School of Electronic and Electrical Engineering Coursework Cover Sheet

Submit coursework, with this cover sheet, to the Submission Box / Student Support Office before the specified deadline.

This sheet may be removed so repeat your student ID, etc. on the title page of your coursework.

Student Number	**Name** *(optional)*
201562426	**Weiming Xu**

Module Code	ELEC 5620M
Module Title	Embedded System Design
Title of Coursework Item	Assignment 3
Name of Lecturer	**David Cowell**

STUDENT DECLARATION (taken to include the "LU Declaration of Academic Integrity")

- I confirm that this coursework is entirely my original work
- I have not allowed any other student to gain access to any part of this coursework in either physical or electronic form.
- No part of the material was cut-and-pasted in digital form from another source (even if subsequently altered)

I understand that failure to observe any of these requirements will be regarded as cheating or plagiarism, and the appropriate University sanctions will apply.

As an absolute minimum, this will result in a mark of zero being returned for this coursework item and the work having to be re-done.

Signed:……Weiming Xu…… Date…….. 5/5/2022…….

Abstraction

This report will introduce the functions of the design, the software and hardware used in this project, the algorithm and code debugging method to achieve the goals, and the hardware test process. Finally, this report will show the result of this project and conclude.

Introduction

In this mini-project, a video game named Find the Red Ball is designed. In this game, there are three bottles put on the desk. A red ball is covered by one of the bottles. Therefore, the players need to uncover one of the bottles to find where the red ball is. They need to solve three problems, and they will be graded for each selection. After finishing the game, they can see the final points in the seven-segment display. Besides, there is no limitation on the times of choice. Consequently, all the players can finish the game, but their grades will differ.

Software

In this project, Eclipse for ARM DS-5 5.27.1 is used to build the environment to compile, debug and test the code [1].

Hardware

In this project, the DE1-SoC with ARM Cortex-49 is used as hardware to operate the command and test whether all the functions work well [2]. Five peripherals are used in this project and introduced in the following.

- LT24 display board

  On the LT24 display board, a desk and three bottles upturned on the desk will be displayed (shown in appendix A, figure 1). The bottles will be labeled from number one to number three from right to left in this report. The players can select the bottle and uncover it. If the red ball is under this bottle, the red ball will show under the bottle (shown in appendix A, figure 2). If not, it will be empty under the bottle (shown in

appendix A, figure 3). After finishing the game, a victory picture will show on the screen (shown in appendix A, figure 4).

- Pushbuttons Keys

  Four buttons are used in this project. The buttons will be named from key 1 to key 4 in the number sequence from left to right. The functions are introduced in **Table 1**.

Table 1

Pushbutton keys	Function
Key 1	Uncover bottle 1
Key 2	Uncover bottle 2
Key 3	Uncover bottle 3
Key 4	Restart the game

- Seven-Segment Display

  The seven-segment display is used to display the grade of players. The initial point is five points. If the players select the right bottle and find the red ball, they will get one point. Therefore, the current grade will add one and display it on the seven-segment display. By contrast, if the players do not choose the right bottle, they will lose one point. The total point will munis one and displayed on the seven-segment display. Consequently, players can know their current point on seven-segment in real-time.

- Red LED

  Red LED is used to inform the player they found the red ball. The red LED will light on if the players find the red ball. The LED will not react if they do not find the red ball.

- Audio

  The audio peripheral will play the background music during the game.

Variable and Function

Tool

In this program, several drivers will be used to achieve some functions.

- Watchdog

  The watchdog is used to monitor the execution of code. The program will reset the watchdog periodically, but the watchdog can not be reset when the program crashes. Subsequently, the watchdog program will make the processor perform a hard reset [1].

- Seven segment display

  Seven segment display is used to display the grade the players get in real-time. A pair of parallel I/O ports control the seven-segment display in the Leeds SoC Computer (LSC). For each display, the lower 7-bits are used to manage the seven-segment. If writing 1 to any bit, the corresponding LED will be light on [3].

- ARM A9 Private Timer

  There is a part called ARM A9 Private Timer (MPCore Timer) within the processor, which is used in this project. A clock sign provided from the CPU frequency will be used to produce an accurate count rate by this timer. Table 3 shows the register map of the private timer [4].

Table 3

Address	**31…16**	**15…8**	**7…3**	**2**	**1**	**0**	
0xFFFEC600	Load Value						Load
0xFFFEC604	Current Value						Counter
0xFFFEC608		Prescaler		I	A	E	Control
0xFFFEC60C						F	Interrupt Status

In the address of Load, the load value will be stored. If bite A is set to 1, the counter register will be reloaded when the timer reaches zero. By contrast, if bite A is set to 0, the counter register will stop when the timer reaches zero. As for bite E, it is set to fix the frequency for the timer to count down the time, and the frequency could be adjusted by setting Prescaler. The F will be set high if the current value reaches zero. Furthermore, the code will poll the flag F until bite F is set. Finally, the flag will be cleared by writing the 1 to bite F [4].

Variable

Seven variables are being used in this program, as shown in **table 4**.

Table 4

Variable Name	Type	Value	Function
taskID	Int	0	Use to figure out which question we are in
single_hex_display	Int	5	Use to display the grade
phase	Double	0	The initial phase
inc	Double	440.0 * PI2 / F_SAMPLE	The increment of phase
ampl	Double	8388608.0	Tone amplitude.
keys_pressed	Int	0	Use to determine which key is pressed
taskCount	Int	3	The number of questions.

Peripheral

The peripherals used will be shown in **table 5** [5].

Table 5

I/O Peripheral	Address
4x Push Buttons	0xFF200050
7-segment	0xFF200000
LT24 LCD	0xFF200060
Audio System	0xFF203040
Watchdog Timer	0xFFD0200C
Private Timer	0xFFFF0000

Array of Function Pointer

There are twelve functions in the array. The first three are about the questions, and the lefts are about the answers.

- Question function

The question function will display the picture of questions on the LT24 LCD. There are three questions, so three questions function will be used to display the question on the LT24 LCD

- Answer function

  There are three questions and three different answers to each question. Therefore, there are nine answers in total. In this report, the answer functions will be divided into right answer functions and wrong answer functions. The right answer function will display the picture of the right answer on the LT24 LCD and the light on the red LED. Besides, when the right answer is selected, the grade will add 1 and be displayed on the seven-segment display. The wrong answer function will show the picture of the wrong answers on the LT24 LCD, and the grade will be minus 1 and be displayed on the seven-segment display. The red LED will not be light on if the wrong selection is activated.

Imagine

Twelve images are used in this project. Their functions are shown in table 6. All the pictures can be seen in Appendix A.

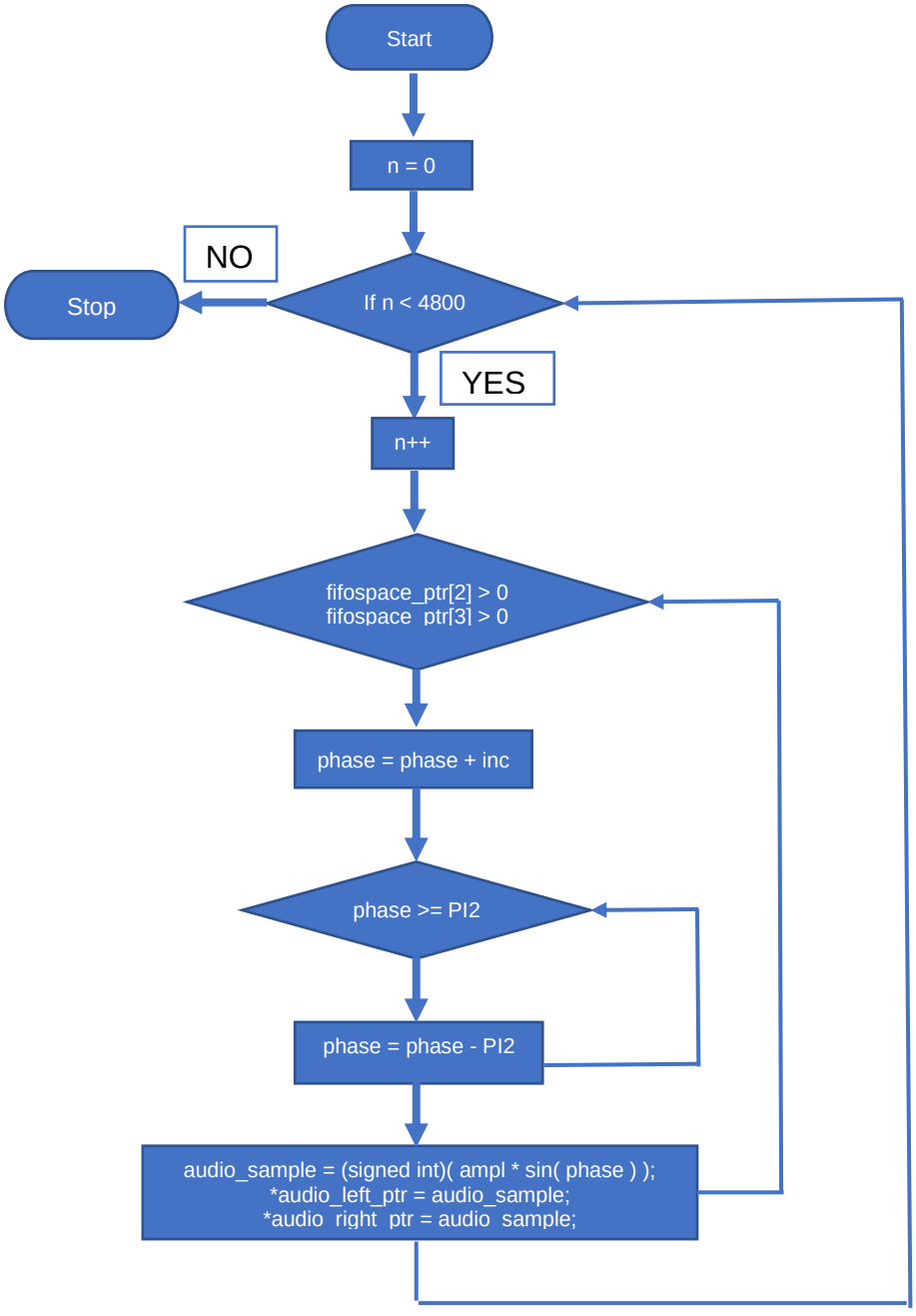
Table 6

Types of Function	No.	Function
Questions	q_1	Show the question 1
	q_1	Show the question 2
	q_1	Show the question 3
Right Answers	a_1	Uncover the first bottle for Q1
	a_5	Uncover the second bottle for Q2
	a_9	Uncover the third bottle for Q3
Wrong Answers	a_2	Uncover the first bottle for Q2
	a_3	Uncover the first bottle for Q3
	a_4	Uncover the second bottle for Q1
	a_6	Uncover the second bottle for Q3
	a_7	Uncover the third bottle for Q1
	a_8	Uncover the third bottle for Q2

Code Main Structure

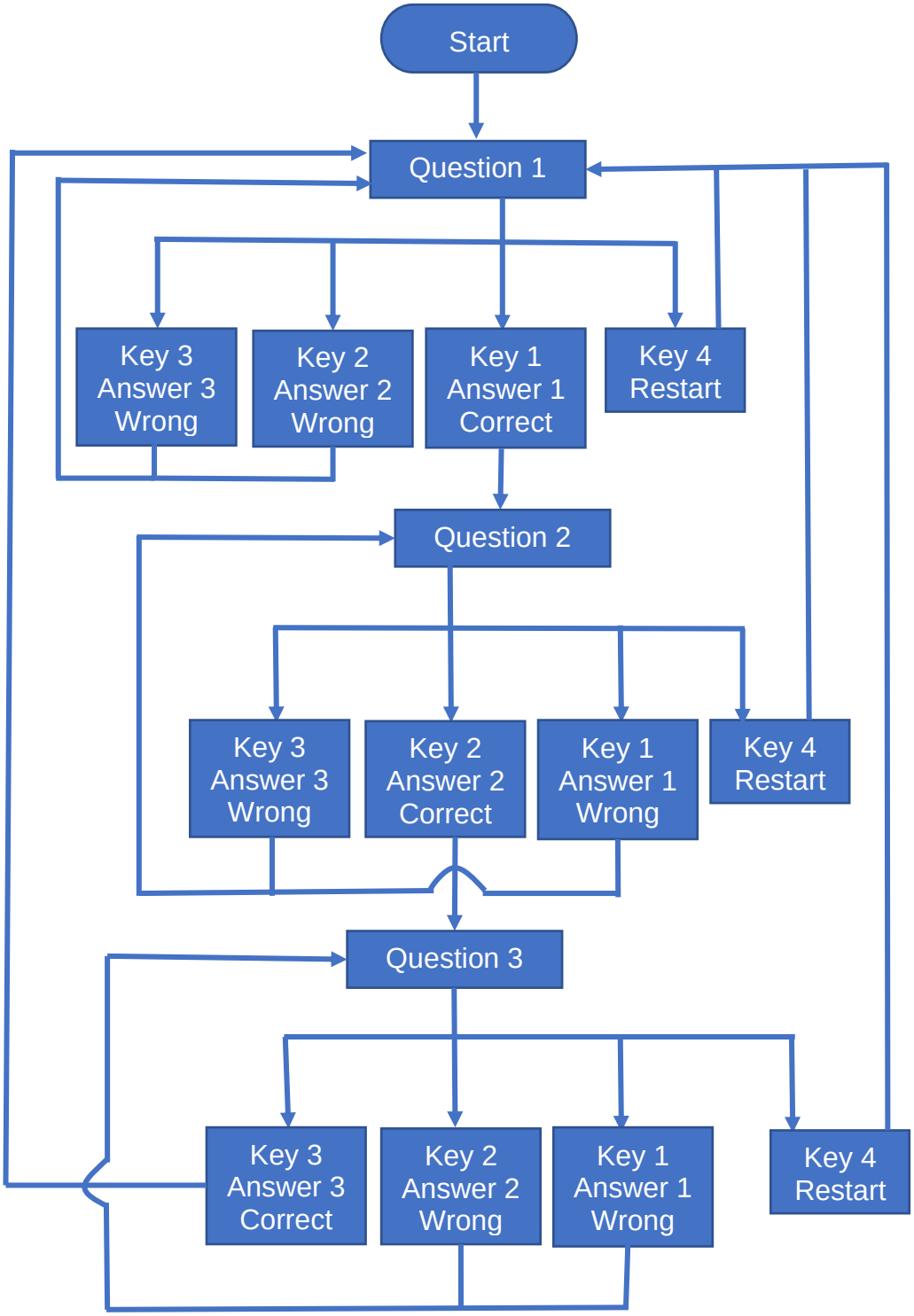
Audio

In this project, the audio system will be used to play the background sound. When the players start the game, the microphone will beep. This sound will push the players to make the decision quickly and avoid feeling bored. The initial phase is 0, the phase increments for every loop is 440.0 * PI2 / F_SAMPLE based on the desired frequency 440hz, and the amplitude is 2^23. The logic diagram is shown below [6].



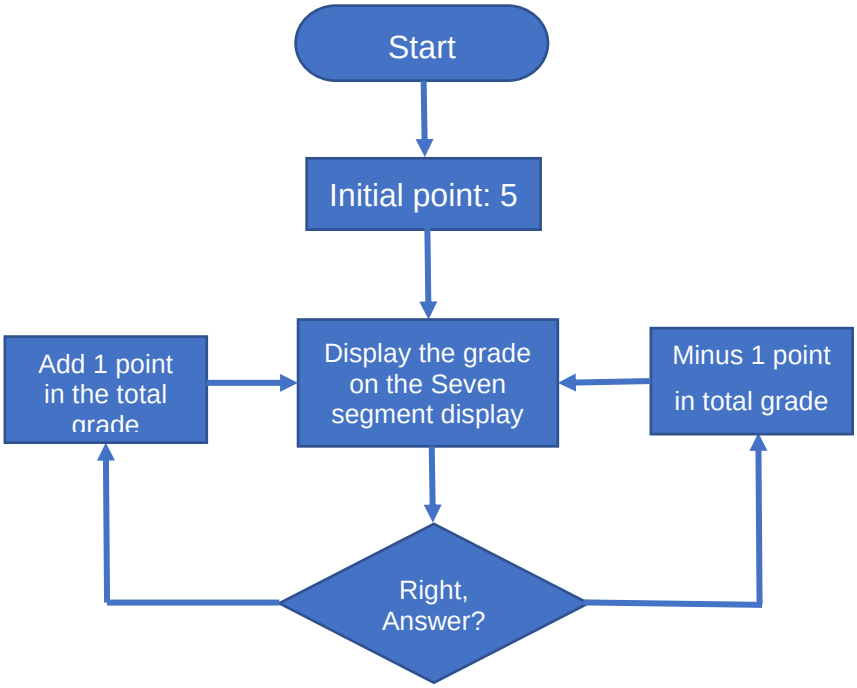
Main Algorithm

To achieve the goal, the picture on the LT24 LCD needs to be changed when the connected peripherals are activated. The logic diagram will be shown below to explain how the code works [7].



Seven Segment Display

In this project, seven segment display is used to display the grade the player gets in this game. The logic diagram is shown below.



Debugging

The DS-5 debugger is used in this project to find the errors in the code and determine where the problem is and what type the problem is. Therefore, it is easier to find and revise the error. In the debugging process, we can add the breakpoint to test the specific part of the code and check the registers and variables to identify whether the code is running correctly.

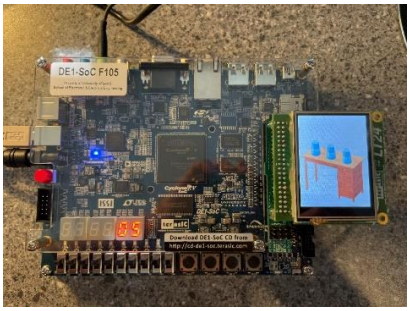
In my first version of the code, the audio did not work. However, there are not any syntax faults. It is difficult to determine where the problem is. Using the DS-5 debugger, I added the breakpoint at the end of the audio code, observed the operation process, and checked the registers and variables. Consequently, I found the code did not run the loop since the lack of a loop statement, so the code could not get into the loop. The audio can normally play after adding a 'for' statement before the audio function.

DS-5 Debugger is very useful in this project to help me determine the errors and revise them.

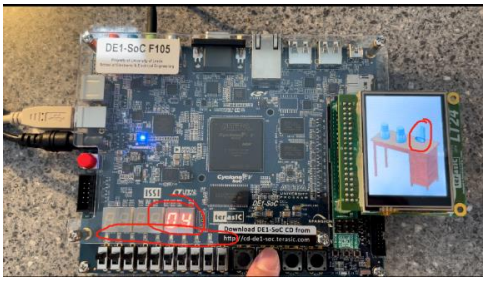
Hardware Test

To determine the feasibility of the program, a hardware test is necessary. In the following, the hardware test is shown step by step, and the test picture is shown in each step.

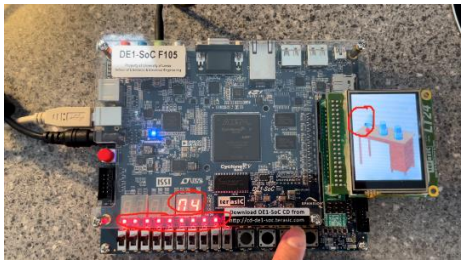
- Step 1: In this step, the initial status is checked. The LT24 LCD can display the picture well. The seven-segment display will display the initial grade of the game. The audio can play the music normally.



- Step 2: In this step, the wrong answer keys are tested. The non-ball picture will be displayed on the LT24 LCD, minus the grade.

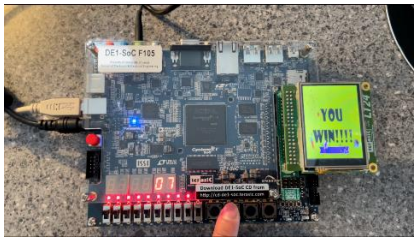


- Step 3: The correct key is tested in this step. The red ball will show on the screen, the red LED will light on, and the grade will add 1 point.



- Step 4

  The whole process of the game will be tested in this stage. After finishing the game, the victory picture will be shown on the screen.



- Step 5

  After finishing the game, the restart key can be used to return the first question.

- Step 6

  The restart key is tested whether the game can return to the first question at any time when the restart key is pressed.

Conclusion

From this project, I designed a video using not only the LT24 LCD and buttons but also the audio, red LED, and seven-segment display. I learned much new knowledge and reviewed what I had learned before. In this process, I know how to display and change the picture on the LT24 LCD and how to play the music. Besides, I had a deeper understanding of using the seven-segment and red LED, which I have done before.

Designing a game is a difficult job. We need to create the pictures and design the logic to play them sequentially. Besides, we need to know how to use the peripherals to cooperate to achieve our goal. Furthermore, it is necessary to understand how to use different drivers like watchdogs and timers.

Consequently, this project allowed me to practice what I had learned and enhanced my understanding of them.

Reference

[1] ELEC5620M: Embedded Systems Design Unit 1.1: ARM DS-5 Introduction
[2] ELEC5620M: Embedded System Design Assignment2: Stopwatch
[3] ELEC5620M: Embedded Systems Design Assignment 1: GitHub Upload
[4] ELEC5620M: Embedded Systems Design Unit 2.1: Event Timers and Task Schedulers
[5] ELEC5620M: Embedded System Design Unit 1.2: Pointers and Functions
[6] ELEC5620M: Embedded Systems Design Unit 3.1: Generating Audio
[7] ELEC5620M: Embedded Systems Design Unit 3.2: Display and Optimisation

Appendix A

Figure 1 to Figure 3

Figure 4 and Figure 5

Figure 6 and Figure 7

Figure 8

Appendix B

Main

```
/*
 * DE1-SoC LT24 Example
 *
 * Displays a test pattern on the LT24
 * LCD using the LT24 bare metal driver.
 *
 *  Created on: 09 Feb 2018
 */

#include "DE1SoC_LT24/DE1SoC_LT24.h"
#include "HPS_Watchdog/HPS_Watchdog.h"
#include "HPS_usleep/HPS_usleep.h"
#include "DE1SoC_SevenSeg/DE1SoC_SevenSeg.h"
#include "DE1SoC_WM8731/DE1SoC_WM8731.h"

//Include Floating Point Math Libraries
#include <math.h>
#include <stdlib.h>
//Include our image
#include "question.h"
#include "answer_1_0.h"
#include "answer_1_1.h"
#include "answer_2_0.h"
#include "answer_2_1.h"
#include "answer_3_0.h"
#include "answer_3_1.h"
#include "win.h"

// Select which displays to use.
#define SINGLE_DISPLAY_LOCATION 0

//Define some useful constants
#define F_SAMPLE 48000.0        //Sampling rate of WM8731 Codec (Do not change)
#define PI2      6.28318530718  //2 x Pi      (Apple or Peach?)

// Global variables
unsigned int taskID = 0;
unsigned int single_hex_display_value=5;


// This can be used to terminate the program (i.e. crash + reboot)
// if the "status" value returned from a driver doesn't match the
// "successStatus" value.
void exitOnFail(signed int status, signed int successStatus){
    if (status != successStatus) {
        exit((int)status); //Add breakpoint here to catch failure
    }
}

// Peripheral base addresses.
volatile unsigned int *key_ptr = (unsigned int *)0xFF200050;

// Store the state of the keys last time we checked.
// This allows us to determine when a key is pressed, then released.
unsigned int key_last_state = 0;

//GetPressedKeys
unsigned int getPressedKeys() {

    // Store the current state of the keys.
    unsigned int key_current_state = *key_ptr;

    // If the key was down last cycle, and is up now, mark as pressed.
    unsigned int keys_pressed = (~key_current_state) & (key_last_state);

    // Save the key state for next time, so we can compare the next state to this.
    key_last_state = key_current_state;

    // Return result.
    return keys_pressed;
}

//Define new data type for function which takes an int parameter and returns nothing.
typedef void (*TaskFunction)(int);
//Some functions for display pictures
```

```
//Functions for question

void q_1 (int id){
		//display the picture
		exitOnFail(
				LT24_copyFrameBuffer(question,0,0,240,320),
				LT24_SUCCESS); //Exit if not successful
}

void q_2 (int id){
		//display the picture
		exitOnFail(
				LT24_copyFrameBuffer(question,0,0,240,320),
				LT24_SUCCESS); //Exit if not successful
}

void q_3 (int id){
		//display the picture
		exitOnFail(
				LT24_copyFrameBuffer(question,0,0,240,320),
				LT24_SUCCESS); //Exit if not successful
}

//Functions for answers
void a_1 (int id){
		volatile unsigned int *LEDR_ptr = (unsigned int *) 0xFF200000;
		*LEDR_ptr = 0; ///Invert all LEDs.
		//display the picture
		exitOnFail(
				LT24_copyFrameBuffer(answer_1_1,0,0,240,320),
				LT24_SUCCESS); //Exit if not successful

		*LEDR_ptr = ~*LEDR_ptr; ///Invert all LEDs.
   // Display on the 7-segment display
		single_hex_display_value++;
			DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION,
					single_hex_display_value);
			// Move to the next question
			  taskID++;
}

void a_2 (int id){
		volatile unsigned int *LEDR_ptr = (unsigned int *) 0xFF200000;


		*LEDR_ptr = 0; ///Invert all LEDs.
		//display the picture
		exitOnFail(
				LT24_copyFrameBuffer(answer_1_0,0,0,240,320),
				LT24_SUCCESS); //Exit if not successful
		// Display the 7-segment display
		single_hex_display_value--;
			DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION,
					single_hex_display_value);
}

void a_3 (int id){
		volatile unsigned int *LEDR_ptr = (unsigned int *) 0xFF200000;


		*LEDR_ptr = 0; ///Invert all LEDs.
		//display the picture
		exitOnFail(
				LT24_copyFrameBuffer(answer_1_0,0,0,240,320),
				LT24_SUCCESS); //Exit if not successful
		// Display on the 7-segment display
		single_hex_display_value--;
			DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION,
					single_hex_display_value);
}

void a_4 (int id){
		volatile unsigned int *LEDR_ptr = (unsigned int *) 0xFF200000;


		*LEDR_ptr = 0; ///Invert all LEDs.
		//display the picture
		exitOnFail(
				LT24_copyFrameBuffer(answer_2_0,0,0,240,320),
				LT24_SUCCESS); //Exit if not successful
		// Display on the 7-segment display
		single_hex_display_value--;
			DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION,
```

```
                                        single_hex_display_value);
}
void a_5 (int id){
        volatile unsigned int *LEDR_ptr = (unsigned int *) 0xFF200000;


        *LEDR_ptr = 0; ///Invert all LEDs.
        //display the picture
        exitOnFail(
                        LT24_copyFrameBuffer(answer_2_1,0,0,240,320),
                        LT24_SUCCESS); //Exit if not successful
        // Display on the 7-segment display
        single_hex_display_value++;
                DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION,
                                        single_hex_display_value);

                  *LEDR_ptr = ~*LEDR_ptr; ///Invert all LEDs.
                  // Move to the next question
                 taskID++;
}

void a_6 (int id){
        volatile unsigned int *LEDR_ptr = (unsigned int *) 0xFF200000;


        *LEDR_ptr = 0; ///Invert all LEDs.
        //display the picture
        exitOnFail(
                        LT24_copyFrameBuffer(answer_2_0,0,0,240,320),
                        LT24_SUCCESS); //Exit if not successful
        // Display on the 7-segment display
        single_hex_display_value--;
                DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION,
                                        single_hex_display_value);
}

void a_7 (int id){
        volatile unsigned int *LEDR_ptr = (unsigned int *) 0xFF200000;


        *LEDR_ptr = 0; ///Invert all LEDs.
        //display the picture
        exitOnFail(
                        LT24_copyFrameBuffer(answer_3_0,0,0,240,320),
                        LT24_SUCCESS); //Exit if not successful
        // Display on the 7-segment display
        single_hex_display_value--;
                DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION,
                                        single_hex_display_value);
}

void a_8 (int id){
        volatile unsigned int *LEDR_ptr = (unsigned int *) 0xFF200000;


        *LEDR_ptr = 0; ///Invert all LEDs.
        //display the picture
        exitOnFail(
                        LT24_copyFrameBuffer(answer_3_0,0,0,240,320),
                        LT24_SUCCESS); //Exit if not successful
        // Display on the 7-segment display
        single_hex_display_value--;
                DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION,
                                        single_hex_display_value);
}

void a_9 (int id){
        volatile unsigned int *LEDR_ptr = (unsigned int *) 0xFF200000;


        *LEDR_ptr = 0; ///Invert all LEDs.
        //display the picture
        exitOnFail(
                        LT24_copyFrameBuffer(answer_3_1,0,0,240,320),
                        LT24_SUCCESS); //Exit if not successful

        *LEDR_ptr = ~*LEDR_ptr; ///Invert all LEDs.
        // Display on the 7-segment display
        single_hex_display_value++;
                DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION,
                                        single_hex_display_value);
                // Move to the next question
                  taskID++;
```

```
}
//
// Main Function
// =============
int main(void) {
		//define the variables
		unsigned int keys_pressed = 0; // initial the keys buttons
		const unsigned int taskCount = 3;  // Set the number of question
		unsigned int n;

		//Pointers for audio
   volatile unsigned char* fifospace_ptr;
   volatile unsigned int*  audio_left_ptr;
   volatile unsigned int*  audio_right_ptr;

   //Variables
   //Phase Accumulator
   double phase = 0.0;  // Phase accumulator
   double inc   = 0.0;  // Phase increment
   double ampl  = 0.0;  // Tone amplitude (i.e. volume)
   signed int audio_sample = 0;

   // Set the pointer for the function
		TaskFunction questionFunctions[taskCount] = {&q_1,&q_2,&q_3};
		TaskFunction answer1Functions[taskCount] = {&a_1,&a_2,&a_3};
		TaskFunction answer2Functions[taskCount] = {&a_4,&a_5,&a_6};
		TaskFunction answer3Functions[taskCount] = {&a_7,&a_8,&a_9};

		/* Pointers to peripherals */
		   // ARM A9 Private Timer Load
		   volatile unsigned int *private_timer_load      = (unsigned int *) 0xFFFEC600;
		   // ARM A9 Private Timer Value
		   volatile unsigned int *private_timer_value     = (unsigned int *) 0xFFFEC604;
		   // ARM A9 Private Timer Control
		   volatile unsigned int *private_timer_control   = (unsigned int *) 0xFFFEC608;
		   // ARM A9 Private Timer Interrupt
		   volatile unsigned int *private_timer_interrupt = (unsigned int *) 0xFFFEC60C;
		   /* Local Variables */
		   unsigned int lastBlinkTimerValue = *private_timer_value;
		   const unsigned int blinkPeriod = 50000;

		   // Set the "Load" value of the timer to max value:
		     *private_timer_load      = 0xFFFFFFFF;
		     // Set the "Prescaler" value to 0, Enable the timer (E = 1), Set Automatic reload
		     // on overflow (A = 1), and disable ISR (I = 0)
		     *private_timer_control   = (0 << 8) | (0 << 2) | (1 << 1) | (1 << 0);

		     DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION, single_hex_display_value);

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

   //Initialise the Audio Codec.
   exitOnFail(
         WM8731_initialise(0xFF203040),  //Initialise Audio Codec
         WM8731_SUCCESS);                //Exit if not successful

   //Clear both FIFOs
   WM8731_clearFIFO(true,true);

   //Grab the FIFO Space and Audio Channel Pointers
   fifospace_ptr = WM8731_getFIFOSpacePtr();
   audio_left_ptr = WM8731_getLeftFIFOPtr();
   audio_right_ptr = WM8731_getRightFIFOPtr();

   //Initialise Phase Accumulator
   inc   = 440.0 * PI2 / F_SAMPLE;  // Calculate the phase increment based on desired frequency - e.g. 440Hz
   ampl  = 8388608.0;               // Pick desired amplitude (e.g. 2^23). WARNING: If too high = deafening!
   phase = 0.0;

   while(1){

		// Read the current time
      unsigned int currentTimerValue = *private_timer_value;
      keys_pressed = getPressedKeys();

    // Set the loop for the audio play
    if (taskCount > 0){
		for (n=0; n<48000; n++){
      if ((fifospace_ptr[2] > 0) && (fifospace_ptr[3] > 0)){
```

```
                    //If there is space in the write FIFO for both channels:
                //Increment the phase
                phase = phase + inc;
                //Ensure phase is wrapped to range 0 to 2Pi (range of sin function)
                while (phase >= PI2) {
                    phase = phase - PI2;
                    }
                // Calculate next sample of the output tone.
                audio_sample = (signed int)( ampl * sin( phase ) );
                // Output tone to left and right channels.
                *audio_left_ptr = audio_sample;
                *audio_right_ptr = audio_sample;
            }
             }
      }

    // Check if it is time to blink
    if ((lastBlinkTimerValue - currentTimerValue) >= blinkPeriod) {
        // Check whether the question is finished
         if (taskID<3){
          // When the task is ready to be run, call function, taskID is param
                    questionFunctions[taskID](taskID);
                    //Check whether the key 1 is pressed
                    if (keys_pressed & 0x1){
                            // When the task is ready to be run, call function, taskID is param
                            answer1Functions[taskID](taskID);
                    }
                    //Check whether the key 2 is pressed
                    else if (keys_pressed & 0x2){
                            // When the task is ready to be run, call function, taskID is param
                            answer2Functions[taskID](taskID);
                    }
                    //Check whether the key 3 is pressed
                    else if (keys_pressed & 0x4){
                            // When the task is ready to be run, call function, taskID is param
                            answer3Functions[taskID](taskID);
                    }
                    //Check whether the key 4 is pressed
                    else if (keys_pressed & 0x8){
                            // Initialize the value
                            taskID = 0;
                            single_hex_display_value = 5;
              DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION,
                             single_hex_display_value);
                    }
        }
        // if finish the question
        else {
           //Display the victory picture
              exitOnFail(
                                    LT24_copyFrameBuffer(win,0,0,240,320),
                                    LT24_SUCCESS); //Exit if not successful

           // Check whether the key 4 is pressed
              if (keys_pressed & 0x8){
                        // Initialize the value
                        taskID = 0;
                single_hex_display_value = 5;
                DE1SoC_SevenSeg_SetSingle(SINGLE_DISPLAY_LOCATION,
                             single_hex_display_value);
              }
        }

         // To avoid accumulation errors, we make sure to mark the last time
         // the task was run as when we expected to run it. Counter is going
         // down, so subtract the interval from the last time.
         lastBlinkTimerValue = lastBlinkTimerValue - blinkPeriod;
    }

    // --- You can have many other events here by giving each an if statement
    // --- and its own "last#TimerValue" and "#Period" variables, then using the
    // --- same structure as above.
    // Next, make sure we clear the private timer interrupt flag if it is set
    if (*private_timer_interrupt & 0x1) {
             // If the timer interrupt flag is set, clear the flag
             *private_timer_interrupt = 0x1;
         }
    // Finally, reset the watchdog timer.
       HPS_ResetWatchdog();
    }
}
```

Appendix C

DE1SoC_SevenSeg.h

```
/*
 * DE1SoC_SevenSeg.h
 *
 *  Created on: 12 Feb 2021
 *      Author: Harry Clegg
 */

/*****************************************************************************
 * DO NOT MODIFY THIS FILE
 * YOU CAN COMPLETE THE ASSIGNMENT WITHOUT ANY CHANGES TO THIS FILE
 *****************************************************************************/

#ifndef DE1SoC_SEVENSEG_H_
#define DE1SoC_SEVENSEG_H_

/**
 * DE1SoC_SevenSeg_Write
 *
 * Low level set function to send a byte value to a selected display.
 *
 * Inputs:
 *                  display:        index of the display to update (0-5)
 *                  value:                  byte to set display to.
 */
void DE1SoC_SevenSeg_Write(unsigned int display, unsigned char value);

/**
 * DE1SoC_SevenSeg_SetSingle
 *
 * Set a single seven segment display from a hexadecimal (0x0-0xF) value.
 *
 * Inputs:
 *                  display:        index of the display to update (0-5)
 *                  value:                  value to assign to the display (0x0-0xF)
 */
void DE1SoC_SevenSeg_SetSingle(unsigned int display, unsigned int value);

/**
 * DE1SoC_SevenSeg_SetDoubleHex
 *
 * Set a pair of seven segment displays from a hexadecimal (0x00-0xFF) value.
 *
 * Inputs:
 *                  display:        index of the pair of displays to update (0-4)
 *                  value:                  value to assign to the display (0x00-0xFF)
 */
void DE1SoC_SevenSeg_SetDoubleHex(unsigned int display, unsigned int value);

/**
 * DE1SoC_SevenSeg_SetDoubleDec
 *
 * Set a pair of seven segment displays from a decimal (00-99) value.
 *
 * Inputs:
 *                  display:        index of the pair of displays to update (0-4)
 *                  value:                  value to assign to the display (00-99)
 */
void DE1SoC_SevenSeg_SetDoubleDec(unsigned int display, unsigned int value);

#endif /* DE1SoC_SEVENSEG_H_ */
```

Appendix D

DE1SoC_SevenSeg.c

```
/*
 * DE1SoC_SevenSeg.c
 *
 *  Created on: 12 Feb 2021
 *      Author: Harry Clegg
 *                      You!
 */

#include "DE1SoC_SevenSeg.h"

// ToDo: Add the base addresses of the seven segment display peripherals.

// ToDo: Ad the base addresses of the seven segment display peripherals.

// Create the 2 peripheral address for 6 seven-segments displays
// peripheral_1 is the address for HEX0, HEX1, HEX2 and HEX3, and peripheral_2 is for HEX4 and HEX5
volatile unsigned char *sevenseg_base_lo_ptr = (unsigned char *) 0xFF200020;
volatile unsigned char *sevenseg_base_hi_ptr = (unsigned char *) 0xFF200030;

// There are four HEX displays attached to the low (first) address.
#define SEVENSEG_N_DISPLAYS_LO 4

// There are two HEX displays attached to the high (second) address.
#define SEVENSEG_N_DISPLAYS_HI 2

void DE1SoC_SevenSeg_Write(unsigned int display, unsigned char value) {
    // Select between the two addresses so that the higher level functions
    // have a seamless interface.

    if (display < SEVENSEG_N_DISPLAYS_LO) {
        // If we are targeting a low address, use byte addressing to access
        // directly.
        sevenseg_base_lo_ptr[display] = value;
    } else {
        // If we are targeting a high address, shift down so byte addressing
        // works.
        display = display - SEVENSEG_N_DISPLAYS_LO;
        sevenseg_base_hi_ptr[display] = value;
    }
}

void DE1SoC_SevenSeg_SetSingle(unsigned int display, unsigned int value) {

        const unsigned char segCode [16]= {0x3F, 0x06, 0x58, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7, 0x6F, 0x77, 0x7C,
0x59, 0x3E, 0x79, 0x71};
        if (value < 17){
                DE1SoC_SevenSeg_Write(display, segCode[value]) ;
                DE1SoC_SevenSeg_Write(display+1, segCode[0]) ;
            }
                else{
                    DE1SoC_SevenSeg_Write(display,0x40) ;
                }
    // ToDo: Write the code for driving a single seven segment display here.
    // Your function should turn a real value 0-F into the correctly encoded
    // bits to enable the correct segments on the seven segment display to
    // illuminate. Use the DE1SoC_SevenSeg_Write function you created earlier
    // to set the bits of the display.
}
```

```
void DE1SoC_SevenSeg_SetDoubleDec(unsigned int display, unsigned int value) {

	// create a variable about the number of characters of 0-9
			int range = 11;
			// create an array about the hexadecimal of 0-9
			const unsigned char segCode [10]= {0x3F, 0x06, 0x58, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7,
0x6F};

				if (value < 11){
					// if the value is under 10, it will be display in ones space
					DE1SoC_SevenSeg_Write(display, segCode[value]);
					DE1SoC_SevenSeg_Write(display+1, segCode[0]);
			}

				else{
			// otherwise, tens space will also be used to display

					//calculate which character will be displayed in ones space
					DE1SoC_SevenSeg_Write(display, segCode[value % range]);

					//calculate which character will be display in tens space
					DE1SoC_SevenSeg_Write(display+1, segCode[value / range]);

			}

	// ToDo: Write the code for setting a pair of seven segment displays
   // here. Good coding practice suggests your solution should call
   // DE1SoC_SevenSeg_SetSingle() twice.
   // This function should show the first digit of a DECIMAL number on
   // the specified 'display', and the second digit on the display to
   // the left of the specified display.

   /** Some examples:
    *
    *        input | output | HEX(N+1) | HEX(N)
    *    ----- | ------ | -------- | ------
    *        5 |     05 |        0 |      5
    *       30 |     30 |        3 |      0
    *     0x90 |     96 |        9 |      6
    */
}
```