[redacted]

[redacted]

Educational Media Device - Mini Project

[redacted]

Abstract

An educational media device was designed on DE1-SoC board. The educational media device is aimed on teaching counting from one to ten to kinder garden kids. The main aim behind development of this project is to understand the decoding of WAV audio files and produce an audio output on WM8731 audio codec. The project also focused on use of ARM A9 Private timers, LT24 display, 7-segment displays, Interrupts and push buttons on embedded application development. It focuses on showcasing various programming techniques using C on embedded boards and optimize them using DS-5. The educational media device was designed, developed, and tested for any potential bugs.

Keywords: Educational media device, ARM A9, private timer, WM8731, Interrupts.

Introduction

DE1-SoC is a hardware design platform with Cyclone V SoC and embedded ARM A9 processor. The educational media device is aimed to teach counting from one to ten using images and audio output which aids in teaching.

The media device has following specifications.

The device has two modes.

1. Count Mode – Counts from one to ten. Starts from one, and after one number has completed, the user would have to press **Button 1** to move to the next number. The mode uses audio and visual aid for better understanding.
2. Test Mode – Checks the learning of the child. On displaying a number, the child would have to press the push **Button 1** for displayed number of times. For example, if number two is displayed, the child would have to press the button for 2 times before the system moves to the next number.
3. During start up, the user would have to select which mode to run. The user can select the mode using **Button 2** and **Button 3**. Button 2 corresponds to Count Mode, and Button 3 corresponds to Test Mode.
4. When any mode is running, pressing **Button 4** would reset the board and call mode selector and the user would be able to change the mode.
5. When in Test Mode, without any input from user for 10 seconds, resets the board, and calls mode selector.
6. Animation on LT24 display during mode selector.

From left to right, the button layout on LT24 board is shown below.

Figure 1. Button layout on DE1-SoC board.

Technical Discussion

ARM A9 Private Timer:

Timer allows the user to count the number of clock cycles, which in turn can be used for accurate time keeping. ARM has an in-built timer, ARM A9 Private Timer which is a 32-bit counter that interrupts when it reaches zero. ARM A9 private timer has four registers, Private Timer Load Register, Private Timer Counter Register, Private Timer Control Register and Private Timer Interrupt Status Register and Prescalar. The value on the following registers controls the operation of A9 Private Timer.

Load Register is loaded with a value which equated to the number of cycles to be counted. The value in the Load Register is copied into Control Register provided auto reload mode is enabled. The value inside the Control Register is decremented on every peripheral clock. When the value in Control Register reaches zero the Timer Interrupt Status Register is flagged high. Prescalar is used to modify the clock period for Timer Value decrementing event [1].

The formula for calculating the time taken to decrement is,

$$Time\ taken\ in\ seconds = \frac{\left(\left(\ Prescalar + 1\ \right) X\ \left(\ load\ value + 1\right)\right)}{Peripheral\ Clock}$$

The peripheral clock is 225 MHz for DE1 SoC. For our application, we would set the timer to flag the interrupt high when the counter has counted for one second. Using the above formula, we calculate the values for load value and prescalar to count one second. Figure 2 shows the Private Timer Control Bit. The image is referenced from ARM Technical Reference Manual [1].

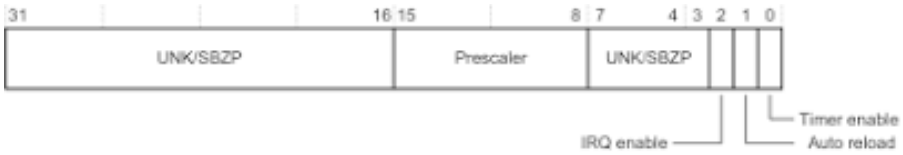


Figure 2. Private Timer Control Bit

7 Segment Display:

Memory can be accessed in C using pointers. Memory mapping is mapping of peripherals in DE1-SoC to memory space in processor. By changing the value stored in the address, the peripherals in DE1 – SoC board can be controlled. The DE1- SoC has six 7-segment displays. From right to left, the first four seven segment displays are controlled with base address 0xFF200020 while the last two seven segment displays can be controlled with 0xFF200030. 7-bit value inside the memory address control the value displayed on seven segment displays. When 1 is set, it turns ON the corresponding bar, and when 0 is set, it turns OFF the corresponding bar.

Push Buttons and Red LED's:

The DE1- SoC board can perform basic I/O function using push buttons and RED LED's. Similarly, to 7 segment display, memory mapping controls the function of Push Buttons and Red LEDs.

The push buttons are controlled using memory address 0xFF200050 and Red LEDs are controlled using memory address 0xFF200000.

LT24 Display:

The LT24 is a colour, touch LCD display module. It has a resolution of 320x240 infamously known as QVGA – The resolution on the first Android smartphones. The LT24 display has ILI9341 display controller. The LT24 display requires 16-bit colour data in RGB565 format. 5-bits for Red and Blue and 6-bits for Green. The display can be controlled using simple Parallel Input/Output (PIO) controller or by using dedicated hardware controller. When controlled using standard PIO interface, the process is quite slow, it would be further discussed in testing part of this report. This process can be speeded up by using custom FPGA peripheral. Controlling LT24 display requires control of many registers and complicated. The driver supplied with the Resources Repo is used with this application. The supplied driver supports this hardware optimised operation using dedicated hardware controller. Hardware optimisation is enabled using pre-processor macro, HARDWARE_OPTIMISED. The macro is defined inside ARM C Compiler settings. When this macro is defined, the driver automatically enables LCD hardware operation.

WM8731 Audio CODEC:

The DE1-SoC has a built-in audio encoder/decoder (CODEC). It is based on WM8731, an audio CODED designed specifically for MP3 audio players and recorders. This IC is interfaced with in-house designed audio processor core [2]. By default, the driver provided in the repo samples the audio at 48000Hz. The audio is synthesised using a 128-sample deep FIFO buffer for input/output.

The function of the WM8731 IC can be controlled by modifying the values in program register. Most of the audio file formats like WAV and MP3 are sampled at 44.1 KHz. The supplied driver in the repo only accepts samples at 48KHz. The driver has been modified to sample at both 44.1 KHz and 48 KHz. During initialisation, the programmer has the option to initialise the IC at either 48KHz or 44.1 KHz. The function 'WM8731_initialise' initialises the IC at 48 KHz. The function 'WM8731_initialise_44' initialises the IC at 44.1 KHz.

Operation of WM8731_initialise_44 Function:

The following information about initialising the IC at 44.1 KHz is obtained from WM8731 datasheet [3].

The DAC filters in WM9731 performs 24-bit processing to convert incoming digital audio data to be sampled at specified sample rate for processing by analog DAC. The sampling rate can

be controlled by De-emphasis control. By default, it is disabled. Setting 11in de-emphasis control can set the sampling rate at 48KHz and 10 can set the sampling rate 44.1 KHz.

WM8731 provides two modes of operation – normal and USB modes. In Normal mode, the user controls the sample rate by using appropriate MCLK and sample rate control register. In USB mode, the user must use a fixed MCLK frequency to generate sample rates.

In Normal mode, MCLK is set up according to desired sample rate. For DAC and ADC sample rate of 48KHz, sample rate registers are set with '0000' and for a sample rate of 44.1KHz, the sample rate registers are set with '1000'. WM8731 can be set for various other sampling rates by changing the register map. Functionality such as input from microphone can be performed. More information about the CODEC is in the datasheet.

Interrupts:

Interrupt is a mechanism of interrupting the current execution of a process [4]. When interrupt occurs during execution of a program, the processor executes a specific instruction at a specific memory location called the vector. The vector memory location contains the address of function/program to load into program counter in case of an interrupt. The function that calls the interrupt is called Interrupt Service Routine.

As we have seen above, the application requires call of interrupt when Button 4 is pressed. IRQ is a hardware interrupt that executes a specific function of code in response to an event. When Button 4 is pressed, Interrupt occurs, and the processor looks up the address of the function to execute from Vector Base Address Register (VBAR). After the interrupt function has completed, the program returns to intsruction it was running before the interrupt was called.

Configuration of the interrupt is a complicated process. The task is simplified with help of supplied IRQ drivers in resources repo.

The function HPS_IRQ_registerHandler(IRQ_LSC_KEYS, mode_select_interrupt); is used initialise when to trigger interrupt. IRQ_LSC_KEYS correspond to IRQ Source Interrupt ID, the information on which event should the interrupt be triggered?. mode_select_interrupt is the function that needs to be called when interrupt is triggered.

When interrupt is triggered, the mode_select_interrupt function is called which resets the mode of the system. After running the interrupt function, the program returns to the instruction executed before the interrupt was called.

Decoding WAV Files:

The following information about WAV files is obtained from references [5,6].

WAV – Waveform Audio File Format is file format standard developed by IBM and Microsoft. WAV is uncompressed file format, meaning raw audio bitstream is stored as raw data in WAV file format. The WAV can contain uncompressed file format, but mostly WAV is used uncompressed.

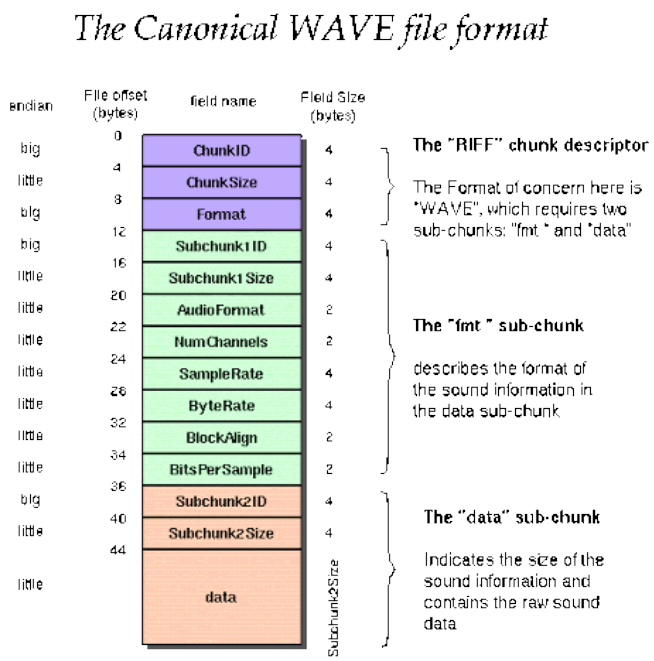


Figure. 3 WAV File Header

Figure 3. shows WAV file header [7]. The file starts with the header followed by raw sound data. The audio data can be either mono or stereo. In case of stereo audio, the odd data in data stream represents data to be sent to left channel and even data in data stream represents the data to be sent to right data channel. To read a WAV file, the file header is first read. After the header is read, the data contains the audio data is read and stored into an audio file buffer. The audio data length, also known as Bits Per Sample can be either 8 or 16 bits wide. When a WAV file needs to be decoded, the raw audio data is fed into FIFO space of WM8731 provided its empty. As the data is uncompressed, no additional processing is necessary. The uncompressed audio stream contains the sampled data which can be processed by WM8731.

In case of other audio formats, like MP3 – Its often compressed using some compression format, meaning that the data needs to be uncompressed before it can be sent to CODEC. MP3 decoding is compute-intensive and bare metal C MP3 decoders are hard to come by. For our application using common compressed audio formats such as MP3 makes the system complex, and uncompressed WAV files can be simple alternatives.

Reading Files from SD Card:

exFAT is file system introduced by Microsoft. It is optimised for flash memory drives such as memory cards and SD cards. File system is kind of like a librarian, it controls how data is stored and retrieved. The DE1 SoC has a micro-SD card which can be used to store WAV files. But, our applications needs to understand exFAT file system. FATFs is exFAT file system module for embedded systems. It is platform independent and can be implemented in microcontrollers with limited resources [8]. The file stored in the SD Card is read using the following steps.

The drive is declared and allocated memory using malloc function. The drive is then mounted using function f_mount(). The file to be read is then declared. Before the file can be read, the size that is going to be read from SD Card needs to be allocated in memory using malloc() function. After allocation the memory, the file is opened using function f_open(). After opening

the file, the function f_read() is used to read the file. The function is called with following parameters f_read (File Object, Buffer to store read data, Number of bytes to read, Number of bytes read). The read pointer contains the data that is read from SD Card.

The file read operation flow is as shown below.

Malloc (File System Pointer) -> f_mount() -> f_open() -> Malloc(File Pointer) -> f_read()

Usually, FPGA On-Chip memory is used to store the program code. But the FPGA has limited memory space. Fortunately, the DE1-SoC is equipped with 1GB DDR3 memory. We can switch to this memory, and we would have the space needed to load WAV music files onto the memory. The scatter file DDRRomRam.scat included in the resources repo is written to use half of the memory for program code and other half for stack and heap. The starting address when using DDR3 memory would be little complicated.

Also, FATFs requires GNU extensions to compile properly. GNU extensions can be enabled as follows " C/C++ Build` -> `Settings` -> `ARM C Compiler 5` -> `Source Language` -> `Enable GNU Extensions (--gnu) ".

Displaying Images with LT24 Driver:

We can store JPEG images in RGB565 format as an array that stores the image data. An online utility is used to create an C file in RGB 565 format [9].

Function call signed int LT24_copyFrameBuffer (const unsigned short * framebuffer, unsigned int xleft, unsigned int ytop, unsigned int width, unsigned int height);

This function requires pointer to frame buffer, x and y co-ordinates on the LT24 display and width and height of the image.

Flow of the system:

During start-up, the board initialises as follows.

1. LT24 display is initialised, program exits on fail.
2. WM8731 audio CODEC is initialised.
3. FATFs is initialised, SD card is mounted, and all the WAV files are copied to their corresponding buffer pointers.
4. Interrupts are initialised.
5. During Loading, 7 segments display "LOAD" meaning that the board is initialising.

After initialising the board, the system enters to mode selector where the user can select the mode of operation by pressing button 2 or button 3.

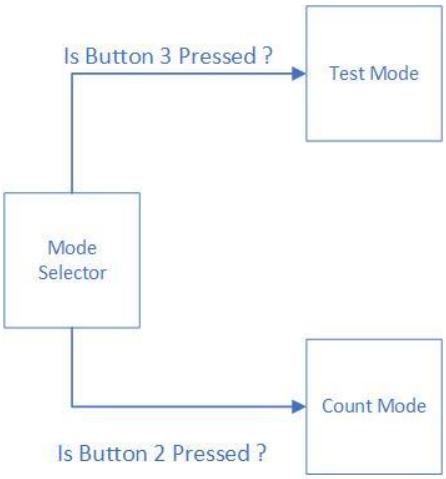


Figure 4. Mode Selector

Count Mode:

Count mode is designed to teach kinder garden kids counting from one to ten. When entered this mode, the system automatically works as follows.

1. Intro Music is played.
2. The current number is informed using audio visual aid.
3. Audio visual aid starts counting from one to current number.
4. Intro music is played again.

After running the above sequence, the system waits for the user to press button 1. If button 1 is pressed, the system increments current number and starts the sequence for next number.

The polling on Count Mode checks if the button has been pressed only after the audio-visual sequence is complete in a blocking manner. After completing the audio-visual sequence, the system waits for the user to press button 1 in a blocking manner indefinitely.

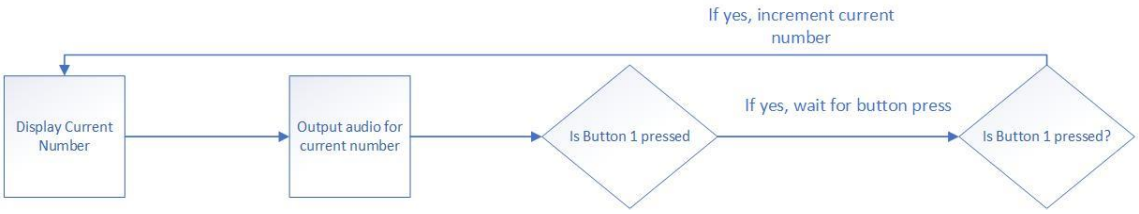


Figure 5. Count Mode – High Level

Test Mode:

The test mode is designed to test the counting knowledge of children. On displaying a number in LT24 display, the user would have to press the button 1 for displayed number of times. That is, if number 2 is displayed on LT24 display, the user would have to press push button for 2 times before the system goes to next number.

Also, after displaying the number with audio aid, if the button is not pressed for the displayed number of times, the system times out after 10 seconds with aid of ARM A9 private timer. The time before timeout is displayed with help of red LEDs.

After pressing the button for correct number of times, the system outputs audio information of current number before proceeding to the next number.

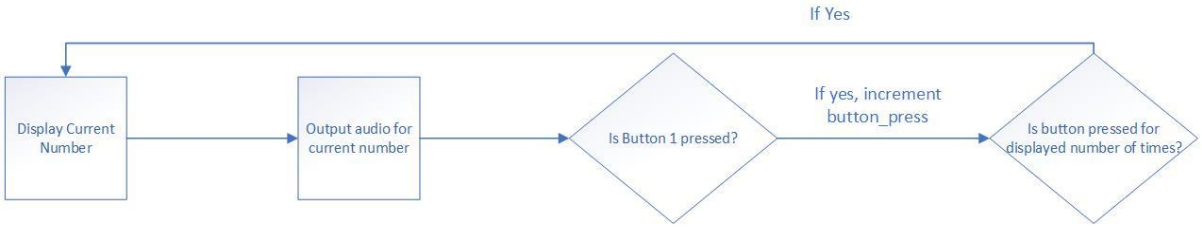


Figure 6. Test Mode – High Level

In either mode, pressing **Button 4** calls a hardware interrupt to enter to mode selector.

Testing and Bug Fixing

Initially, the FATFs module is integrated and tested for its functionality. When using DDRRamRom.scat, the starting address is little odd because of the requirements of the pre-loader.

By using printf function, the read file is opened, read and the WAV file header is printed on to the App Console to verify if the file has been read correctly. If the file has been read correctly, then the file size variable, and the bit size variable would display correct values when printed out to App Console.

Breakpoints are used at various points in the code to verify if the program executes in a way it was intended to. Step over instruction shows the step-by-step execution of line of code and vital in execution of the program. Step over instruction is of great help to debug the code especially in places where loops are used.

By debugging, various bugs in the code has been fixed. Notably, issues when programming that occurs due to improper setting of variables can be fixed. During debugging LT24 display, when not using HARDWARE OPTIMISED driver, the fps of the display is less than one frame per second. Setting of pixels in LT24 display can be seen visually. However, using HARDWARE OPTIMISED driver is more efficient, and the display refreshes much faster with a higher, more useful frame rate. The higher frame rate is utilised to display an animation in LT24 display in mode selector. Videos are just pictures refreshed at faster rates to give a notion of "moving picture" to our eyes. Combination of two pictures taking advantage of faster refresh rate, is used to produce an animation in mode selector.

Though most of the bugs in the system has been fixed, more testing is required to make the application more stable. It can be seen when length of the code is increased, the location of the function call in the program changes how the program works. When interrupt is initialised inside board_initialise() function, the function which is called to initialise the board, the interrupt worked. However, on increasing the length if the code by adding a line to reset red LEDs when timer counts for 10 seconds, the interrupt function is not called when Button 4 is pressed, and the board simply starts running from __main instead. However, when the interrupts are initialised inside main() function instead of board_initialise() function it can be seen that the interrupt function is called correctly and the interrupt works as expected.

Upon researching the internet, it can be understood that such errors do happen in embedded boards due to stack corruption, and the processor is unable to find the interrupt function. Or the Interrupt Vector Table might have an error? Usage of breakpoints is not helpful, as the DS5 breaks when the flow of the program fails as shown above. New methods need to be improvised in order to solve the above bug. Code length has been reduced and the program works for now but increasing the length of the code even by a line causes the program to fail.

Additionally, driver for WM8731 has been developed to work at 44.1KHz, further testing is required to test the functionality of the driver.

Images of disassembly view of registers has been added to Annexure.

Additional Functionality:

When initialising our board, we are copying all the audio files to their corresponding buffers. However, copying files more than 400KB in size resets the board as the watchdog runs out. Watchdog reset function can be added inside FATFs function to reset the watchdog after a specific sequence of events.

Alternatively, we can simultaneously copy files from SD Card and write them to audio buffer. This requires use of pthread functionality in C program. But pthread requires a functioning Operating System as threading and task scheduling are performed by Operating Systems. DE1-SoC does support Linux but use of Linux for such simple applications is unjustified. There are simpler Operating Systems that support threading, but more research needs to be done before implementation.

Smaller images files can be used to decrease program size and perform upscaling on smaller images before displaying them on LT24 display.

Optimisations:

Embedded programmer needs to be careful about resource usage and program execution time. It is important to program loops efficiently to minimise execution time. In while loops, we have decremented and counted down to terminate at zero. Using decrementing loops instead of incrementing loops reduces instruction size after compiler converts it into assembly code.

ARM DS 5 compiler can be set to optimise for both time and space. -Ospace optimises for space and -Otime optimises for time. -O0 means minimum optimisation and -O3 means maximum optimisation. There is no guarantee that compiler can optimise for both space and time. When set to maximum optimisation, our program size is not reduced as most of our code contains array of images, and compiler cannot optimise them.

We have used Default option for optimisation which aims to find balance between both space and time.

```
========================================================================

    Total RO  Size (Code + RO Data)              6602036 (6447.30kB)
    Total RW  Size (RW Data + ZI Data)         536874292 (524291.30kB)
    Total ROM Size (Code + RO Data + RW Data)    6602956 (6448.20kB)

========================================================================

'Finished building target: MiniProject.axf'
' '
```

Figure 7. RO Size after optimisation.

Conclusion:

We have designed an educational media device on DE1-SoC. We have demonstrated the usage of basic I/O in DE1 SoC board along with applications of A9 Private Timers and Interrupts in ARM processors. We have also demonstrated usage of audio CODEC to produce an audio output from an uncompressed audio file. Basic memory mapping registers were designed, and it gave a basic understanding of how bit mapping is used to control peripherals in ARM processors.

As an additional challenge, we made use of FATFs module and integrated it into our application to read WAV audio files from SD card and produce an audio output. Interrupts were used to control the functionality of the board when called upon. More knowledge about stacks and memory handling in ARM processors were understood.

Also, usage of functions, polling using blocking and non-blocking programming techniques are also understood. Optimisations were manually performed, and further optimisations was attempted with help of in-built compiler optimisation in ARM DS5.

More emphasis was given to testing to understand more about the flow of the program. Testing gave a deeper understanding of working of an ARM processor.

References:

[1]. ARM Cortex A-9 MP Core Technical Reference Manual. ARM Developers.

[2]. ELEC 5620M – Embedded Systems Design Notes.

[3]. WM8731/8731L Datasheet – Wolfson Microelectronics.

[4]. Mano, M. Morris. *Computer system architecture.* Prentice-Hall of India, 2003.

[5]. Microsoft Corporation (June 1998). "WAVE and AVI Codec Registries - RFC 2361".

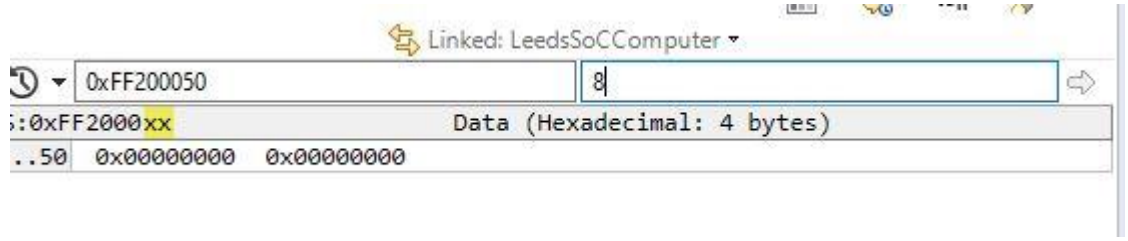
[6]. IBM; Microsoft (August 1991). "Multimedia Programming Interface and Data Specifications 1.0"

[7]. Microsoft WAVE soundfile format (sapp.org)

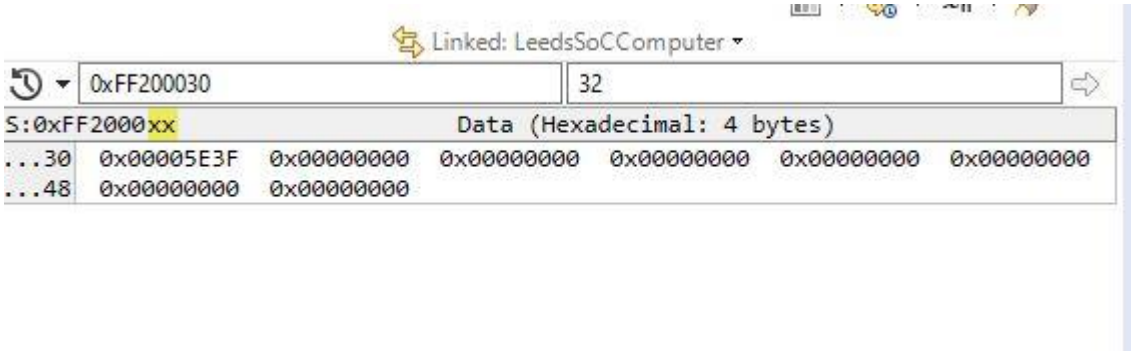
[8]. FATFs- Generic FATFs File System Module – elm-chan.org

[9]. Rinky Dink Electronics - Rinky-Dink Electronics (rinkydinkelectronics.com)

Annexure:



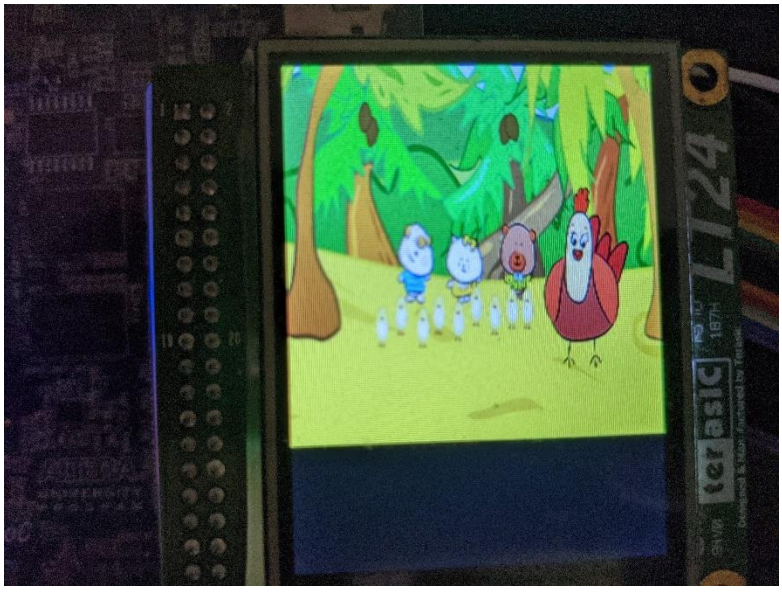
Annexure Figure 1. Memory disassembly in key_ptr.



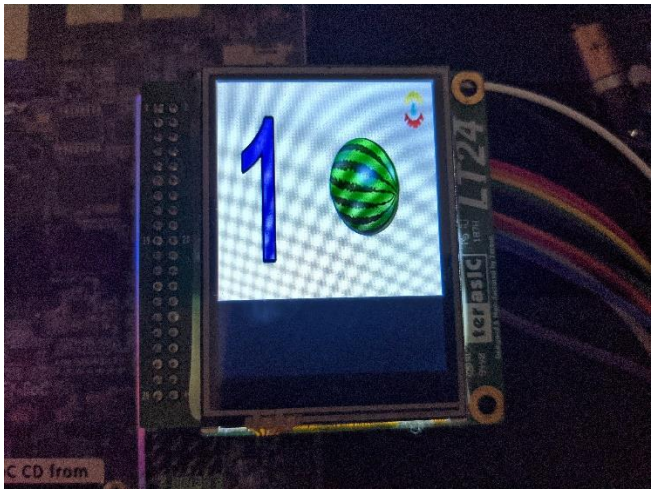
Annexure Figure 2. Memory disassembly for seven segment display.

```
Printing File Information Data
Frequency: 48000
Bits per Sample : 16
Address of Copy_buffer: 1073741772
Address Stored in Copy Buffer: 0
Address stored in Temp Buffer after malloc: 539261960
 To Read : 249662
open success
```

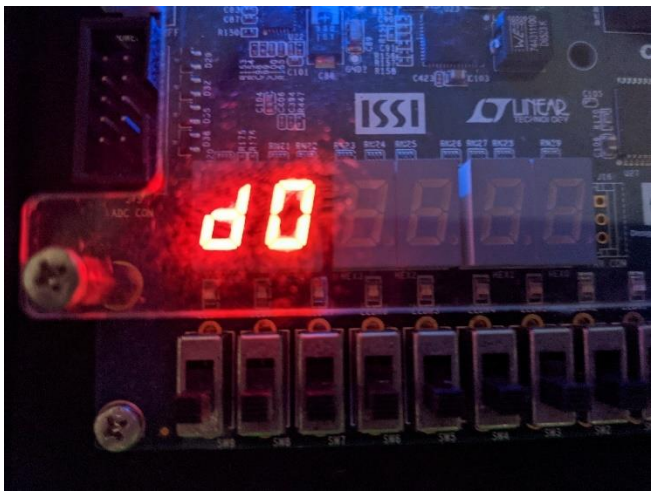
Annexure Figure 3. WAV Header Console Printout



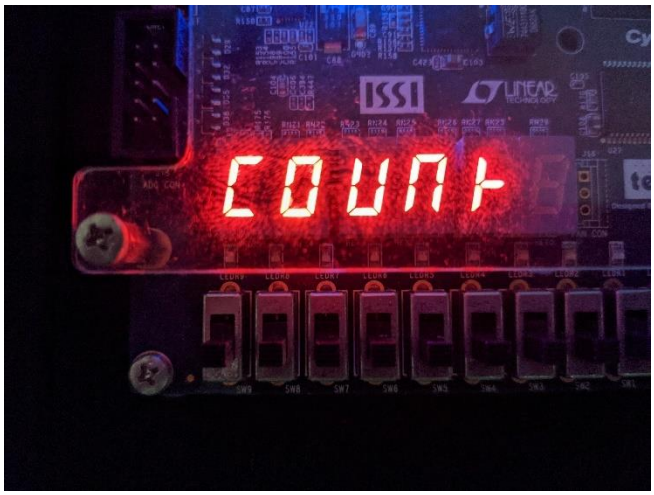
Annexure Figure 4. Welcome Animation



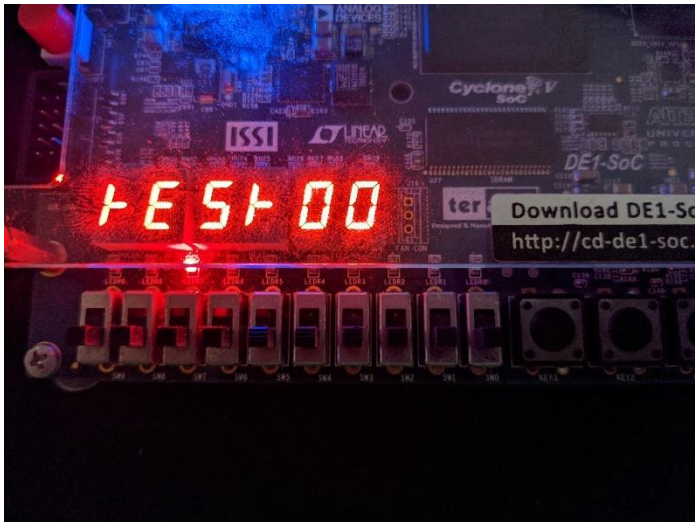
Annexure Figure 5. Counting Audio Visual Aid



Annexure Figure 6. Mode Selector



Annexure Figure 7. Count Mode



Annexure Figure 8. Test mode – 00 represents the current number of button presses.

Program Files

Main.c

```
/* main.c - File

                                        File Name : main.c -> Main function for the program, each function declaration explained seperately.

                                        Author: Arul Prakash Samathuvamani ( APS )

                                        Changelog:



*/

//-------------------------------------------------------------------------------------------------------------

// Header Declaration

#include "DE1SoC_WM8731/DE1SoC_WM8731.h" // Audio Driver
```

```
#include "HPS_Watchdog/HPS_Watchdog.h" // HPS Watchdog driver
#include "DE1SoC_LT24/DE1SoC_LT24.h" // LT24 Display Driver
#include "HPS_usleep/HPS_usleep.h" // usleep driver
#include "Images/image.h" // Function header containing images
#include <stdbool.h> // C Bool Library
#include <stdio.h> // C STD Input/Output
#include <math.h> // C Math Header
#include <stdint.h>
#include "FatFS/ff.h" // FATFs Library
#include <stdlib.h> // C STD Library
#include "SevenSeg.h" // Drivers for seven segment display

// Drivers for Private Timer DMA

#include "PrivateTimer.h"
#include "HPS_IRQ\HPS_IRQ.h"



// Function Definitions

#define MODE_SELECT 0;
#define PLAY_MODE 1;
#define TEST_MODE 2;



// ------------------------------------------------------------------------------------------------------

// Global Variables Declaration

                                        FATFS *fs; // Pointer that shows File system object
```

FIL *intro_file; // File pointer to point to "intro.wav" file.

FIL *one_file; // File pointer to point to "one.wav" file. Plays "One"

FIL *two_file; //File pointer to point to "two.wav" file. Plays "two"

FIL *three_file; // File Pointer to point to "three.wav" file. Plays "three"

FIL *four_file; // File pointer to point to "four.wav" file. Plays "Four"

FIL *five_file; // File pointer to point to "five.wav" file. Plays "five"

FIL *six_file; // File pointer to point to "six.wav" file. plays " six".

FIL *seven_file; // File pointer to point to "seven.wav" file. Plays "seven".

FIL *eight_file; // File pointer to point to "eight.wav" file. Plays "eight".

FIL *nine_file; // File pointer to point to "nine.wav" file. Plays "nine".

FIL *ten_file; // File pointer to point to "ten.wav" file. Plays "ten".

FIL *music_file; // File pointer to point to "music.wav" file. Plays "music"

FILINFO fno; // FILINFO TypeDef - Holds information about object read

FRESULT fr; // FATFS Return Enum - More information - http://elm-chan.org/fsw/ff/doc/rc.html

int16_t *one_buffer;

int16_t *two_buffer;

int16_t *three_buffer;

int16_t *four_buffer;

int16_t *five_buffer;

int16_t *six_buffer;

```
int16_t *seven_buffer;
int16_t *eight_buffer;
int16_t *nine_buffer;
int16_t *ten_buffer;
int16_t *intro_buffer;
int16_t *music_buffer;

volatile unsigned char* fifospace_ptr; // FIFO I2C Free Space Pointer
volatile unsigned int* audio_right_ptr; // Right Audio Channel Pointer
volatile unsigned int* audio_left_ptr; // Left Audio Channel Pointer
volatile unsigned int* LED_ptr = (unsigned int *) 0xFF200000; // LED Pointer Base Address

// set private timer interrupt to private timer interrupt base address
volatile unsigned int *private_timer_interrupt_value = (unsigned int *) 0xFFFEC60C;

unsigned int volume =100000; // Variable to store current audio volume information

// Variables to store buffer file size

unsigned int one_size;     // Size of one_buffer
unsigned int two_size;     // Size of two_buffer
unsigned int three_size;   // Size of three_buffer
unsigned int four_size;    // Size of four_buffer
unsigned int five_size;    // Size of five_buffer
unsigned int six_size;     // Size of six_buffer
```

unsigned int seven_size; // Size of seven_buffer

unsigned int eight_size; // Size of eight_buffer

unsigned int nine_size; // size of nine_buffer

unsigned int ten_size; // size of ten_buffer

unsigned int intro_size; // size of intro_buffer

unsigned int music_size; // size of music buffer

// Declaration of Key_ptr pointer that points to key_press address. 4-bit address, changes accordingly to key press.

volatile unsigned int *key_ptr = (unsigned int *) 0xFF200050;

// Function variables used to indentify the pressed key.

unsigned int key_last_state = 0; // Last key press

unsigned int key_pressed; // denotes what is the currently pressed key

unsigned int mode; // Used to select mode

int temp_mode = 0; // Temporary Mode Store Variable

unsigned int anime_mode = 0; // Variable used for animation mode

// ---

/* Function -> exitOnFail

Function Usage: LT24 Display

Function Takes : LT24 Driver Function, LT24 Success Status

Function Returns: Exits if function call fails

Other Uses: Can be used to determine if other function has successfully run

Author : Arul Prakash Samathuvamani (APS) based on file by David Cowell (DC)

---

*/

```
void exitOnFail(signed int status, signed int successStatus){

                                    if(status != successStatus){
                                       exit((int)status);
                                    }
}


//-------------------------------------------------------------------------------------------------------

// Wav Header TypeDef Declaration
// Author : Arul Prakash Samathuvamani referenced from Online Sources

typedef struct {
```

```
                                uint8_t id[4];                  /** should always contain
"RIFF"      */
                                uint32_t totallength;       /** total file length minus 8       */
                                uint8_t wavefmt[8];         /** should be "WAVEfmt "
*/
                                uint32_t format;            /** Sample format. 16 for PCM
format. */
                                uint16_t pcm;               /** 1 for PCM format
*/
                                uint16_t channels;                  /**         Channels
*/
                                uint32_t frequency;         /** sampling frequency
*/
                                uint32_t bytes_per_second;          /**     Bytes per
second              */
                                uint16_t bytes_per_capture;         /**     Bytes per
capture             */
                                uint16_t bits_per_sample;           /** Bits per sample
*/
                                uint8_t data[4];                    /** should always
contain "data"      */
                                uint32_t bytes_in_data;     /** No. bytes in data
*/
} WAV_Header_TypeDef;

/** Wav header. Global as it is used in callbacks. */

/*
-------------------------------------------------------------------------------------------------------
---

Function: set_hello

Function Usage: Prints hello to directly to seven segment display, written seperately to reduce
run time
```

Function Input : NONE

Function Returns: Void

Author: APS

---
*/

void set_hello () {

// write hello to seven segment display

sevenseg_write (5, 116); // set h to display 5
sevenseg_write (4, 121); // set e to display 4
sevenseg_write (3 , 56); // set l to display 3
sevenseg_write (2, 56); // set l to display 2
sevenseg_write (1, 63); // set o ro display 1
sevenseg_write (0, 0); // turns off other display

}

/*

Function: set_do

Function Usage: Prints do to seven segment display, written seperately to reduce runtime

Function Input : NIL

Function Returns: Void

Author: APS

---
*/

```
void set_do () {

                                        // write do to seven seg display

                                        sevenseg_write ( 5, 94); // set d to display 5
                                        sevenseg_write ( 4, 63 ); // set o to display 4
                                        sevenseg_write ( 3,  0 );  // turns off other display
                                        sevenseg_write ( 2,  0 );  // turns off other display
                                        sevenseg_write ( 1,  0 );  // turns off other display
                                        sevenseg_write ( 0,  0 );  // turns off other display
}
```

/*

Function: set_load

Function Usage: Prints load to seven segment display, written seperately to reduce runtime

Function Input : NIL

Function Returns: Void

Author: APS

---
*/

void set_load () {

// write load to seven seg display

sevenseg_write (5, 56); // set l to display 5
sevenseg_write (4, 63); // set o to display 4
sevenseg_write (3, 119); // set a to display 3
sevenseg_write (2, 94); // set d to display 2
sevenseg_write (1, 0); // turns off other display
sevenseg_write (0, 0); // turns off other display

}

/*

Function: set_count

Function Usage: Prints count to seven segment display, written seperately to reduce runtime

Function Input : NIL

Function Returns: Void

Author: APS

---
*/

```
void set_count () {

                                        // write count to seven seg display

                                        sevenseg_write ( 5, 57 ) ; // set c to display 5
                                        sevenseg_write ( 4, 63 ); // set o to display 4
                                        sevenseg_write ( 3, 62); // set u to display 3
                                        sevenseg_write (2, 55 ); // set n to display 2
                                        sevenseg_write (1, 112); // set t to display 1
                                        sevenseg_write ( 0,  0 );  // turns off other display

}
```

/*

Function: set_test

Function Usage: Prints test to seven segment display, written seperately to reduce runtime

Function Input : NIL

Function Returns: Void

Author: APS

---

*/

```
void set_test () {

                                        // write test to seven seg display

                                        sevenseg_write ( 5, 112 ); // set t to display 5
                                        sevenseg_write ( 4, 121 ); // set e to display 4
                                        sevenseg_write ( 3, 109 ); // set s to display 3
                                        sevenseg_write ( 2, 112 ); // set t to display 2
                                        sevenseg_write ( 1,  0 );  // turns off other display
                                        sevenseg_write ( 0,  0 );  // turns off other display


}
```

/*

Function Name: mode_select_interrupt

Function Description: interrupt function to set the system to sleep or wake up mode

Function Input: NIL

Function Return: VOID

Function Author: APS

---

*/

```
void mode_select_interrupt (HPSIRQSource interruptID, bool isInit, void* initParams){

                                    if(!isInit){

                                        unsigned int press;

                                        press = key_ptr[3]; // read push button interrupt register
                                        key_ptr[3] = press; // set the value again to interrupt register to reset the register

                                        HPS_ResetWatchdog(); // reset watchdog
                                        set_do(); // Ask what to do!

                                        temp_mode = 0; // Set temp mode to zero
                                        mode = 0; // Reset the mode to MODE_SELECT
                                    }
```

```
}

/*
-------------------------------------------------------------------------------------------------------
---

Function: fr_status

Function Usage: Prints status of f_open function on FATFs

Function Input : FRESULT - FATFs Return enum - More Information - http://elm-chan.org/fsw/ff/doc/rc.html

Function Returns: Void

Author: APS

-------------------------------------------------------------------------------------------------------
---
*/

void fr_status ( FRESULT status ) {

                                        // Prints Corresponding status in Console - Check the above website in Function Description for
                                        // more information about the enums.

                                        switch (status) {
                                           case FR_OK:
                                                   printf("open success \n");
                                                   break;
                                           case FR_NO_FILE:
```

```
            printf("NO file \n");
            break;
        case FR_DISK_ERR:
            printf("diskerror \n");
            break;
        case FR_INT_ERR:
            printf("INT ERROR \n");
            break;
        case FR_NOT_READY:
            printf("FR NOT READY \n");
            break;
        case FR_NO_PATH:
            printf("No path \n");
            break;
        case FR_DENIED:
            printf("fr denied \n");
            break;
        case FR_EXIST:
            printf(" FR Exist \n");
            break;
        case FR_INVALID_OBJECT:
            printf("FR invalid Object \n");
            break;
        case FR_WRITE_PROTECTED:
            printf("write protect \n");
            break;
        case FR_INVALID_DRIVE:
            printf("invalid drive \n");
            break;
        case FR_TOO_MANY_OPEN_FILES:
```

```
                printf("too many open \n");
                break;
            case FR_INVALID_NAME:
                printf("invalid name \n");
                break;
            case FR_NOT_ENABLED:
                printf("fr not enabled \n");
                break;
            case FR_NO_FILESYSTEM:
                printf("no file system \n");
                break;
            case FR_TIMEOUT:
                printf("fr time out \n");
                break;
            case FR_NOT_ENOUGH_CORE:
                printf("core error \n");
                break;
            default:
                printf("read failure \n");
        }
}

/*
-------------------------------------------------------------------------------------------
```

Function: file_fetch_initialise

Function Usage: Initialises Files Pointer for FATFs Library

Function Input: NIL, fetches all variables and saves them to pre-set variable pointers known.

Function Working: Allocates the memory, Mounts the File System and Opens the File after allocating memory for the file

Function Returns: None

Author: APS

---

*/

void file_fetch_initialise (){

fs = malloc (sizeof (FATFS)); // allocate memory for FATFS pointer.

HPS_ResetWatchdog(); // Reset Watchdog

fr = f_mount (fs , "" , 0); // Mount the drive to program

// Check if mounting is success

switch(fr){

case FR_OK: // Mount Success
printf ("Mount success \n");
break;
default: // For all other cases mount failed.
printf("error \n");

}

```
// Allocate size for FIL enum

// Allocate memory for all declared files above

intro_file = malloc ( sizeof (FIL));
one_file   = malloc ( sizeof (FIL));
two_file   = malloc ( sizeof (FIL));
three_file = malloc ( sizeof (FIL));
four_file  = malloc ( sizeof (FIL));
five_file  = malloc ( sizeof (FIL));
six_file   = malloc ( sizeof (FIL));
seven_file = malloc ( sizeof (FIL));
eight_file = malloc ( sizeof (FIL));
nine_file  = malloc ( sizeof (FIL));
ten_file   = malloc ( sizeof (FIL));
music_file = malloc ( sizeof (FIL));

HPS_ResetWatchdog(); // Reset Watchdog

printf("Opening Files \n");

// open the corresponding files -> uses function f_open and passes the status to fr_status which prints the status onto console and resets watchdog

// Calls FATFs system to open the file and reset watchdog

printf("Opening intro.wav \n");
fr_status( f_open ( intro_file ,"intro.wav", FA_READ) );  // open intro.wav
HPS_ResetWatchdog(); // reset watchdog
```

```
                printf("Opening one.wav \n");
                fr_status( f_open ( one_file , "one.wav", FA_READ) ); //
opens one.wav
                HPS_ResetWatchdog(); // reset watchdog
                printf("Opening two.wav \n");
                fr_status( f_open ( two_file , "two.wav", FA_READ) ); //
opens two.wav
                HPS_ResetWatchdog(); // reset watchdog
                printf("Opening three.wav \n");
                fr_status( f_open (three_file, "three.wav", FA_READ) ); //
opens three.wav
                HPS_ResetWatchdog(); // reset watchdog
                printf("Opening four.wav \n");
                fr_status( f_open (four_file, "four.wav", FA_READ) ); //
opens four.wav
                HPS_ResetWatchdog(); // reset watchdog
                printf("Opening five.wav \n");
                fr_status( f_open (five_file, "five.wav", FA_READ) ); //
opens five.wav
                HPS_ResetWatchdog(); // reset watchdog
                printf("Opening six.wav \n");
                fr_status( f_open (six_file, "six.wav", FA_READ) ); //
opens six.wav
                HPS_ResetWatchdog(); // reset watchdog
                printf("Opening seven.wav \n");
                fr_status( f_open (seven_file, "seven.wav", FA_READ) );
// opens seven.wav
                HPS_ResetWatchdog(); // reset watchdog
                printf("Opening eight.wav \n");
                fr_status( f_open (eight_file, "eight.wav", FA_READ) ); //
opens eight.wav
                HPS_ResetWatchdog(); // reset watchdog
                printf("Opening nine.wav \n");
```

```
                                    fr_status( f_open( nine_file, "nine.wav", FA_READ) ); //
opens nine.wav
                                    HPS_ResetWatchdog(); // resets watchdog
                                    printf("Opening ten.wav \n");
                                    fr_status( f_open( ten_file, "ten.wav", FA_READ) );
//opens ten.wav
                                    HPS_ResetWatchdog(); // reset watchdog
                                    printf("Opening music.wav \n");
                                    fr_status ( f_open ( music_file, "music.wav", FA_READ )
); // opens music.wav
                                    HPS_ResetWatchdog(); // reset watchdog

}

/*

-----------------------------------------------------------------------------------------------------------
-----------------

Function Name: getPressedKeys

Function Description: Get which key is pressed

Function Input: None

Function Return: Pressed Key

Author : APS

-----------------------------------------------------------------------------------------------------------
-----------------
```

```
*/

unsigned int getPressedKeys () { // To find which key is pressed

   unsigned int key_current_state = *key_ptr; // find what key is pressed

   if( key_current_state != key_last_state){ // if the pressed key is different than previously pressed key

                                          key_last_state = key_current_state;

                                          key_pressed = key_current_state; // set the key_pressed to currently pressed key

                                       }else{

                                          key_pressed = 0; // if nothing is pressed, set the value to zero.

                                       }

   return key_pressed;

}

/*

-------------------------------------------------------------------------------------------------------------------

Function Name: LT24_initialise

Function Description: Initialises LT24 Display
```

Function Input: None

Function Return: VOID

Author : APS

---

*/

```
void display_initialise() {

                                        exitOnFail(
        LT24_initialise(0xFF200060,0xFF200080), //Initialise LCD
        LT24_SUCCESS);                          //Exit if not successful
    HPS_ResetWatchdog();

}
```

/*

Function Name: audio_initialise

Function Description: Initialise Audio on WM8731

Function Input: NIL

Function Returns: VOID

Author: APS

---

*/

void audio_initialise(){

exitOnFail (WM8731_initialise(0xFF203040) , WM8731_SUCCESS); // Initialise WM8731 by calling initialise function in audio driver

WM8731_clearFIFO (true, true) ; // clear FIFO space

// Get memory addresses connecting to I2C bus

fifospace_ptr = WM8731_getFIFOSpacePtr();

audio_left_ptr = WM8731_getLeftFIFOPtr();

audio_right_ptr = WM8731_getRightFIFOPtr();

}

/*

Function Name: sound_out

Function Declaration: Checks if FIFO pointer is available, and passes the input WAV data to output

Function Input: Audio Data, Read Size

Function Return: VOID

Author: APS

---

*/

```
void sound_out( int16_t *audio_buffer, unsigned int audio_size ){

                int current_pt = 0; // data index
                int volume = 10000; // Volume of audio output

                while ( current_pt < (audio_size/2) && mode){ // If the data passed onto FIFO buffer is less than file size

                    if ((fifospace_ptr[2] > 0) && (fifospace_ptr[3] > 0)) { // Checks if FIFO pointer is free
                        *audio_left_ptr = audio_buffer[current_pt] * volume; // Pass data onto buffer
                        *audio_right_ptr = audio_buffer[current_pt+1] * volume; // Pass data onto buffer
                        current_pt = current_pt +2;
                    }
                    HPS_ResetWatchdog(); // reset watchdog
                }

}
```

```
/*
-------------------------------------------------------------------------------------------------------------------
Function Name: intro_out

Function Declaration: Checks if FIFO pointer is available, and passes the input WAV data to output -> Passes Only one sample to DAC

Function Input: Audio Data, Audio Size.

Function Return: VOID

Author: APS

-------------------------------------------------------------------------------------------------------------------
*/

void intro_out( int16_t *audio_buffer, unsigned int audio_size ){

                                        int current_pt = 0; // data index
                                        int volume = 10000; // Volume of audio output

                                        //printf("Audio Size: %d", audio_size);

                                        while ( current_pt < (audio_size/2) ){ // If the data passed onto FIFO buffer is less than file size

                                            if ((fifospace_ptr[2] > 0) && (fifospace_ptr[3] > 0)) { // Checks if FIFO pointer is free
```

```
                                                *audio_left_ptr                 =
audio_buffer[current_pt] * volume; // Pass data onto buffer
                                                *audio_right_ptr                =
audio_buffer[current_pt+1] * volume; // Pass data onto buffer
                                                current_pt = current_pt +2;
                                            }
                                        HPS_ResetWatchdog(); // reset watchdog
                                    }
}



/*
-------------------------------------------------------------------------------------------------------------
-----------------------
Function Name: buffer_size

Function Description: Get Size of Data Buffer

Function Input: FIL Pointer

Function Return: File Size

Function Author: APS

-------------------------------------------------------------------------------------------------------------
----------------------
*/

unsigned int buffer_size ( FIL *input_file ){

                                    WAV_Header_TypeDef TempHeader ; // Declare Temp
WAV Header to get its size
```

int file_size; // Variable for storing file size

file_size = f_size (input_file); // Get Size of the file

file_size = file_size - sizeof (TempHeader); // Subract the header to get actual data size

return file_size; // Returns the size

}

/*

Function Name: fr_read_function

Function Description: Reads the WAV file, understands data from wav header and places them into corresponding buffer pointer and

sends the output to sound out

Function Input: FIL Pointer.

Function Return: Address ?

Author: APS

Changelog :

Modified the functionality to call sound_out from here, Bug when passing malloc pointer back.

Removed Function Inputs

---

*/

unsigned int fr_read_function (FIL *input_file, int16_t *copy_buffer) {

WAV_Header_TypeDef wavHeader; // Cleares a WAV_Header variable to read wav header.

int file_size; // variable fir storing file size

unsigned int read_size =0;

file_size = f_size (input_file); // Read the total size of wav file

printf ("File Size: %u \n", file_size); // Print the data to console

fr_status (f_read (input_file, &wavHeader, sizeof(wavHeader), &read_size)); // Read the WAV file header

// Print File Information on Console

printf("Read the file \n");

printf("Printing File Information Data \n");

printf("Frequency: %u \n", wavHeader.frequency);

printf("Bits per Sample : %u \n", wavHeader.bits_per_sample);

// Whats the actual wav data size ?

```
file_size = (file_size - sizeof(wavHeader));
 // Buffer to copy to

HPS_ResetWatchdog(); // Reset watchdog

// Copy the file to its buffer
printf("Address of Copy_buffer: %d \n", &copy_buffer);
printf("Address Stored in Copy Buffer: %d \n", copy_buffer);

int16_t *temp_buffer;

temp_buffer = (int16_t *)malloc(sizeof(int16_t) *file_size); // Allocate the memory for WAV file data.

printf("Address stored in Temp Buffer after malloc: %d \n", temp_buffer);
printf(" To Read : %d \n", file_size);
// Begin Copy

fr_status ( f_read(input_file, temp_buffer, file_size, &read_size) ); // Read the file

printf("Read Size : %d \n", read_size );
printf("Address Stored in Copy Buffer before copy: %d \n", copy_buffer);
copy_buffer = temp_buffer;
printf("Address Stored in Copy Buffer  after copy: %d \n", copy_buffer);
HPS_ResetWatchdog(); // Reset Watchdog.

//sound_out (copy_buffer, read_size); // Pass the function to sound_out function which calls audio driver
```

```
                HPS_ResetWatchdog(); // Reset Watchdog.

                return temp_buffer;
}

/*
-------------------------------------------------------------------------------------------------------------
Function Name: image_write

Function Description: Writes to LT24 Display, Calls LT24_copyFrameBuffer with starting coordinates as (0,0)

Function Input: Image Array of size 240x240

Function Returns: VOID

Author: APS
-------------------------------------------------------------------------------------------------------------
*/

void image_write(unsigned short image[57600]){

                //LT24_copyFrameBuffer( IMAGE_DATA, STARTING_COORDINATES, ENDING_COORDINATES, IMAGE_HEIGHT, IMAGE_WIDTH);
                exitOnFail (
                        LT24_copyFrameBuffer (image,0,0,240,240), // Copies the input image to LT24 display using its driver.
```

```
                                                    LT24_SUCCESS);

}

/*
-------------------------------------------------------------------------------------------------------------
Function Name: read_files

Function Description: Read All the Files to its Buffer

Function Input: NIL

Function Return: VOID

Function Author: APS

-------------------------------------------------------------------------------------------------------------
*/

void read_files(){

                                    intro_buffer = fr_read_function ( intro_file, intro_buffer );
                                    one_buffer   = fr_read_function ( one_file ,  one_buffer   );
                                    two_buffer   = fr_read_function ( two_file,   two_buffer   );
                                    three_buffer = fr_read_function ( three_file, three_buffer );
                                    four_buffer  = fr_read_function ( four_file,  four_buffer  );
                                    five_buffer  = fr_read_function ( five_file,  five_buffer  );
                                    six_buffer   = fr_read_function ( six_file,   six_buffer   );
```

seven_buffer = fr_read_function (seven_file, seven_buffer);

eight_buffer = fr_read_function (eight_file, eight_buffer);

nine_buffer = fr_read_function (nine_file, nine_buffer);

ten_buffer = fr_read_function (ten_file, ten_buffer);

music_buffer = fr_read_function (music_file, music_buffer);

}

/*

Function Name: initialise_buffer_size

Function Description: Get size of all buffers by calling buffer_size

Function Input: NIL

Function Return: VOID

Function Author: APS

---

*/

void initialise_buffer_size (){

intro_size = buffer_size (intro_file);

one_size = buffer_size (one_file);

two_size = buffer_size (two_file);

```
                                        three_size = buffer_size ( three_file );
                                        four_size  = buffer_size ( four_file );
                                        five_size  = buffer_size ( five_file);
                                        six_size   = buffer_size ( six_file );
                                        seven_size = buffer_size ( seven_file );
                                        eight_size = buffer_size ( eight_file );
                                        nine_size  = buffer_size ( nine_file );
                                        ten_size   = buffer_size ( ten_file );
                                        music_size = buffer_size ( music_file );
                                        printf ("Buffer Size of music : %d \n", music_size );

}



/*
-------------------------------------------------------------------------------------------------------------------------
Function Name: board_initialise

Function Description: Initialise the board during startup

Function Input: NIL

Function Return: VOID

Function Author: APS

-------------------------------------------------------------------------------------------------------------------------
*/
```

```
void board_initialise(){

                    // Initialise LT24 Display

                    set_hello(); // Say hello

                    printf ("Initialising LT24 Display \n");
                    display_initialise(); // initialise display
                    HPS_ResetWatchdog(); // reset watchdog



                    // Initialise Audio on WM8731

                    printf (" Initialising WM8731 DAC \n");
                    audio_initialise(); // Initialise audio
                    HPS_ResetWatchdog(); // Reset watchdog

                    // Initialise A9 Private Timer

                    printf("Initialising Private Timer \n");
                    intialise_timer ( 225000000 ); // Initialise the Timer to
Count One Second

                    // Initialise FileSystem to begin copying files

                    printf(" Initialising FAT SD Card on Board \n");
                    file_fetch_initialise(); // Initialise File System to copy files
from SD Card
                    HPS_ResetWatchdog(); // Reset watchdog
```

```
set_load(); // Say that you are loading


// Read the files and its size

read_files();
HPS_ResetWatchdog();
initialise_buffer_size();
HPS_ResetWatchdog();

// Initialise IRQs

HPS_IRQ_initialise ( NULL );

// Configure button 4 to call interrupt

key_ptr[2] = 0x8;


printf ("Initialising Complete \n");


set_do(); // Ask what to do!

mode = MODE_SELECT; // set mode to select process



}
```

```
/*
-------------------------------------------------------------------------------------------------------------------
Function Name: test_play

Function Description: Test to press the button for correct number of times

Function Input: Input on which number is currently in test, Current Button Press Count

Function Return: VOID

Function Author: APS

-------------------------------------------------------------------------------------------------------------------
*/

void test_play ( int i , int BUTTON_PRESS_COUNT) {

                                        sevenseg_double ( 0,  BUTTON_PRESS_COUNT );

                                        if ( mode ){

                                           switch ( i ) {

                                           case ( 1 ):        // Display one

                                                    image_write( one );
```

```
            break;

    case ( 2 ):
            // Display Two
            image_write ( two_two );

            break;


    case ( 3 ):
            // Display Three
            image_write ( three_three );

            break;

    case( 4 ):

            // Display Four

            image_write ( four_four );

            break;

    case ( 5 ):

            // Display Five

            image_write ( five_five );

            break;
```

```
case ( 6 ):
        // Display Six
        image_write ( six_six );
        break;
case ( 7 ) :
        // Display Seven
        image_write ( seven_seven);
        break;
case ( 8 ):
        // Display Eight
        image_write ( eight_eight );
        break;
case ( 9 ):
        // Display Nine`
        image_write ( nine_nine );
```

```
                    break;

                case ( 10 ):

                    // Display Ten

                    image_write ( ten_ten );

                    break;

                }
            }

}

/*
-------------------------------------------------------------------------------------------------------------
Function Name: count_play

Function Description: Called to play counter from One to Ten.

Function Input: Input on which number to count

Function Return: VOID

Function Author: APS

-------------------------------------------------------------------------------------------------------------
```

```
*/

void count_play( int i ){

                                        // Okay to play, we read the files one by one and play it.

                                        printf("Counting From One to Ten \n");

                                        //int i = 0; // To check which number is currently in count

                                        if ( mode ) {
                                        // To play one

                                           switch ( i ) {



                                           case ( 1 ):        // Display one

                                                   image_write( one );

                                                   // Play One`
                                                   sound_out ( one_buffer , one_size );

                                                   // Play intro music

                                                   sound_out ( intro_buffer, intro_size );

                                                   break;
```

```
case ( 2 ):
        // To play Two
        // All functions have same structure -> Display the image, play the count and then at the end play the song, turn to count

        image_write ( two_two );
        sound_out ( two_buffer ,two_size );

        image_write ( two );
        sound_out ( one_buffer , one_size );

        image_write ( two_two );
        sound_out ( two_buffer, two_size );
        sound_out ( intro_buffer , intro_size );

        break;


case ( 3 ):
        // To Play Three
        image_write ( three_three );
        sound_out ( three_buffer, three_size );

        image_write( three );
        sound_out ( one_buffer , one_size );

        image_write ( three_two );
        sound_out ( two_buffer, two_size );

        image_write ( three_three );
        sound_out ( three_buffer, three_size);
```

```
            sound_out ( intro_buffer, intro_size);

            break;

        case( 4 ):

            // To Play Four

            image_write ( four_four );
            sound_out ( four_buffer, four_size );

            image_write ( four );
            sound_out ( one_buffer, one_size );

            image_write ( four_two );
            sound_out ( two_buffer, two_size );

            image_write ( four_three );
            sound_out ( three_buffer, three_size );

            image_write ( four_four );
            sound_out ( four_buffer, four_size );
            sound_out ( intro_buffer, intro_size );

            break;

        case ( 5 ):

            // To Play Five
```

```
                image_write ( five_five );
                sound_out ( five_buffer, five_size );

                image_write ( five );
                sound_out ( one_buffer, one_size );

                image_write ( five_two );
                sound_out ( two_buffer, two_size );

                image_write ( five_three );
                sound_out ( three_buffer, three_size );

                image_write ( five_four );
                sound_out ( four_buffer, four_size );

                image_write ( five_five );
                sound_out ( five_buffer, five_size );
                sound_out ( intro_buffer, intro_size );

                break;

        case ( 6 ):

                // To Play Six

                image_write ( six_six );
                sound_out ( six_buffer, six_size );

                image_write ( six );
                sound_out ( one_buffer, one_size);
```

```
            image_write ( six_two );
            sound_out ( two_buffer, two_size );

            image_write ( six_three );
            sound_out ( three_buffer, three_size );

            image_write ( six_four );
            sound_out ( four_buffer, four_size );

            image_write ( six_five );
            sound_out ( five_buffer , five_size );

            image_write ( six_six );
            sound_out ( six_buffer, six_size );
            sound_out ( intro_buffer, intro_size );

            break;

    case ( 7 ) :

            // To Play Seven

            image_write ( seven_seven);
            sound_out ( seven_buffer , seven_size );

            image_write ( seven );
            sound_out ( one_buffer, one_size );

            image_write ( seven_two );
```

```
                sound_out ( two_buffer, two_size );

                image_write ( seven_three );
                sound_out ( three_buffer , three_size );

                image_write ( seven_four );
                sound_out ( four_buffer, four_size );

                image_write ( seven_five );
                sound_out ( five_buffer, five_size );

                image_write ( seven_six );
                sound_out ( six_buffer, six_size );

                image_write ( seven_seven );
                sound_out ( seven_buffer, seven_size );
                sound_out ( intro_buffer, intro_size );

                break;

        case ( 8 ):

                // To Play Eight

                image_write ( eight_eight );
                sound_out ( eight_buffer, eight_size );

                image_write ( eight );
                sound_out ( one_buffer, one_size );
```

```
                image_write ( eight_two );
                sound_out ( two_buffer, two_size );

                image_write ( eight_three );
                sound_out ( three_buffer, three_size );

                image_write ( eight_four );
                sound_out ( four_buffer, four_size );

                image_write ( eight_five );
                sound_out ( five_buffer, five_size );

                image_write ( eight_six );
                sound_out ( six_buffer, six_size );

                image_write ( eight_seven );
                sound_out ( seven_buffer, seven_size );

                image_write ( eight_eight );
                sound_out ( eight_buffer, eight_size );
                sound_out ( intro_buffer, intro_size );

                break;

        case ( 9 ):

                // To Play Nine

                image_write ( nine_nine );
                sound_out ( nine_buffer, nine_size );
```

```
image_write ( nine );
sound_out ( one_buffer, one_size );

image_write ( nine_two );
sound_out ( two_buffer, two_size );

image_write ( nine_three );
sound_out ( three_buffer, three_size );

image_write ( nine_four );
sound_out ( four_buffer, four_size );

image_write ( nine_five );
sound_out ( five_buffer, five_size);

image_write ( nine_six );
sound_out ( six_buffer, six_size );

image_write ( nine_seven );
sound_out ( seven_buffer, seven_size );

image_write ( nine_eight );
sound_out ( eight_buffer, eight_size );

image_write ( nine_nine );
sound_out ( nine_buffer, nine_size );
sound_out ( intro_buffer, intro_size );

break;
```

```
case ( 10 ):

        // To Play Ten

        image_write ( ten_ten );
        sound_out ( ten_buffer, ten_size );

        image_write ( ten );
        sound_out ( one_buffer, one_size );

        image_write ( ten_two );
        sound_out ( two_buffer, two_size );

        image_write ( ten_three );
        sound_out ( three_buffer, three_size );

        image_write ( ten_four );
        sound_out ( four_buffer, four_size );

        image_write ( ten_five );
        sound_out ( five_buffer, five_size );

        image_write ( ten_six );
        sound_out ( six_buffer, six_size );

        image_write ( ten_seven );
        sound_out ( seven_buffer, seven_size );

        image_write ( ten_eight );
```

```
                                        sound_out ( eight_buffer, eight_size );

                                        image_write ( ten_nine );
                                        sound_out ( nine_buffer, nine_size );

                                        image_write ( ten_ten );
                                        sound_out ( ten_buffer, ten_size );
                                        sound_out ( intro_buffer, intro_size );
                                        break;

                                    }
                                }

}


/*
-------------------------------------------------------------------------------------------------------------
Function Name: animation

Function Description: Intro Image Animation

Function Input: NIL

Function Return: VOID

Function Author: APS

-------------------------------------------------------------------------------------------------------------
```

```
*/

void animation(){

                                    if ( anime_mode < 10 ){  // if to display animation image1

                                       image_write ( anime1 ); // Animation image 1 to display
                                       anime_mode = anime_mode + 1; // Increment display1

                                    }else if (anime_mode < 20 ){ //play animation image 2

                                       image_write ( anime2 ); // Write animation image 2 to
display
                                       anime_mode = anime_mode + 1; // Increment display2

                                    }else{

                                       anime_mode = 0;
                                    }


}


/*
-------------------------------------------------------------------------------------------------------------
------------------------
Function Name: test_audio_out

Function Description: Out Function for each sound
```

Function Input: Input on which number to produce sound

Function Return: VOID

Function Author: APS

---

*/

```
void test_audio_out ( int COUNT ){

                    switch ( COUNT ){ // What Sound should i Produce ?


                      case ( 1 ):

                            sound_out ( one_buffer, one_size ); // Sound One
                            break;

                      case ( 2 ):

                            sound_out ( two_buffer, two_size ); // Sound
Two
                            break;

                      case ( 3 ):

                            sound_out ( three_buffer, three_size ); // Sound
Three
                            break;
```

```
                case ( 4 ):
                        sound_out ( four_buffer, four_size ); // Sound
Four
                        break;
                case ( 5 ):
                        sound_out ( five_buffer, five_size ); // Sound
Five
                        break;
                case ( 6 ):
                        sound_out ( six_buffer, six_size ); // Sound Six
                        break;
                case ( 7 ):
                        sound_out ( seven_buffer, seven_size ); // Sound
Seven
                        break;
                case ( 8 ):
                        sound_out ( eight_buffer, eight_size ); // Sound
Eight
                        break;
                case ( 9 ):
```

```
                                        sound_out ( nine_buffer, nine_size ); // Sound
Nine
                                        break;

                                    case ( 10 ):

                                        sound_out ( ten_buffer, ten_size ); // Sound Ten
                                        break;

                                }

}



/*
-----------------------------------------------------------------------------------------------------------
-------------------

Function : Main

Function Author: APS

-----------------------------------------------------------------------------------------------------------
-----------------
*/


int main ()
{
```

```
board_initialise();

int IS_START = 0; // If the count is starting for first time

int i = 1; // Variable to pass to count function

int CURRENT_COUNT = 10;

int PRESS_COUNT = 0;

HPS_ResetWatchdog();

intro_out ( music_buffer, music_size ); // Say the application

printf(" Initialising Interrupt \n");

// IRQ interrupt handler for push button press
HPS_IRQ_registerHandler( IRQ_LSC_KEYS, mode_select_interrupt);

HPS_ResetWatchdog();

// Enter the software based on modes
while (1) {

   if ( mode == 0 ){


        HPS_ResetWatchdog(); // Resets the watch dog
        animation(); // Display Animation
```

```
//intro_out ( music_buffer, music_size );
key_pressed = getPressedKeys(); // Get which key is pressed

if ( key_pressed & 0x2 ){ // Button 2 is pressed - Go to Play mode - 1
    set_count(); // Say that you are going to play counting
    temp_mode = 1; // Set temp mode to say you are ready to count
    HPS_ResetWatchdog(); // Reset the watchdog
    IS_START = 1;

}else if ( key_pressed & 0x4 ) {// Button 3 is pressed
    set_test(); // Say that you are going to test counting
    temp_mode = 2; // set temp accordingly
    IS_START = 1;
    CURRENT_COUNT = 10;
    PRESS_COUNT = 0;
    HPS_ResetWatchdog(); // Reset the watchdog

}else if ( key_pressed & 0x1 ){ // Button 1 is pressed
    mode = temp_mode; // Set the mode
    reset_interrupt();
    HPS_ResetWatchdog();
}
```

```
}else if ( mode == 1 ) {

        key_pressed = getPressedKeys();

        if ( IS_START ) {

            i = 1; // set the count to 1
            count_play ( i ); // Play Count 1
            IS_START = 0; // Reset IS_START
            i = i + 1; // increment i
            HPS_ResetWatchdog();
        }
        else if ( key_pressed & 0x1 ) { // Button 1 is
pressed

            count_play ( i );
            if ( i < 10 ) {

                i = i + 1; // Go to next number
            }else {

                i = 1;
            }
            HPS_ResetWatchdog();
        }
        HPS_ResetWatchdog();

}else if ( mode == 2 ) {
```

```
                key_pressed = getPressedKeys ();

                if ( IS_START ) { // Start of the program for first time
                    i = 1;
                    test_play ( i , PRESS_COUNT ); // Display Button Press Count
                    if ( key_pressed & 0x1 ){ // Is Button Pressed?
                        PRESS_COUNT = PRESS_COUNT + 1; // Increment PRESS_COUNT if button is pressed
                        test_play ( i, PRESS_COUNT ); // Display updated PRESS_COUNT
                        sound_out ( one_buffer, one_size ); // Produce the sound
                        i = i + 1; // Go to next count
                        IS_START = 0; // Program ran, no longer initial run
                        PRESS_COUNT = 0; // Reset PRESS_COUNT
                        reset_interrupt(); // Reset Timer -> RUN AGAIN
                        HPS_ResetWatchdog(); // Reset Watchdog
                        CURRENT_COUNT = 10; // CURRENT_COUNT for Timer - Count for 10 seconds

                    } else {

                        if ( CURRENT_COUNT >= 0 ){ // Is count for seconds less than 10
```

```
                                                    if( ( *private_timer_interrupt_value & 0x1 ) ) { // Read Timer Interrupt Value, If YES, then Timer has ran for 1 seconds

                                                        *LED_ptr = ( 1 << CURRENT_COUNT ); // Show it on RED LEDs

                                                        CURRENT_COUNT = CURRENT_COUNT - 1; // Increase Count size

                                                        reset_interrupt(); // Reset Timer -> RUN AGAIN

                                                        HPS_ResetWatchdog(); // Reset Watchdog

                                                    }
                                                }else{

                                                    // *LED_ptr = ( 0 << CURRENT_COUNT ); // Reset RED LED

                                                    mode = 0; // If timer counts for 10 seconds -> Reset to mode selector

                                                    set_do(); // set the seven segment to mode display

                                                }
                                            }
                                        } else {

                                            test_play ( i , PRESS_COUNT ); // Ok, not initial run

                                            if ( ( key_pressed & 0x1 ) & ( PRESS_COUNT < i)) { // check if button is pressed, and less than current number of count

                                                PRESS_COUNT = PRESS_COUNT + 1; // Increase Count for Button Press

                                                test_play ( i, PRESS_COUNT ); // Display PRESS_COUNT on SevenSeg
```

```
                reset_interrupt();  // Reset Timer Interrupt
                CURRENT_COUNT = 10; // CURRENT_COUNT for timer - Count for 10 seconds

            }else if ( PRESS_COUNT == i ) { // If Button Pressed for correct number of times
                test_audio_out ( i ); // Pass to test_audio_out to produce corresponding audio
                PRESS_COUNT = 0; // Reset PRESS_COUNT
                CURRENT_COUNT = 10; // Reset Timer Count
                reset_interrupt(); // Reset Timer Interrupt
                if ( i < 10 ){
                    i = i + 1; // Increase counter count
                }else{
                    IS_START = 1; // Counted for 10 times, set to Intial set count stage
                    i = 0; // For debug purposes set.
                }
            } else{
                if ( CURRENT_COUNT >= 0 ){

                    if( ( *private_timer_interrupt_value & 0x1 ) ) { // Has timer ran for 1 second

                        *LED_ptr = ( 1 << CURRENT_COUNT ); // Show it in RED LED
                        CURRENT_COUNT = CURRENT_COUNT - 1; // Increase timer count
```

```
                reset_interrupt(); // Reset timer
                HPS_ResetWatchdog(); // Reset watchdog

                }
            }else{

                *LED_ptr = ( 0 << CURRENT_COUNT ); // Reset RED LED
                mode = 0; // Go to mode selector
                set_do(); // Set display for mode selector
            }

        }

    }

  }
}


}
```

PrivateTimer.c

```
/*
 * PrivateTimer.c- Stop Watch
 *
 *  Created on: Mar 24, 2021
 *      Author: Arul Prakash Samathuvamani | el20a2ps@leeds.ac.uk
```

```
 */

#include "PrivateTimer.h"

// points the pointers to corresponding base addresses

// set timer pointer to interrupt timer base address
volatile unsigned int *timer_base_ptr          = (unsigned int *)0xFFFEC600;

// set private timer load pointer to timer load address
volatile unsigned int *private_timer_load      = (unsigned int *)0xFFFEC600;

// set private timer pointer to timer value address
volatile unsigned int *private_timer_value     = (unsigned int *)0xFFFEC604;

// set private timer control pointer to timer control base address
volatile unsigned int *private_timer_control   = (unsigned int *)0xFFFEC608;

// set private timer interrupt to private timer interrupt base address
volatile unsigned int *private_timer_interrupt = (unsigned int *)0xFFFEC60C;



// function to initialise the timer
void intialise_timer ( signed int timer_load_value ) {

                                    *private_timer_load = timer_load_value ; // set the timer load value to timer load address
```

```
                                        // set PRESCALAR value of timer to zero. E = 1, A =1, I
=0. i.e enable the timer
                                        // dis
                                        *private_timer_control = ( 0 << 8 ) | (0 << 2) | (1 << 1) | (1
<< 0);



}

// Check if the timer has reached counting down to zero, if yes interrrupt becomes high.
unsigned int interrupt_status ( ) {

                                        unsigned int interrupt_value = *private_timer_interrupt; //
declaration used for testing

                                        // return the value of interrupt.
                                        return *private_timer_interrupt;

}



// reset the timer interrupt. Timer starts to count again.
void reset_interrupt ( ) {

                                        // reset the value of interrupt.
                                        *private_timer_interrupt = 0x1;

}
```

PrivateTimer.h

```
/*
 * PrivateTimer.c- Stop Watch
```

```
 *
 *  Created on: Mar 24, 2021
 *      Author: Arul Prakash Samathuvamani | el20a2ps@leeds.ac.uk
 */



#ifndef PRIVATETIMER_H_
#define PRIVATETIMER_H_

// function to initialise the timer
void intialise_timer( signed int timer_load_value );

// Check if the timer has reached counting down to zero, if yes interrrupt becomes high.
unsigned int interrupt_status ( void ) ;

// reset the timer interrupt. Timer starts to count again.
void reset_interrupt ( void );

#endif /* PRIVATETIMER_H_ */
```

SevenSeg.c

```
/*
 * SevenSeg.c
 *
 *                                      7-Segment Display Driver
 *
 *  Created on: Mar 23, 2021
 *      Author: Arul Prakash Samathuvamani | Based on Driver in Unit 1 Examples

 Changelog:
```

```
 Driver Fetches the corresponding HEX value to be fed from a array to map it.
 */

#include "SevenSeg.h"

/* Driver Functionality
 *
 * 7-segment display displays the value denoted by 7-bit address.
 *
 * X X X X X X X -  The corresponding dash turns on and off based on the value of X. Please refer technical report for mapping of bits.
 *
 */



// set the lower base address for 7-segment display. Denotes display 0-4
volatile unsigned char *sevenseg_base_low_addr = (unsigned char *) 0xFF200020;



// set the higher base address for 7-segment display. Dentoes display 5 and 6
volatile unsigned char *sevenseg_base_high_addr = (unsigned char *) 0xFF200030;

#define number_of_low_display 4 // Define the number of displays addressed by lower base address.

#define number_of_high_display 2 // Define the number of displays addressed by higher base address.

unsigned int number_map[10] = { 63,6,91,79,102,109,125, 7,127, 103 };
```

```
// Write the corresponding value to 7-Segment memory address.
void sevenseg_write(unsigned int display, unsigned char value){

                    if(display < number_of_low_display ) { // if the display number is in lower base

                        sevenseg_base_low_addr[display] = value; // write the value to lower base address

                    }else{

                        display = display - number_of_low_display; // else find the corresponding display in higher base address
                        sevenseg_base_high_addr[display] = value; // and write the value to higher base address
                    }

}

// Sets the value for single display
void sevenseg_single(unsigned int display, unsigned int value){


                    if(value < 10 ){ // for a single display, values can be from 0-9 for stop watch or the display is turned off

                            sevenseg_write ( display, number_map[value] );

                    }else{
                        sevenseg_write(display, 0); // else turn off the display.
```

```
                                    }

}

// used to set value in two displays.

void sevenseg_double(unsigned int display, unsigned int value){

                                    // if the value is less than 10, then only one display is needed
                                    if(value <10){

                                       sevenseg_single  (display,value); // display the value in
first display
                                       sevenseg_single  (display+1,0); // and display zero in the
next

                                    }

                                    // if the value is greater than 10, then both the displays are
needed
                                    else{

                                       sevenseg_single (display,(value % 10)); // display the
lower digit in first display
                                       sevenseg_single (display+1,( value / 10 ) ); // display the
higher digit in second display

                                    }
```

}

SevenSeg.h

```
/*
 * SevenSeg.h
 *
 *                                  7-Segment Display Driver
 *
 *  Created on: Mar 23, 2021
 *      Author: Arul Prakash Samathuvamani | Based on Driver design in Unit 1 Examples.
 */

#ifndef SEVENSEG_H_
#define SEVENSEG_H_



void sevenseg_write (unsigned int display , unsigned char value);


void sevenseg_single(unsigned int display, unsigned int value);


void sevenseg_double(unsigned int display, unsigned int value);


#endif /* SEVENSEG_H_ */
```

Truncated Image Files – Two files are given for sample due to lack of space. All image files are generated online and is in same format

```
#if defined(__AVR__)
    #include <avr/pgmspace.h>
#elif defined(__PIC32MX__)
    #define PROGMEM
#elif defined(__arm__)
```

```
    #define PROGMEM
#endif

const unsigned short anime1[57600] ={…};


#if defined(__AVR__)
    #include <avr/pgmspace.h>
#elif defined(__PIC32MX__)
    #define PROGMEM
#elif defined(__arm__)
    #define PROGMEM
#endif

const unsigned short anime2[57600] ={…};
```

WM8731 Driver

/*

Changelog Arul Prakash Samathuvamani:

Add Initialise Function for 44.1 Khz Sampling Frequency.

*/

#include "DE1SoC_WM8731.h"

#include "../HPS_I2C/HPS_I2C.h"

//

// Driver global static variables (visible only to this .c file)

//

//Driver Base Address

volatile unsigned int *wm8731_base_ptr = 0x0;

//Driver Initialised

bool wm8731_initialised = false;

//

```
// Useful Defines
//

//WM8731 ARM Address Offsets
#define WM8731_CONTROL    (0x0/sizeof(unsigned int))
#define WM8731_FIFOSPACE  (0x4/sizeof(unsigned int))
#define WM8731_LEFTFIFO   (0x8/sizeof(unsigned int))
#define WM8731_RIGHTFIFO  (0xC/sizeof(unsigned int))

//I2C Register Address Offsets
#define WM8731_I2C_LEFTINCNTRL   (0x00/sizeof(unsigned short))
#define WM8731_I2C_RIGHTINCNTRL  (0x02/sizeof(unsigned short))
#define WM8731_I2C_LEFTOUTCNTRL  (0x04/sizeof(unsigned short))
#define WM8731_I2C_RIGHTOUTCNTRL (0x06/sizeof(unsigned short))
#define WM8731_I2C_ANLGPATHCNTRL (0x08/sizeof(unsigned short))
#define WM8731_I2C_DGTLPATHCNTRL (0x0A/sizeof(unsigned short))
#define WM8731_I2C_POWERCNTRL    (0x0C/sizeof(unsigned short))
#define WM8731_I2C_DATAFMTCNTRL  (0x0E/sizeof(unsigned short))
#define WM8731_I2C_SMPLINGCNTRL  (0x10/sizeof(unsigned short))
#define WM8731_I2C_ACTIVECNTRL   (0x12/sizeof(unsigned short))

//Initialise Audio Controller
signed int WM8731_initialise ( unsigned int base_address) {
    signed int status;
    //Set the local base address pointer
    wm8731_base_ptr = (unsigned int *) base_address;
    //Ensure I2C Controller "I2C1" is initialised
    if (!HPS_I2C_isInitialised(0)) {
        status = HPS_I2C_initialise(0);
        if (status != HPS_I2C_SUCCESS) return status;
```

```
    }

    //Initialise the WM8731 codec over I2C. See Page 46 of datasheet

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_POWERCNTRL    <<9) | 0x12); //Power-up chip. Leave mic off as not used.

    if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_LEFTINCNTRL    <<9) | 0x17); //+4.5dB Volume. Unmute.

    if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_RIGHTINCNTRL <<9) | 0x17); //+4.5dB Volume. Unmute.

    if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_LEFTOUTCNTRL <<9) | 0x70); //-24dB Volume. Unmute.

    if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_RIGHTOUTCNTRL<<9) | 0x70); //-24dB Volume. Unmute.

    if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_ANLGPATHCNTRL<<9) | 0x12); //Use Line In. Disable Bypass. Use DAC

                                        if (status != HPS_I2C_SUCCESS) return status;

                                        status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_DGTLPATHCNTRL<<9) | 0x06); //Enable High-Pass filter. 48kHz sample rate.

    if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_DATAFMTCNTRL <<9) | 0x4E); //I2S Mode, 24bit, Master Mode (do not change this!)

                                        if (status != HPS_I2C_SUCCESS) return status;

                                        status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_SMPLINGCNTRL <<9) | 0x00); //Normal Mode, 48kHz sample rate

                                        if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_ACTIVECNTRL    <<9) | 0x01); //Enable Codec

    if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_POWERCNTRL    <<9) | 0x02); //Power-up output.
```

```
    if (status != HPS_I2C_SUCCESS) return status;
    //Check if the base pointer is valid. This allows us to use the library to initialise the I2C side only.
    if (base_address == 0x0) return WM8731_ERRORNOINIT;
    //Mark as initialised so later functions know we are ready
    wm8731_initialised = true;
    //Clear the audio FIFOs
    return WM8731_clearFIFO(true,true);
}



signed int WM8731_initialise_44 ( unsigned int base_address) {
    signed int status;
    //Set the local base address pointer
    wm8731_base_ptr = (unsigned int *) base_address;
    //Ensure I2C Controller "I2C1" is initialised
    if (!HPS_I2C_isInitialised(0)) {
        status = HPS_I2C_initialise(0);
        if (status != HPS_I2C_SUCCESS) return status;
    }
    //Initialise the WM8731 codec over I2C. See Page 46 of datasheet
    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_POWERCNTRL    <<9) | 0x12); //Power-up chip. Leave mic off as not used.
    if (status != HPS_I2C_SUCCESS) return status;
    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_LEFTINCNTRL   <<9) | 0x17); //+4.5dB Volume. Unmute.
    if (status != HPS_I2C_SUCCESS) return status;
    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_RIGHTINCNTRL <<9) | 0x17); //+4.5dB Volume. Unmute.
    if (status != HPS_I2C_SUCCESS) return status;
    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_LEFTOUTCNTRL <<9) | 0x70); //-24dB Volume. Unmute.
```

```
    if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_RIGHTOUTCNTRL<<9) | 0x70); //-24dB Volume. Unmute.

    if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_ANLGPATHCNTRL<<9) | 0x12); //Use Line In. Disable Bypass. Use DAC

                                        if (status != HPS_I2C_SUCCESS) return status;

                                        status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_DGTLPATHCNTRL<<9) | 0x04); //Enable High-Pass filter. 44.1kHz sample rate.

    if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_DATAFMTCNTRL <<9) | 0x4E); //I2S Mode, 24bit, Master Mode (do not change this!)

                                        if (status != HPS_I2C_SUCCESS) return status;

                                        status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_SMPLINGCNTRL <<9) | 0x20); //Normal Mode, 44.1kHz sample rate

                                        if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_ACTIVECNTRL <<9) | 0x01); //Enable Codec

    if (status != HPS_I2C_SUCCESS) return status;

    status = HPS_I2C_write16b(0, 0x1A, (WM8731_I2C_POWERCNTRL <<9) | 0x02); //Power-up output.

    if (status != HPS_I2C_SUCCESS) return status;

    //Check if the base pointer is valid. This allows us to use the library to initialise the I2C side only.

    if (base_address == 0x0) return WM8731_ERRORNOINIT;

    //Mark as initialised so later functions know we are ready

    wm8731_initialised = true;

    //Clear the audio FIFOs

    return WM8731_clearFIFO(true,true);

}


//Check if driver initialised

bool WM8731_isInitialised() {
```

```
    return wm8731_initialised;
}

//Clears FIFOs
// - returns true if successful
signed int WM8731_clearFIFO( bool adc, bool dac) {
    unsigned int cntrl;
    if (!WM8731_isInitialised()) return WM8731_ERRORNOINIT; //not initialised
    //Read in current control value
    cntrl = wm8731_base_ptr[WM8731_CONTROL];
    //Calculate new value - with corresponding bits for clearing adc and/or dac FIFOs
    if (adc) {
        cntrl |= (1<<2);
    }
    if (dac) {
        cntrl |= (1<<3);
    }
    //Assert reset flags
    wm8731_base_ptr[WM8731_CONTROL] = cntrl;
    //Clear the flags
    if (adc) {
        cntrl &= ~(1<<2);
    }
    if (dac) {
        cntrl &= ~(1<<3);
    }
    //Then clear reset flags
    wm8731_base_ptr[WM8731_CONTROL] = cntrl;
    //And done.
    return WM8731_SUCCESS; //success
```

```
}

//Get FIFO Space Address
volatile unsigned char* WM8731_getFIFOSpacePtr( void ) {
    return (unsigned char*)&wm8731_base_ptr[WM8731_FIFOSPACE];
}

//Get Left FIFO Address
volatile unsigned int* WM8731_getLeftFIFOPtr( void ) {
    return &wm8731_base_ptr[WM8731_LEFTFIFO];
}

//Get Right FIFO Address
volatile unsigned int* WM8731_getRightFIFOPtr( void ) {
    return &wm8731_base_ptr[WM8731_RIGHTFIFO];
}
```