SANTA CLARA UNIVERSITY
Electrical and Computer Engineering Department

Real-Time Embedded Systems - ECEN 121
Lab 9
Dylan Thornburg & Kai Hoshide

USB Music Player

Andrew Wolfe

Prelab:

Review this entire lab.

Review the code in main.c for the first part of the lab (Lab9p1.zip from the Lab Handouts directory – in the Src subdirectory.

How many cycles of the sine wave will be placed into the buffer audiobuffer[] ? Show the math.
We have 16,000 samples in the buffer, and know that 40 samples makes up one wave cycle. Therefore we have 16,000 samples / 40 samples per cycle. After doing this division and canceling units, we get 400 total sin wave cycles.

Using the schematic below, the output of port PA5 will range from 0-3V and in practice will be an AC signal with a peak-to-peak voltage of 3V. How can you calculate the peak-to-peak voltage at Tip when powered, stereo speakers are plugged into the audio adapter?
Just do Voltage division.
0 to 3V * (10k/(5.1k+10k)= 0 to 1.987V or ~2V. Peak to peak is 1.987V or ~2V.

Lab Overview:

In this lab you will program the system to read a music file from a USB drive and play the music on the DAC. The 3.5mm connector on the board connects to an audio-grade stereo DAC. That DAC is a bit too complicated to program for the time we have, so like ELEN 120, we will connect a speaker to the 12-bit DAC with an adapter.

The key challenge here is that the music files are up to 18 MB of data. Your board has only 128KB of RAM. That means we can't read the entire music file into memory before we play it.

We will use a double-buffering scheme. The music files are 16-bit unsigned values sampled at 16KHz. We will set up two buffers, each of which has 16000 entries and can thus hold 1 second of music. You will fill up one buffer by reading data from the USB drive. Once it is full, you can play the music in that buffer and while it is playing you fill up the second buffer from the USB drive. Then you swap. This keeps happening until you get to the end of the file.

In part 1 of the lab, we will not use USB. We will fill up a 1 second buffer with a sine wave (at 400Hz) and you will play it back repeatedly.

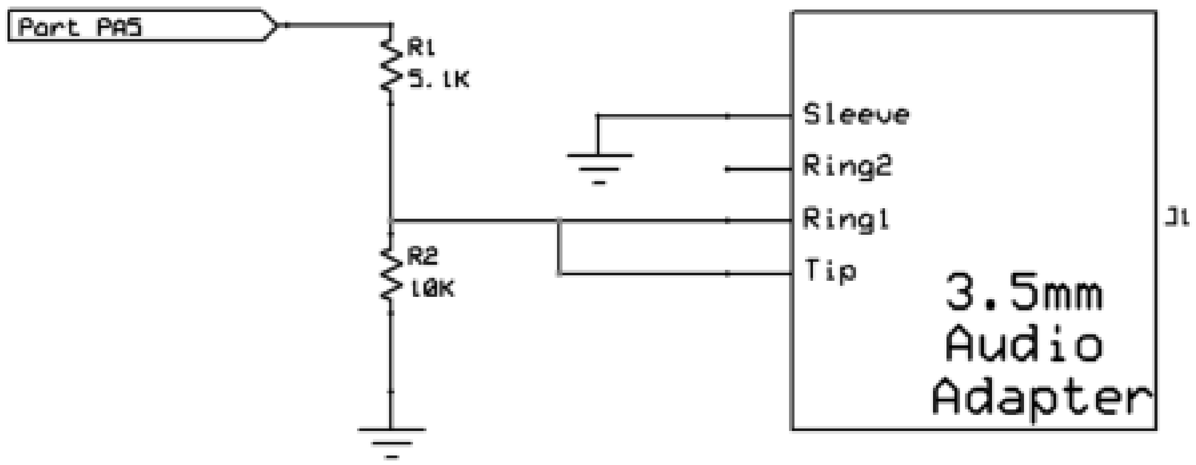
In part 2 of the lab, you must implement the double buffering scheme.

In part 3 of the lab, you add a display to show how long the song has been

playing. In part 4 of the lab, you add the remote control.

Preparation:

Connect the speaker adapter to your board as shown in the following schematic:



Connect the speakers to the 3.5mm connector and use a spare USB port to power the speakers. You can set the speaker volume near the middle to start using the dial. You can then turn it up to maximum later when you hear sound.

Lab Project 1:

In part 1 of this project, you will play a 400Hz sine wave on your speakers from the DAC.

You can begin by copying the project file Lab9p1.zip from the Lab Handouts directory and unzipping it into your MX directory. Like Labs 7 and 8, you will need to open it in CubeMX and generate code to match the library paths to your machine. This is an incomplete project that should compile but will not do anything.

When you compile this project, you may get a Warning.

```
Lab9p1\Lab9p1.axf: Warning: L6989W: Could not apply patch sdcomp-29491-629360 to
instruction VPOP {d8-d12} at offset 0x24, instruction is within an IT block and is
not the last instruction for sin_i_x.o(i.____kernel_sin$lsc).
```

This is not a problem. It is a known bug in the STM32L4 libraries that will not be a problem on our board.

2

My program has a buffer called audiobuffer[] that can hold 16000 16-bit, unsigned sound samples. It is intended to be played back at 16KHz. I fill it with a 400Hz sine wave in main().

I have configured timer 6 to interrupt 16K times per second. I have initialized timer 6 and DAC1 Channel 2, which outputs on Port A Pin 5.

I have also provided you with a function write_DAC1Ch2(uint16_t dacval, int volume) that writes an unsigned 12-bit value to the DAC. You need to specify the volume, which I have predefined as volume.

Each time the Timer 6 interrupt happens you should:

- Write the next value in audiobuffer[] to the DAC.
- If you are at the end of audiobuffer[], start again at the beginning.

I have indicated where you should put your code:

```
/**************************************************/ /*
      Put your code here to produce a 400Hz Sine Wave */
 /**************************************************/
```

You may need to create additional variables.

If you do this correctly, your speaker will play an annoying 400Hz tone continuously. You can check the frequency with the scope.

To demo – Play the tone for the TA. Turn in your modified interrupt handler and any other code you changed with the report.

CODE:

```
void TIM6_DAC_IRQHandler(void)
{
 /* USER CODE BEGIN TIM6_DAC_IRQn 0 */

 /* USER CODE END TIM6_DAC_IRQn 0 */
 HAL_TIM_IRQHandler(&htim6);
 HAL_DAC_IRQHandler(&hdac1);
 /* USER CODE BEGIN TIM6_DAC_IRQn 1 */

 /**************************************************/
```

```
    write_DAC1Ch2(audiobuffer[i], DFLT_VOLUME);
    i=(i+1)% ABUF_SIZE;
/* Put your code here to produce a 400Hz Sine Wave */
 /**************************************************/


 /* USER CODE END TIM6_DAC_IRQn 1 */
}
```

Lab Project 2:

For Part 2 of the project, you will play a song from the USB flash drive. Get a numbered USB drive from the song-file USB drive bin that the TA has and connect it to your board. Don't mix it up with the USB drive from Lab 8.

You can begin by copying the project file Lab9p2.zip from the Lab Handouts directory and unzipping it into your MX directory. Like Labs 7 and 8, you will need to open it in CubeMX and generate code to match the library paths to your machine. This is an incomplete project that should compile but will not do anything.

To get continuous playback, we will use 2 audio buffers. The idea is that the first buffer is filled with music samples from the USB drive. Once it is full, the interrupt handler can start playing from that buffer. While music is being played from the first buffer, the second buffer is filled with data. When the playback interrupt finishes playing the first buffer, it marks it as empty and starts playing the second buffer. The empty buffer will be filled by the USB software and so forth until the end of the song is reached.

Some rules:

- Don't start playing a buffer until it is full.
- Don't play past the end of the song.

I have set up the file-reading code to generate a tone if you go past the end of the song. You cannot remove this code.

Since you are already USB experts, I have written the USB software for you. You need to write the playback software.

This is not a lot of software to write. (My implementation took about 11 lines of code). But do not think that it is trivial. In order to do this, you need to understand the code I have already written and the algorithm for buffer management and playback. You may need to add additional variables.

Play the song from the file songfile.raw for the TA. You need to make sure the entire song plays without any glitches then stops.

Once you have everything playing and stopping at the right times, tell the professor it is time for your team to show off. He will show you the next steps.

To demo - play the song for the TA. TikTok dances optional. Turn in your modified interrupt handler and any other code you changed with the report.

CODE:

```
/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */




// IR Remote Variables
int irval;
int startCount;
int i = 0;
int j = 0;
int flag = 0;
int k = 0;
int count = 0;
/* USER CODE END PV */

void TIM6_DAC_IRQHandler(void)
{
  /* USER CODE BEGIN TIM6_DAC_IRQn 0 */

  /* USER CODE END TIM6_DAC_IRQn 0 */
```

```
 HAL_TIM_IRQHandler(&htim6);
 HAL_DAC_IRQHandler(&hdac1);
 /* USER CODE BEGIN TIM6_DAC_IRQn 1 */

 /****************************************************/
/* Put your code here to play a song                                                    */
 if(lastbuffer == 0)
{
        if (flag == 0)
        {
                        if(abuf_full[0] == true)
                        {
                                // Switch to the next buffer
                                 write_DAC1Ch2(audiobuffer[k][i], DFLT_VOLUME);
                                 i=(i+1)%ABUF_SIZE;
                        }
                        if(i == 0)
                        {
                                        abuf_full[0] = false;
                                        flag = 1;
                                        k++;
                        }
        }
        if (flag == 1)
        {
                        if(abuf_full[1] == true)
                        {
                                // Switch to the next buffer
                                 write_DAC1Ch2(audiobuffer[k][j], DFLT_VOLUME);
                                 j=(j+1)%ABUF_SIZE;
                        }
                        if(j == 0)
                        {
                                        abuf_full[1] = false;
                                        flag = 0;
                                        k--;
                        }
        }
}
else if(flag !=2)
{
        write_DAC1Ch2((uint16_t) audiobuffer[k][count], DFLT_VOLUME);
        count = (count+1)%ABUF_SIZE;
        if((count*2) > (lastbuffer-1))
        {
                flag = 2;
        }
}
 /****************************************************/


 /* USER CODE END TIM6_DAC_IRQn 1 */
}
```

Lab Project 3:

Make a copy of Lab9p2 and call it Lab9p3. Add in an LCD display that counts how long your song has been playing in minutes and seconds. Remember that the RTC needs to be enabled.

Demo.

MAIN.C CODE:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under Ultimate Liberty license
  * SLA0044, the "License"; You may not use this file except in compliance with
  * the License. You may obtain a copy of the License at:
  *                             www.st.com/SLA0044
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "fatfs.h"
#include "usb_host.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdbool.h>
#include "song16.h"
#include "stm32l476g_discovery.h"
#include "stm32l476g_discovery_glass_lcd.h"
#include "display.h"


/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */


/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
DAC_HandleTypeDef hdac1;

LCD_HandleTypeDef hlcd;

RTC_HandleTypeDef hrtc;

TIM_HandleTypeDef htim6;
TIM_HandleTypeDef htim16;

/* USER CODE BEGIN PV */

extern ApplicationTypeDef Appli_state;


int16_t audiobuffer[NUM_ABUF][ABUF_SIZE];		/* File read buffer for audio */
volatile bool abuf_full[NUM_ABUF];
bool file_open = false;
int fill_buf = 0;
int lastbuffer = 0;
volatile uint8_t volume = DFLT_VOLUME;			// Volume range is 0 to 255
uint8_t noisehigh[8] = {0xff, 0xdd, 0xbb, 0x99, 0x77, 0x55, 0x33, 0x11};

char *rfilename = "songfile.raw";


FIL MyFile;			/* File object */



/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DAC1_Init(void);
static void MX_TIM6_Init(void);
static void MX_RTC_Init(void);
static void MX_LCD_Init(void);
static void MX_TIM16_Init(void);
void MX_USB_HOST_Process(void);
```

```
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */




static void FS_FileOpen(void) {

		/* Register the file system object to the FatFs module */
  if(f_mount(&USBHFatFS, (TCHAR const*)USBHPath, 0) == FR_OK) {
			/* Open the text file object with read access */
			if(f_open(&MyFile, rfilename, FA_READ) == FR_OK) {
				HAL_GPIO_WritePin(LD_G_GPIO_Port, LD_G_Pin,GPIO_PIN_SET);
				file_open = true;
				return;
			}
		}
				HAL_GPIO_WritePin(LD_R_GPIO_Port, LD_R_Pin,GPIO_PIN_SET);
}


static void FS_FileClose(void) {
		/* Close the open text file */
		f_close(&MyFile);
		file_open = false;
}


static void FS_FileRead(uint8_t *abuffer) {
		FRESULT res;                          /* FatFs function common result code */
  uint32_t bytesread;                  /* File write/read counts */


						/* Read data from the 16-bit raw file if the next buffer is free */
		res = f_read(&MyFile, abuffer, ABUF_BYTE_SIZE, (void *)&bytesread);
		if((bytesread < ABUF_BYTE_SIZE) && (res == FR_OK)) {
			lastbuffer = bytesread;
			for (int i = bytesread/2; i<ABUF_BYTE_SIZE/2; i++) {
				abuffer[i+i+1] = noisehigh[i%8];
				abuffer[i+i] = 0;
			}
		}
}




void write_DAC1Ch2(int16_t dacval, uint8_t volume) {
		int tempval;
		tempval = (((int) dacval * (int) volume) + 8388608) >> 12;
		hdac1.Instance->DHR12R2 = tempval;
}





/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USB_HOST_Init();
  MX_FATFS_Init();
  MX_DAC1_Init();
  MX_TIM6_Init();
  MX_RTC_Init();
  MX_LCD_Init();
  MX_TIM16_Init();
  /* USER CODE BEGIN 2 */

		/* initialize data structures */

		for (int i=0; i<NUM_ABUF; i++) {
			abuf_full[i] = false;
		}
		/* start devices */
  HAL_DAC_Start(&hdac1, DAC1_CHANNEL_2);
          HAL_TIM_Base_Start_IT(&htim6);                    /* Start Timer 6 at 16KHz to run DAC */
          HAL_TIM_Base_Start_IT(&htim16);
          BSP_LCD_GLASS_Init();
          BSP_LCD_GLASS_ClearBar(0xFFFFFFFF);
          BSP_LCD_GLASS_Clear();
          uint8_t x = 48;//set up all zeroes
          BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_OFF , 5);
          BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_OFF , 4);
          BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_ON , 3);
          BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_OFF , 2);


  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
    MX_USB_HOST_Process();

    /* USER CODE BEGIN 3 */
    /* Mass Storage Application State Machine */
    switch(Appli_state) {
      case APPLICATION_READY:
        if (!file_open) FS_FileOpen();
                                    if (file_open && (lastbuffer == 0)) {
                                                if (!abuf_full[fill_buf]) {
                                                      FS_FileRead((uint8_t *) audiobuffer[fill_buf]);
                                                      abuf_full[fill_buf++] = true;
                                                      fill_buf = fill_buf % NUM_ABUF;
                                                }
                                          }
                                          if (file_open && (lastbuffer != 0)) {
                                                FS_FileClose();
          Appli_state = APPLICATION_IDLE;
                                          }
        break;

      case APPLICATION_IDLE:
        default:
        break;
      }




                        }
  /* USER CODE END 3 */
}
```

STM32L4XX_IT.C CODE:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file    stm32l4xx_it.c
  * @brief   Interrupt Service Routines.
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under Ultimate Liberty license
  * SLA0044, the "License"; You may not use this file except in compliance with
  * the License. You may obtain a copy of the License at:
  *                             www.st.com/SLA0044
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32l4xx_it.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdbool.h>
#include "song16.h"
#include "fatfs.h"
#include "stm32l476g_discovery.h"
#include "stm32l476g_discovery_glass_lcd.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN TD */

/* USER CODE END TD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */
```

```
// IR Remote Variables
int irval;
int startCount;
int i = 0;
int j = 0;
int flag = 0;
int k = 0;
int count = 0;
extern volatile int seccount;
volatile int tenseccount = 0;
volatile int tenmincount = 0;
extern volatile int mincount;
extern volatile int tenth;
/* USER CODE END PV */


void TIM1_UP_TIM16_IRQHandler(void)
{
  /* USER CODE BEGIN TIM1_UP_TIM16_IRQn 0 */

  /* USER CODE END TIM1_UP_TIM16_IRQn 0 */
  HAL_TIM_IRQHandler(&htim16);
  /* USER CODE BEGIN TIM1_UP_TIM16_IRQn 1 */
		if(flag !=2)
		{
			uint8_t ch;
			uint8_t ch2;
			uint8_t ch3;
			uint8_t ch4;
			tenth++;
			if(tenth > 9)
			{
				seccount++;
				tenth=0;
				if (seccount> 9)
				{
					seccount = 0;
					tenseccount++;
					if(tenseccount > 5)
						{
							tenseccount = 0;
							mincount++;
							if (mincount > 9)
							{
								mincount = 0;
								tenmincount++;
								if (tenmincount > 5)
								{
									mincount = 0;
									tenmincount = 0;
								}
								ch4 = tenmincount + 48;
								BSP_LCD_GLASS_DisplayChar(&ch4, POINT_OFF,
DOUBLEPOINT_OFF , 2);
```

```
                                                                        }
                                                                        ch3 = mincount + 48;
                                                                        BSP_LCD_GLASS_DisplayChar(&ch3, POINT_OFF,
DOUBLEPOINT_ON , 3);
                                                        }
                                                        ch2 = tenseccount + 48;
                                                        BSP_LCD_GLASS_DisplayChar(&ch2, POINT_OFF, DOUBLEPOINT_OFF ,4
);
                                        }
                                        ch = seccount + 48;
                                        BSP_LCD_GLASS_DisplayChar(&ch, POINT_OFF, DOUBLEPOINT_OFF ,5 );
                                }
                        }
  /* USER CODE END TIM1_UP_TIM16_IRQn 1 */
}
```

Lab Project 4:

Add in an IR remote control. The center button should pause playback (and, of course pause the playback count.) If you press it again, it should restart from the pause point. The up and down button should change the volume (by the amount defined by VOLUME_INCREMENT.) It should do the appropriate thing when it reaches minimum or maximum volume. Finally, the back button should restart the song (and the timer). This takes quite a bit of thought and may require some research.

Optionally – use the bars on the right side of the display to indicate the volume (within the limited resolution available)

Demo as much as you can get working by the end of Week 10.

STM32L4XX_IT.C CODE:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file    stm32l4xx_it.c
  * @brief   Interrupt Service Routines.
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under Ultimate Liberty license
  * SLA0044, the "License"; You may not use this file except in compliance with
  * the License. You may obtain a copy of the License at:
  *                             www.st.com/SLA0044
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
```

```
#include "stm32l4xx_it.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdbool.h>
#include "song16.h"
#include "fatfs.h"
#include "stm32l476g_discovery.h"
#include "stm32l476g_discovery_glass_lcd.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN TD */

/* USER CODE END TD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
/* USER CODE BEGIN PV */



// IR Remote Variables
int irval;
int startCount;
int i = 0;
int j = 0;
int flag = 0;
int k = 0;
int count = 0;
extern volatile int seccount;
volatile int tenseccount = 0;
volatile int tenmincount = 0;
extern volatile int mincount;
extern volatile int tenth;
extern unsigned int irdat[];
uint32_t x;
int IrCount = 0;
int data = 0;
int IrFlag =0;
int volume2 = DFLT_VOLUME;
extern int ch0;
/* USER CODE END PV */
```

```
void TIM7_IRQHandler(void)
{
  /* USER CODE BEGIN TIM7_IRQn 0 */

  /* USER CODE END TIM7_IRQn 0 */
  HAL_TIM_IRQHandler(&htim7);
  /* USER CODE BEGIN TIM7_IRQn 1 */
		x = HAL_GPIO_ReadPin(IR_IN_GPIO_Port, IR_IN_Pin);
		if(IrFlag == 0)
		{
			if(x == 0 || data == 1)
			{
				data = 1;
				if(x==0)
				{
					irdat[IrCount] = 1;
					IrCount = (IrCount +1) %SAMPLE_COUNT;
				}
				else
				{
					irdat[IrCount]=0;
					IrCount= (IrCount +1) %SAMPLE_COUNT;
				}
				if(IrCount == SAMPLE_COUNT-1)
				{
					IrFlag =1;
					data =0;
				}
			}
		}
  /* USER CODE END TIM7_IRQn 1 */
}
```

MAIN.C CODE:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under Ultimate Liberty license
  * SLA0044, the "License"; You may not use this file except in compliance with
  * the License. You may obtain a copy of the License at:
  *                             www.st.com/SLA0044
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "fatfs.h"
#include "usb_host.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdbool.h>
#include "song16.h"
#include "stm32l476g_discovery.h"
#include "stm32l476g_discovery_glass_lcd.h"
#include "display.h"


/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */


/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
DAC_HandleTypeDef hdac1;

LCD_HandleTypeDef hlcd;

RTC_HandleTypeDef hrtc;

TIM_HandleTypeDef htim6;
TIM_HandleTypeDef htim7;
TIM_HandleTypeDef htim16;

/* USER CODE BEGIN PV */

extern ApplicationTypeDef Appli_state;


int16_t audiobuffer[NUM_ABUF][ABUF_SIZE];		/* File read buffer for audio */
volatile bool abuf_full[NUM_ABUF];
bool file_open = false;
int fill_buf = 0;
int lastbuffer = 0;
volatile uint8_t volume = DFLT_VOLUME;			// Volume range is 0 to 255
uint8_t noisehigh[8] = {0xff, 0xdd, 0xbb, 0x99, 0x77, 0x55, 0x33, 0x11};
```

```
char *rfilename = "songfile.raw";
volatile unsigned int irdat[SAMPLE_COUNT];
unsigned int z;
extern int IrFlag;
extern int flag;
int temp;
extern int volume2;
uint8_t ch0 = 3;
extern volatile int seccount;
extern volatile int tenseccount;
extern volatile int tenmincount;
extern volatile int mincount;
extern volatile int tenth;
int index;
extern int k;
extern int j;
extern int i;
extern int count;
#define IR_A 0x00ff22dd
#define IR_B 0x00ff02fd
#define IR_C 0x00ffc23d
#define IR_POWER 0x00ff629d
#define IR_UP 0x00ff9867
#define IR_DOWN 0x00ff38c7
#define IR_LEFT 0x00ff30cf
#define IR_RIGHT 0x00ff7a85
#define IR_CIRCLE 0x00ff18e7

FIL MyFile;                 /* File object */



/* USER CODE END PV */



uint32_t parseIRCode()
            {
   uint32_t num = 0;
                        int count = 0;
                        int arrayCount = 0;
                        uint32_t mask = 0x80000000;
                        /*skips header */
                        while(irdat[arrayCount] !=0)
                        {
                                    arrayCount++;
                        }
                        while(irdat[arrayCount] !=1)
                        {
                                    arrayCount++;
                        }
                        /* skips header */
                        while(arrayCount != SAMPLE_COUNT)
                        {
```

```
                                        if(irdat[arrayCount] ==0)
                                        {
                                                count++;
                                        }
                                        else
                                        {
                                                if(count < 8 && count !=0)
                                                {
                                                        mask/=2;
                                                }
                                                else if(count > 8 && count != 0)
                                                {
                                                        num |= mask;
                                                        mask/=2;
                                                }
                                                count = 0;
                                        }
                                        arrayCount++;
                                }
                                return num;
                        }



/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
```

```
MX_USB_HOST_Init();
MX_FATFS_Init();
MX_DAC1_Init();
MX_TIM6_Init();
MX_RTC_Init();
MX_LCD_Init();
MX_TIM16_Init();
MX_TIM7_Init();
/* USER CODE BEGIN 2 */

        /* initialize data structures */

        for (int i=0; i<NUM_ABUF; i++) {
                abuf_full[i] = false;
        }
        /* start devices */

HAL_DAC_Start(&hdac1, DAC1_CHANNEL_2);
        HAL_TIM_Base_Start_IT(&htim6);                  /* Start Timer 6 at 16KHz to run DAC */
        HAL_TIM_Base_Start_IT(&htim16);
        HAL_TIM_Base_Start_IT(&htim7);
        BSP_LCD_GLASS_Init();
        BSP_LCD_GLASS_Clear();
        uint8_t x = 48;//set up all zeroes
        BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_OFF , 5);
        BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_OFF , 4);
        BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_ON , 3);
        BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_OFF , 2);
        BSP_LCD_GLASS_BarLevelConfig(ch0);


/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */
  MX_USB_HOST_Process();

  /* USER CODE BEGIN 3 */
  /* Mass Storage Application State Machine */
                if(IrFlag == 1)
                {
                        uint32_t num = parseIRCode();
                        if(num == IR_A)
                        {

                        }
                        if(num == IR_B)
                        {

                        }
                        if(num == IR_C)
                        {
}
if(num == IR_POWER)
{

}

if(num == IR_UP)
{
        if(volume2+VOLUME_INCREMENT < 255)
        {
                volume2 += VOLUME_INCREMENT;
        }
        if(volume2 > 190)
        {
                ch0=4;
        }
        if(volume2 > 130 && volume2 <= 190)
        {
                ch0=3;
        }
        if(volume2 <= 130 && volume2 > 70)
        {
                ch0=2;
        }
        if(volume2 <=70)
        {
                ch0 = 1;
        }
}

if(num == IR_DOWN)
{
        if(volume2-VOLUME_INCREMENT > 0)
        {
                volume2 -= VOLUME_INCREMENT;
        }
        if(volume2 > 190)
        {
                ch0=4;
        }
        if(volume2 > 130 && volume2 <= 190)
        {
                ch0=3;
        }
        if(volume2 <= 130 && volume2 > 70)
        {
                ch0=2;
        }
        if(volume2 <=70)
        {
                ch0 = 1;
        }
}
```

```
if(num == IR_LEFT)
{
        abuf_full[0] = true;
        abuf_full[1] = true;
        for(index=0; index < ABUF_SIZE; index++)
        {
                audiobuffer[0][index] = 0;
                audiobuffer[1][index] = 0;
        }
        if(flag == 2)
        {
                flag = 0;
                Appli_state = APPLICATION_READY;
                count = 0;
                k=0;
        }
        i=0;
        j=0;
        abuf_full[0] = false;
        abuf_full[1] = false;
        fill_buf = 0;
        lastbuffer = 0;
        FS_FileClose();
        file_open = false;
        BSP_LCD_GLASS_Clear();
        x = 48;//set up all zeroes
        seccount = 0;
        tenseccount = 0;
        tenmincount = 0;
        mincount = 0;
        tenth = 0;
        BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_OFF , 5);
        BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_OFF , 4);
        BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_ON , 3);
        BSP_LCD_GLASS_DisplayChar(&x, POINT_OFF, DOUBLEPOINT_OFF , 2);
        BSP_LCD_GLASS_BarLevelConfig(ch0);

}
if(num == IR_RIGHT)
{

}
if(num == IR_CIRCLE)
{
        //HAL_GPIO_TogglePin(LD_G_GPIO_Port, LD_G_Pin);
        if(flag == 2)
        {
                flag = temp;
        }
        else
        {
                temp = flag;
                flag = 2;
        }
}
```

```
                                    IrFlag=0;
                            }
   switch(Appli_state) {
    case APPLICATION_READY:
      if (!file_open) FS_FileOpen();
                                     if (file_open && (lastbuffer == 0)) {
                                                    if (!abuf_full[fill_buf]) {
                                                            FS_FileRead((uint8_t *) audiobuffer[fill_buf]);
                                                            abuf_full[fill_buf++] = true;
                                                            fill_buf = fill_buf % NUM_ABUF;
                                                    }
                                            }
                                            if (file_open && (lastbuffer != 0)) {
                                                    FS_FileClose();
        Appli_state = APPLICATION_IDLE;
                                            }
      break;

    case APPLICATION_IDLE:
      default:
      break;
    }





                            }
  /* USER CODE END 3 */
}
```