_LCD\_H

#include "mbed.h"

#define ENABLE 0x08 //ORed in to data value to strobe E bit

#define COMMAND\_MODE 0x00 //to clear RS line to 0, for command transfer

#define DATA\_MODE 0x04 //to set RS line to 1, for data transfer

//Function Prototypes

void clr\_lcd(void);

void init\_lcd(void);

void print\_lcd(const char \*string);

void shift\_out(int data);

void write\_cmd(int cmd);

void write\_data(char c);

void write\_4bit(int data);

#endif

*Program Example 3: 4bit\_LCD.h*

# Appendix 3: 4bit\_LCD.cpp

/\*A simple set of functions to write to 2x16 LCD,

operating in 4-bit mode. \*/

#include "4bit\_LCD.h"

DigitalOut CS(D10);

SPI ser\_port(D11, D12, D13); // Initialise SPI, using default settings

void init\_lcd(void){ //follow designated procedure in data sheet

thread\_sleep\_for (40);

shift\_out(0x30); //function set 8-bit

wait\_us(37);

write\_cmd(0x20); //function set

wait\_us(37);

write\_cmd(0x20); //function set

wait\_us(37);

write\_cmd(0x0C); //display ON/OFF

wait\_us(37);

write\_cmd(0x01); //display clear

wait\_us(1520);

write\_cmd(0x06); //entry-mode set

wait\_us(37);

write\_cmd(0x28); //function set

wait\_us(37);

}

void write\_4bit(int data, int mode){ //mode is RS line, cmd=0, data=1

int hi\_n;

int lo\_n;

hi\_n = (data & 0xF0); //form the two 4-bit nibbles that will be sent

lo\_n = ((data << 4) &0xF0);

//send each word twice, strobing the Enable line

shift\_out(hi\_n | ENABLE | mode);

wait\_us(1);

shift\_out(hi\_n & ~ENABLE);

shift\_out(lo\_n | ENABLE | mode);

wait\_us(1);

shift\_out(lo\_n & ~ENABLE);

}

void shift\_out(int data){ //Sends word to SPI port

CS = 0;

ser\_port.write(data);

CS = 1;

}

void write\_cmd(int cmd){ //Configures LCD command word

write\_4bit(cmd, COMMAND\_MODE);

}

void write\_data(char c){ //Configures LCD data word

write\_4bit(c, DATA\_MODE); //1 for data mode

}

void clr\_lcd(void){ //Clears display and waits required time

write\_cmd(0x01); //display clear

wait\_us(1520);

}

void print\_lcd(const char \*string){ //Sends character string to LCD

while(\*string) {

write\_data(\*string++);

wait\_us(40);

}

}

*Program Example 4: 4bit\_LCD.cpp*