CAL POLY

CALIFORNIA POLYTECHNIC STATE UNIVERSITY
CENG - DEPARTMENT OF ELECTRICAL ENGINEERING

EE 329-05: Microcontroller-Based Systems Design
Final Project Capstone: JAB Drum Board

Presented to:
John Penvenne

Presented By:
Justin Rosu
Brayden Daly
Alexander von Fuchs

June 6, 2025

Behavior Description

The JAB Drum Board is a compact, interactive audio playback device designed to mimic the core functionality of a DJ controller. The system integrates user input through potentiometers and external buttons to trigger various audio samples and apply simple sound effects. When the user presses a button, the corresponding pre-stored sound sample is played through an interrupt-driven DAC output at a fixed 44.1 kHz sample rate, ensuring smooth, real-time audio playback.

Potentiometers allow for dynamic adjustment of audio parameters such as pitch or gain, while an adjustable bandpass filter processes the output signal to shape the tonal quality. Each button and knob acts as a modular control, giving users hands-on influence over sound loops and effect modes. The audio data is stored in onboard flash memory, retrieved on demand, and output via an AC-coupled analog circuit to standard speaker hardware.

The behavior of the board reflects a simplified digital audio workstation (DAW), enabling basic composition, live audio manipulation, and experimentation with filters and superposition. The system is optimized for low-latency response and includes future extensibility for DSP enhancements like FFT-based effects or more sophisticated memory management.

System Specifications

Parameter	Value	Units	Description
MCU	STM32L4A6ZG	N/A	Microcontroller used
Operating Clock Speed	16	MHz	System clock frequency
DAC Resolution	12	bits	Digital-to-Analog converter resolution
SPI Clock Speed	16	MHz	Maximum SPI communication speed
Timer Frequency	44.4	kHz	PWM update frequency (TIM2 with ARR=360 @16 MHz)
Timer Duty Cycle Range	0 – 360	Counts	PWM duty cycle value range
Audio Output Range	0 – 3.3	Volts (approx.)	Analog voltage output range from DAC

Audio Sampling Rate	44.1 (configurable)	kHz	Approximate audio sample rate handled
Input Interface	Buttons	N/A	User input via buttons
SPI Data Width	16	bits	Data width configured for SPI transfers
Interrupt Latency	<1	µs	Interrupt response time
Power Supply Voltage	3.3	Volts	Operating voltage for the system
Total Power Consumption	<100	mW	Estimated total power consumption

Code Planning & Software Architecture

This code implements a real-time audio playback system on an STM32L4A6ZG microcontroller using digital-to-analog conversion. It utilizes Timer 2 to periodically trigger interrupts, during which the system mixes up to four audio streams based on playback flags and sample values. These mixed samples are then written to a 12-bit DAC via SPI communication. The main application manages input from a 4x4 keypad to trigger audio samples, adjust playback, or control system states. The system is designed to produce smooth analog audio output from digital sources with precise timing using low-level register access for efficiency and control.

PSEUDOCODE

- Initialize system peripherals (GPIO, SPI, TIM2, DAC)
- Configure SPI and Timer peripherals
- Enable interrupts for timer
- Enter infinite loop:
  - Optionally handle other tasks or background processes (e.g., keypad scanning, user input)
  - Main audio processing and DAC output handled inside TIM2 interrupt
  - Monitor system state or flags as needed

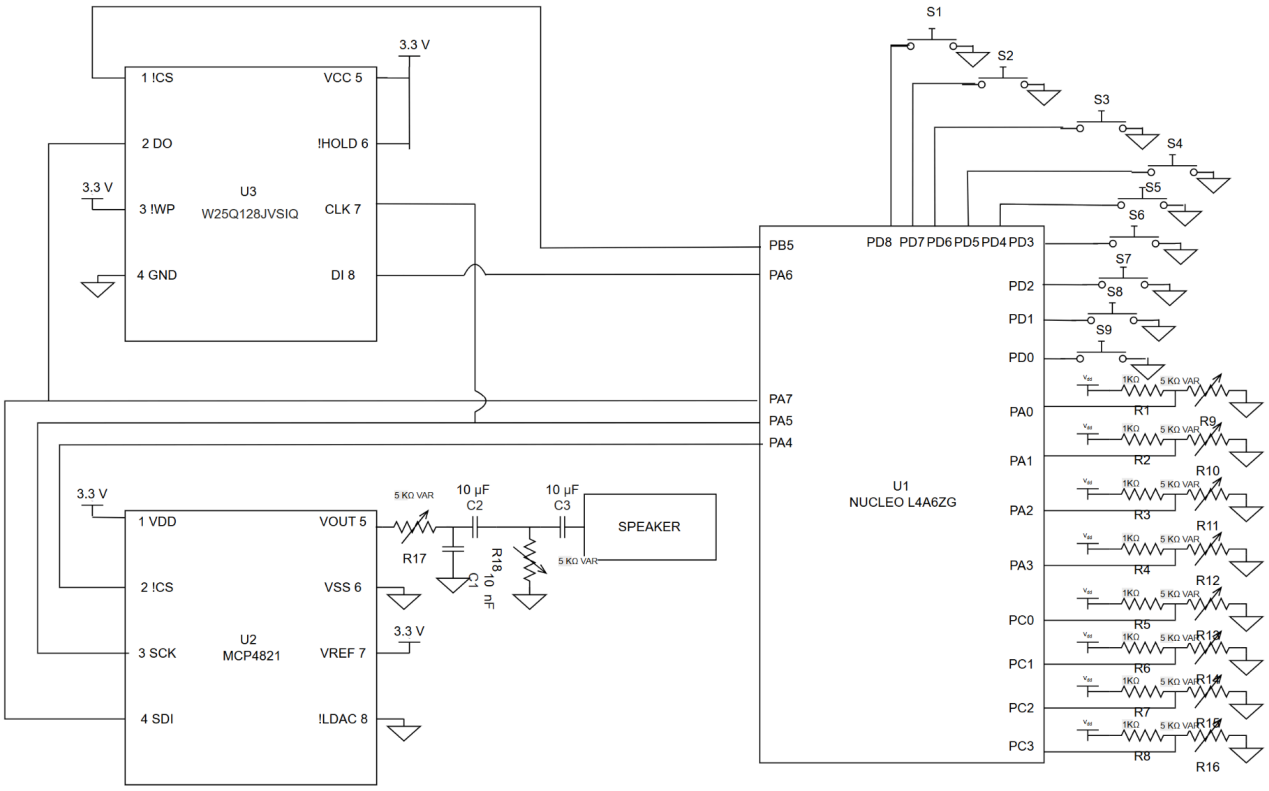
SPI:
- Enable GPIOA and SPI1 clocks for SPI communication
- Configure GPIOA pins (PA4, PA5, PA7) for SPI alternate functions with push-pull output and high speed
- Set SPI1 peripheral as master

- Disable SPI during configuration
- Configure SPI for:
  - 16-bit data frames
  - Motorola frame format
  - MSB first transmission
  - Clock polarity and phase = 0
  - Automatic NSS pulse generation
  - Enable SS output
- Enable SPI peripheral

- Enable TIM2 clock
- Configure TIM2:
  - Set auto-reload register (ARR) for timer period
  - Set compare register (CCR1) for duty cycle
  - Enable capture/compare interrupt and update interrupt
- Clear interrupt flags
- Enable TIM2 interrupt in NVIC
- Enable global interrupts
- Start TIM2 timer

- In TIM2 interrupt handler:
  - Check for capture/compare interrupt flag, clear if set
  - Check for update interrupt flag, clear if set
    - Calculate mixed audio output by summing scaled audio samples only if respective streams are active
    - Apply offset to center combined signal
    - Send combined output to DAC via SPI
    - Increment audio sample counters for each stream

System Schematic



Calculation & Characterization

Interrupt Timer Calculations:

$$\frac{1}{16MHZ} = 62.5\ ns \quad \frac{DESIRED\ PLAYBACK\ TIME}{62.5ns} = \frac{22.68us}{62.5ns} = 363\ ticks$$

Audio Normalization:

DAC OUT = (AUDIO SAMPLE1- 256) *IS_AUDIOPLAYING +
(AUDIO SAMPLE2- 256) *IS_AUDIOPLAYING +
(AUDIO SAMPLE3- 256) *IS_AUDIOPLAYING +
(AUDIO SAMPLE4- 256) *IS_AUDIOPLAYING

Attribution

Justin: Audio Driver Circuit & Software, helped with memory integration

Brayden: Memory integration, helped with Audio Driver Software, Analog Reading

Alex: Bandpass filter, analog read functionality, soldering hardware, wireharnessing, 3D design of overall project, hardware selection

Appendix A : Bill of Materials

QTY	Part type	Designator	Description	vendor	Vendor p/n	Unit $	$ qty
2	CAPACITOR	C2-3	CAP CER 10UF 50V X7S RADIAL	Digikey	445-FK20X7S1H106K-ND	$1.04	$2.08
1	CAPACITOR	C1	CAP CER 10000PF 50V X7R RADIAL	Digikey	399-4148-ND	$0.23	$0.23
10	POTENIAMTER	R9-18	Potentiometer 5K B5K Variable Resistors 15mm Shaft 3Pins 5K	Amazon	B00N1ZIXKA	$0.46	$4.65
1	IC	U3	IC FLASH 128MBIT SPI/QUAD 8SOIC	Digikey	W25Q128JVSIQTR-ND	$1.53	$1.53
1	IC	U2	IC DAC 12BIT V-OUT 8DIP	Digikey	MCP4821-E/P-ND	$3.27	$3.27
1	MCU	U1	NUCLEO-144 STM32L4A6ZG EVAL BRD	Digikey	497-NUCLEO-L4A6ZG-ND	$19.99	$19.99
3	SWITCH	S3-5	12mm Momentary Push Button Switch Black Shell with pre-Wiring	Amazon	B09BKXT1J1	$2.50	$7.50
4	SWITCH	S6-9	SWITCH TACTILE SPST-NO 0.05A 12V	Amazon	2223-TS02-66-70-BK-260-LCR-D-ND	$0.10	$0.40
2	SWITCH	S1-2	16mm Momentary Push Button Switch Sliver Shell	Amazon	B09BKYLWCH	$2.70	$5.40
8	RESISTOR	R0-R8	RES 1K OHM 1% 1/8W AXIAL	Digikey	RNF18FTD1K00CT-ND	$0.10	$0.10

Appendix B: References

STMicroelectronics, *STM32L47xxx, STM32L48xxx, STM32L49xxx and STM32L4Ax 32-bit advanced ARM®-based MCUs: Reference Manual*, RM0351, Rev 6, Aug. 2023. [Online]. Available: https://www.st.com/resource/en/reference_manual/rm0351-stm32l47xxx-stm32l48xxx-stm32l49xxx-and-stm32l4axxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

Winbond Electronics Corporation, *W25Q128JV 128M-bit Serial Flash Memory with Dual/Quad SPI*, Datasheet, Rev. F, Mar. 27, 2018. [Online].

Appendix C : Source code

Python Code to process sound data

```
import numpy as np
from scipy.io import wavfile
import os

MAX_BYTES = 300 * 1024  # 300 KB
MAX_SAMPLES = MAX_BYTES // 2  # 2 bytes per uint16_t

#WAV -> array(ints) -> (possibly uart)

def wav_to_c_array(filename, output_name="audio_data"):
    sample_rate, audio = wavfile.read(filename)

    # Convert stereo to mono if needed
    if len(audio.shape) == 2:
        print("Stereo detected. Averaging channels to mono.")
        audio = audio.mean(axis=1)

    # Normalize float audio to 16-bit signed integer
    if np.issubdtype(audio.dtype, np.floating):
        audio = (audio * 32767.0).astype(np.int16)

    # Limit to MAX_SAMPLES
    audio = audio[:MAX_SAMPLES]

    # Scale to 12-bit unsigned (0–4095)
    audio = ((audio.astype(np.int32) + 32768) * (4095.0 / 65535.0)).astype(np.uint16)
    return audio.tolist()


#convert
def uint16_to_bytes_big_endian(input_array):

    byte_array = []
    for val in input_array:
        intval = int(val)
        hexval = hex(intval)
        hex1 = hexval[2:]
        hex2temp = hexval[3::-1]
        hex2 = hex2temp [::-1]
        byte_array.extend(intval.to_bytes(2, byteorder='big'))

    # Format as hex string like 0x07, 0xFF, etc.
    hex_list = [f"0x{byte:02X}" for byte in byte_array]

    # Join into lines of 16 bytes for readability
    lines = [", ".join(hex_list[i:i+16]) for i in range(0, len(hex_list), 16)]

    with open("audio_data_bytes.txt", "w") as f:
        for line in lines:
```

```
            f.write("  " + line + ",\n")

    print("Saved to audio_data_bytes.txt")


# Example usage
if __name__ == "__main__":
    import sys
    input_wav = sys.argv[1] if len(sys.argv) > 1 else "input.wav"
    #files = find_wav_files(os.getcwd())
    #for i in files:
    #    print(i)
    c_array_code = wav_to_c_array(input_wav)
    uint16_to_bytes_big_endian(c_array_code)
```

DAC.h

```
/*******************************************************************************
* EE 329 A5 DAC INTERFACE
*******************************************************************************
* @file           : DAC.h
* @brief          : Runs the functions from Keypad.c and outputs onto LEDs
* project         : EE 329 S'25 Assignment 2
* authors         : Justin Rosu & Stephanie Ly  (JRSL)
* version         : 0.1
* date            : 250412
* compiler        : STM32CubeIDE v.1.12.0 Build: 14980_20230301_1550 (UTC)
* target          : NUCLEO-L4A6ZG
* clocks          : 4 MHz MSI to AHB2
* @attention      : (c) 2023 STMicroelectronics.  All rights reserved.
*******************************************************************************
* KEYPAD PLAN :
* set columns as outputs, rows as inputs w pulldowns
* loop:
* drive all columns HI read all rows
* if any row N is HI
*    set all columns LO
*    drive each column M HI alone
*    read row N until HI □ pressed key loc'n = N, M
* key value = 3N+M+1 for 1..9,special case for *,0,#
*******************************************************************************
* KEYPAD WIRING 4 ROWS 4 COLS (pinout NUCLEO-L4A6ZG = L496ZG)
* CS - PA4
* CLCK - PB5
* SDI - PB7
*******************************************************************************
* REVISION HISTORY
* 0.1 230318 JRSL  created, wires in breadboard, no dac
* 0.2 230410 JRSL  made code to make dac operational
* 0.3 230413 JRSL  implemented modularity
```

```
*****************************************************************************
*****************************************************************************
* 45678-1-2345678-2-2345678-3-2345678-4-2345678-5-2345678-6-2345678-7-234567 */
#ifndef INC_DAC_H_
#define INC_DAC_H_
int DAC_volt_conv(float voltage );
void DAC_init(void);
void DAC_Write(uint16_t, uint8_t );
#endif /* INC_DAC_H_ */
```

DAC.c

```
#include "SPI2.h"
#include "SPI.h"
#include "main.h"
/* ---------------------------- (DAC_init) -----------------------------
* Initializes all things needed for the DAC to work
* Inputs: None
* Outputs: None
* Local vars: None
* ---------------------------------------------------------------------- */
void DAC_init(void){
	//!!!!!NEEED TO USE SPIBUS2 for DAC (16 bit mode and NSS -> Look for original code)
	SPI_Config();
	SPI_init();
}
/* ---------------------------- (DAC_volt_conv) -----------------------------
* Takes desired voltage and ouputs it to a value for the dac
* Inputs: float voltage
* Outputs: int value
* Local vars: None
* ---------------------------------------------------------------------- */
int DAC_volt_conv(float voltage ){
	if (voltage > 3.30){
		return (3.30 *4095)/(2* 2.048);;
	}
	if (voltage > 2.04){
		 return (voltage *4095)/(2* 2.048);
	}
	else{
		return (voltage *4095)/(2.048);
	}
}
/* ---------------------------- (DAC_Write) -----------------------------
* Writes the 12 bits to the dac
* Inputs: dac_code --> 12 bits input to dac, gain_bit --> determines gain
* Outputs: None
* Local vars: word --> is the bits output to the SDI
```

```
* ---------------------------------------------------------------------- */
void DAC_Write(uint16_t dac_code, uint8_t gain_bit){
	dac_code &= 0xFFF; //Ensures input is only 12 bits
	uint16_t word = (gain_bit << 13) | (1<<12) | dac_code;// Makes 16 bit sequence
	while(SPI1->SR & SPI_SR_BSY);
	SPI1->DR = word;		//Outputs 12 bits when dac is ready
	while(!(SPI1->SR & SPI_SR_TXE));
	while(SPI1->SR & SPI_SR_BSY);
}
```

SPI.h

```
/*******************************************************************************
* EE 329 SPI Configuration - SPI Initialization Module
*******************************************************************************
* @file		: SPI_Config.c
* @brief		: Configures SPI1 and associated GPIO pins to communicate
*		with external DAC. Sets up GPIO modes, alternate functions,
*		and SPI control registers.
* project		: EE 329 S'25 Audio Playback Project
* authors		: Justin Rosu	(jrosu@calpoly.edu)
*		Brayden Daly	(bdaly01@calpoly.edu)
*		Alex Von Fuch	(avonfuch@calpoly.edu)
* version		: 1.0
* date		: 250606
* compiler		: STM32CubeIDE v.1.12.0 Build: 14980_20230301_1550 (UTC)
* target		: NUCLEO-L4A6ZG
* clocks		: 4 MHz MSI to AHB2
* @attention	: (c) 2025 STMicroelectronics. All rights reserved.
*******************************************************************************
* FUNCTIONAL OVERVIEW:
* - SPI_Config(): Initializes GPIOA for SPI1 alternate functions.
* - SPI_init(): Configures SPI1 control registers to send 16-bit frames to DAC.
*
* SPI1 Pin Assignments:
*   PA4 - NSS (manual control or NSSP)
*   PA5 - SCK
*   PA7 - MOSI (SDO)
*
* Note: SPI1 must be idle (BSY=0) before configuration.
*******************************************************************************
* REVISION HISTORY
* 1.0 250606 JR/B.D./A.V.F. Initial creation and SPI1 configuration
****
*/
#ifndef SRC_SPI_H_
#define SRC_SPI_H_
void SPI_init( void );
```

```
void SPI_Config(void);
#endif /* SRC_SPI_H_ */
```

SPI.c

```
#include "main.h"
#include "stm32l4xx_hal.h"
/* ---------------------------- (SPI_Config) -----------------------------
* Initializes all things needed for the SPI to work
* Inputs: None
* Outputs: None
* Local vars: None
* ------------------------------------------------------------------------ */
void SPI_Config(void){
        RCC->AHB2ENR |= (RCC_AHB2ENR_GPIOAEN);          // GPIOA: DAC NSS/SCK/SDO
        RCC->APB2ENR |= (RCC_APB2ENR_SPI1EN);           // SPI1 port
        /* USER ADD GPIO configuration of MODER/PUPDR/OTYPER/OSPEEDR registers HERE */
        // configure AFR for SPI1 function (1 of 3 SPI bits shown here)
        // set MODER: AF mode for PA5, PA7; Output mode for PA4
        GPIOA->MODER &= ~(GPIO_MODER_MODE4 |GPIO_MODER_MODE5|
GPIO_MODER_MODE7 );
        GPIOA->MODER |=  ((2 << 8) | (2 << 10) | (2 << 14));
        // set OTYPER: push-pull for all SPI pins
        GPIOA->OTYPER &= ~(GPIO_OTYPER_OT4 | GPIO_OTYPER_OT5 | GPIO_OTYPER_OT7);
        // set OSPEEDR: high speed for all SPI pins"
        GPIOA->OSPEEDR &=  ~((GPIO_OSPEEDR_OSPEED4_Pos) |
(GPIO_OSPEEDR_OSPEED5_Pos) | (GPIO_OSPEEDR_OSPEED7_Pos));
        GPIOA->OSPEEDR |= ((3 << GPIO_OSPEEDR_OSPEED4_Pos) |( 3<<
GPIO_OSPEEDR_OSPEED5_Pos) | (3 << (GPIO_OSPEEDR_OSPEED7_Pos)));
        // set PUPDR: no pull-up/pull-down
  //GPIOA->PUPDR &= ~((3U << (4 * 2)) | (3U << (5 * 2)) | (3U << (7 * 2)));
        GPIOA->AFR[0] &= ~((0x000F << GPIO_AFRL_AFSEL7_Pos)); // clear nibble for bit 7 AF
        GPIOA->AFR[0] |=  ((0x0005 << GPIO_AFRL_AFSEL7_Pos)); // set b7 AF to SPI1 (fcn 5)
        GPIOA->AFR[0] &= ~((0x000F << GPIO_AFRL_AFSEL5_Pos)); // clear nibble for bit 5 AF
        GPIOA->AFR[0] |=  ((0x0005 << GPIO_AFRL_AFSEL5_Pos)); // set b5 AF to SPI1 (fcn 5)
        GPIOA->AFR[0] &= ~((0x000F << GPIO_AFRL_AFSEL4_Pos)); // clear nibble for bit 4 AF
        GPIOA->AFR[0] |=  ((0x0005 << GPIO_AFRL_AFSEL4_Pos)); // set b4 AF to SPI1 (fcn 5)
}
/* ---------------------------- (SPI_init) -----------------------------
* Initializes all things needed for the SPI to work
* Inputs: None
* Outputs: None
* Local vars: None
* ------------------------------------------------------------------------ */
void SPI_init( void ) {
  // SPI config as specified @ STM32L4 RM0351 rev.9 p.1459
  // called by or with DAC_init()
```

```
  // build control registers CR1 & CR2 for SPI control of peripheral DAC
  // assumes no active SPI xmits & no recv data in process (BSY=0)
  // CR1 (reset value = 0x0000)
  SPI1->CR1 &= ~( SPI_CR1_SPE );              // disable SPI for config
  SPI1->CR1 &= ~( SPI_CR1_RXONLY );           // recv-only OFF
  SPI1->CR1 &= ~( SPI_CR1_LSBFIRST );         // data bit order MSb:LSb
  SPI1->CR1 &= ~( SPI_CR1_CPOL | SPI_CR1_CPHA ); // SCLK polarity:phase = 0:0
  SPI1->CR1 |=  SPI_CR1_MSTR;                 // MCU is SPI controller
  // CR2 (reset value = 0x0700 : 8b data)
  SPI1->CR2 &= ~( SPI_CR2_TXEIE | SPI_CR2_RXNEIE ); // disable FIFO intrpts
  SPI1->CR2 &= ~( SPI_CR2_FRF);               // Moto frame format
  SPI1->CR2 |=  SPI_CR2_NSSP;                 // auto-generate NSS pulse
  SPI1->CR2 |=  SPI_CR2_DS;                   // 16-bit data
  SPI1->CR2 |=  SPI_CR2_SSOE;                 // enable SS output// CR1
  SPI1->CR1 |=  SPI_CR1_SPE;                  // re-enable SPI for ops
}
```

Buttons.c

```
/*
* BUTTONS.c
*
*  Created on: May 29, 2025
*      Author: jrosu
*/
#include <KEYPAD.h>
#include "BUTTONS.h"
#include "main.h"
int last_button_state[] = {0,0,0,0};
/* ---------------------------- (Button_Configuration ) -----------------------------
* Initializes pind D0-6 for input mode
* Inputs: None
* Outputs: None
* Local vars: None
* ------------------------------------------------------------------------------ */
void Button_Configuration(void)
{
        //Initialize clock and inputs and outputs and stuff
        //Enable GPIOD clock for peripheral
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIODEN;
        RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
        //set GPIO to input mode (Reset bits 6 and 7 for input mode) (00)
        GPIOD->MODER = 0;
        //INITIALIZES ALL INPUTS
```

```
	GPIOB->MODER &= ~(GPIO_MODER_MODE0 | GPIO_MODER_MODE1 |
GPIO_MODER_MODE2 | GPIO_MODER_MODE3 |
GPIO_MODER_MODE4|GPIO_MODER_MODE5|GPIO_MODER_MODE6 |GPIO_MODER_MODE8);
	GPIOB->MODER |=  (GPIO_MODER_MODE0_0 | GPIO_MODER_MODE1_0 |
GPIO_MODER_MODE2_0 | GPIO_MODER_MODE3_0 |GPIO_MODER_MODE4_0|
GPIO_MODER_MODE5_0|GPIO_MODER_MODE6_0| GPIO_MODER_MODE8_0);
	//Initializes PUPDR
	GPIOD->PUPDR &= PUPDRST;
	GPIOD->PUPDR |= (GPIO_PUPDR_PUPD0_1 | GPIO_PUPDR_PUPD1_1 |
GPIO_PUPDR_PUPD2_1 | GPIO_PUPDR_PUPD3_1 |GPIO_PUPDR_PUPD4_1
|GPIO_PUPDR_PUPD5_1 |GPIO_PUPDR_PUPD6_1);
	GPIOB->PUPDR |= (GPIO_PUPDR_PUPD0_0 | GPIO_PUPDR_PUPD1_0 |
GPIO_PUPDR_PUPD2_0 | GPIO_PUPDR_PUPD3_0 | GPIO_PUPDR_PUPD3_0);
	GPIOD->OTYPER &= (GPIO_OTYPER_OT0 | GPIO_OTYPER_OT1 |GPIO_OTYPER_OT2
|GPIO_OTYPER_OT3| GPIO_OTYPER_OT4 |GPIO_OTYPER_OT5 | GPIO_OTYPER_OT6);
}
/* ---------------------------- (detect_Button_Press ) -----------------------
* Checks if button is pressed and if sound is not already playing
* Inputs: pin, audio play
* Outputs: 1 or 0
* Local vars: None
* ------------------------------------------------------------------------- */
uint8_t detect_Button_Press(uint8_t pin, uint8_t audio_queue) {
  if (pin > 3) return 0;  // invalid
  if ((GPIOD->IDR & (1U << pin)) && (!audio_queue)){
			return 1;
  }
  return 0;
}
```

Buttons.h

```
#ifndef BUTTONS_H
#define BUTTONS_H
#include <stdint.h>
extern volatile uint8_t button_flags;
void Button_Init(void);
uint8_t Button_WasPressed(uint8_t pin,uint8_t audio_queue);
#endif
///*
// * BUTTONS.h
// *
// *  Created on: May 29, 2025
// *      Author: jrosu
// */
//
//#ifndef INC_BUTTONS_H_
```

```
//#define INC_BUTTONS_H_
//
//#define INPUT_PORT GPIOC
//#define DEBOUNCE_DELAY_MS 20
//uint8_t detect_Button_Press(uint8_t);
//void SysTick_Init(void);
//void Button_Configuration(void);
//
//#endif /* INC_BUTTONS_H_ */
```

Main.h

```
/******************************************************************************
* EE 329 Audio Playback System - Main Control
******************************************************************************
* @file          : main.c
* @brief         : Initializes system peripherals and handles playback logic
*                  for multi-track audio with DAC output and button control.
* project         : EE 329 S'25 Audio Playback Project
* authors         : Justin Rosu     (jrosu@calpoly.edu)
*                  Brayden Daly    (bdaly01@calpoly.edu)
*                  Alex Von Fuch   (avonfuch@calpoly.edu)
* version         : 1.0
* date            : 250606
* compiler        : STM32CubeIDE v.1.12.0 Build: 14980_20230301_1550 (UTC)
* target          : NUCLEO-L4A6ZG
* clocks          : 4 MHz MSI to AHB2
* @attention      : (c) 2025 STMicroelectronics. All rights reserved.
******************************************************************************
* FUNCTIONAL OVERVIEW:
* - Initializes HAL, DAC, timers, GPIOs, and buttons
* - Handles up to four concurrent audio sample playbacks
* - Selects playback mode via external GPIO inputs
* - Audio samples stored as uint16_t arrays in flash memory
* - Playback triggered by button press detection
******************************************************************************
* REVISION HISTORY
* 1.0 250606 JR/B.D./A.V.F. Initial integration and multi-mode playback support
******************************************************************************
* 45678-1-2345678-2-2345678-3-2345678-4-2345678-5-2345678-6-2345678-7-2345678 */
#ifndef __MAIN_H
#define __MAIN_H
#ifdef __cplusplus
extern "C" {
#endif
/* Includes ------------------------------------------------------------------*/
#include "stm32l4xx_hal.h"
extern uint16_t audio_sample, audio_sample2,audio_sample3,audio_sample4;
```

```
extern uint16_t audio_counter,audio_counter2,audio_counter3,audio_counter4;
extern uint8_t audio_playing,audio_playing2,audio_playing3,audio_playing4;
extern uint8_t which_audio_playing[];
void Error_Handler(void);
#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
#define LD3_Pin GPIO_PIN_14
#define LD3_GPIO_Port GPIOB
#define USB_OverCurrent_Pin GPIO_PIN_5
#define USB_OverCurrent_GPIO_Port GPIOG
#define USB_PowerSwitchOn_Pin GPIO_PIN_6
#define USB_PowerSwitchOn_GPIO_Port GPIOG
#define STLK_RX_Pin GPIO_PIN_7
#define STLK_RX_GPIO_Port GPIOG
#define STLK_TX_Pin GPIO_PIN_8
#define STLK_TX_GPIO_Port GPIOG
#define USB_SOF_Pin GPIO_PIN_8
#define USB_SOF_GPIO_Port GPIOA
#define USB_VBUS_Pin GPIO_PIN_9
#define USB_VBUS_GPIO_Port GPIOA
#define USB_ID_Pin GPIO_PIN_10
#define USB_ID_GPIO_Port GPIOA
#define USB_DM_Pin GPIO_PIN_11
#define USB_DM_GPIO_Port GPIOA
#define USB_DP_Pin GPIO_PIN_12
#define USB_DP_GPIO_Port GPIOA
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
#define LD2_Pin GPIO_PIN_7
#define LD2_GPIO_Port GPIOB
#ifdef __cplusplus
}
#endif
#endif /* __MAIN_H */
```

Main.c

```
#include "main.h"
#include "DAC.h"
#include <math.h>
#include <stdint.h>
#include "BUTTONS.h"
#include <MEM.h>
// ------------------------- Global Variables -------------------------
```

```
// These hold audio playback state, counters, and sample data
uint16_t audio_sample, audio_sample2, audio_sample3, audio_sample4;
uint16_t audio_counter, audio_counter2, audio_counter3, audio_counter4;
uint8_t audio_playing, audio_playing2, audio_playing3, audio_playing4;
uint8_t mode;
// External audio sample arrays (presumably defined elsewhere)
const uint16_t audio_data[];
const uint16_t audio_data2[];
const uint16_t audio_data3[];
const uint16_t audio_data4[];
const uint16_t audio_data5[];
const uint16_t audio_data6[];
const uint16_t audio_data7[];
const uint16_t audio_data8[];
const uint16_t audio_data9[];
const uint16_t audio_data10[];
const uint16_t audio_data11[];
const uint16_t audio_data12[];
// Lengths of each sample dataset
const size_t audio_data_len = 4189;
const size_t audio_data_len2 = 15000;
const size_t audio_data_len3 = 40000;
const size_t audio_data_len4 = 50093;
const size_t audio_data_len5 = 23752;
const size_t audio_data_len6 = 10886;
const size_t audio_data_len7 = 7917;
const size_t audio_data_len8 = 16990;
const size_t audio_data_len9 = 25521;
const size_t audio_data_len10 = 31993;
const size_t audio_data_len11 = 26260;
const size_t audio_data_len12 = 24596;
// Forward declarations
void delay_us(const uint32_t time_us);
void SystemClock_Config(void);
/* ---------------------------- (main ) -----------------------------
* Main entry point: initializes hardware and runs audio playback loop
* Inputs: None
* Outputs: None
* Local vars: audio_lengths[], audio_lengths2[], audio_lengths3[], audio_lengths4[]
* ---------------------------------------------------------------- */
int main(void)
{
 // ---- Hardware Setup ----
 HAL_Init();
 SystemClock_Config();
 DAC_init();
 Button_Configuration();
 Timer_setup_TIM2(10000);  // 10ms tick?
 GPIOB_Config();
```

```
// ---- Initialize State ----
audio_counter = audio_counter2 = audio_counter3 = audio_counter4 = 0;
audio_sample  = audio_data[audio_counter];
audio_sample2 = audio_data2[audio_counter2];
audio_sample3 = audio_data3[audio_counter3];
audio_sample4 = 0;
audio_playing = audio_playing2 = audio_playing3 = audio_playing4 = 0;
mode = 0;
// Sample length tables for different modes
uint32_t audio_lengths[]  = {audio_data_len,  audio_data_len5,  audio_data_len9};
uint32_t audio_lengths2[] = {audio_data_len2, audio_data_len6,  audio_data_len10};
uint32_t audio_lengths3[] = {audio_data_len3, audio_data_len7,  audio_data_len11};
uint32_t audio_lengths4[] = {audio_data_len4, audio_data_len8,  audio_data_len12};
// ---- Main Loop ----
while (1)
{
  // --- Mode Switching via GPIOD pins ---
  if (GPIOD->IDR & (1 << 4)) mode = 0;
  if (GPIOD->IDR & (1 << 5)) mode = 1;
  if (GPIOD->IDR & (1 << 6)) mode = 2;
  // --- Detect button presses to start playback ---
  if (detect_Button_Press(0, audio_playing)) {
    audio_playing = 1;
    audio_counter = 0;
  }
  if (detect_Button_Press(1, audio_playing2)) {
    audio_playing2 = 1;
    audio_counter2 = 0;
  }
  if (detect_Button_Press(2, audio_playing3)) {
    audio_playing3 = 1;
    audio_counter3 = 0;
  }
  if (detect_Button_Press(3, audio_playing4)) {
    audio_playing4 = 1;
    audio_counter4 = 0;
  }
  // --- Playback channel 1 ---
  if (audio_playing) {
    if (mode == 0) audio_sample = audio_data[audio_counter];
    if (mode == 1) audio_sample = audio_data5[audio_counter];
    if (mode == 2) audio_sample = audio_data9[audio_counter];
    if (audio_counter > audio_lengths[mode] - 1) {
      audio_counter = 0;
      audio_playing = 0;
    }
  }
  // --- Playback channel 2 ---
  if (audio_playing2) {
```

```
    if (mode == 0) audio_sample2 = audio_data2[audio_counter2];
    if (mode == 1) audio_sample2 = audio_data6[audio_counter2];
    if (mode == 2) audio_sample2 = audio_data10[audio_counter2];
    if (audio_counter2 > audio_lengths2[mode] - 1) {
      audio_counter2 = 0;
      audio_playing2 = 0;
    }
  }
  // --- Playback channel 3 ---
  if (audio_playing3) {
    if (mode == 0) audio_sample3 = audio_data3[audio_counter3];
    if (mode == 1) audio_sample3 = audio_data7[audio_counter3];
    if (mode == 2) audio_sample3 = audio_data11[audio_counter3];
    if (audio_counter3 > audio_lengths3[mode] - 1) {
      audio_counter3 = 0;
      audio_playing3 = 0;
    }
  }
  // --- Playback channel 4 ---
  if (audio_playing4) {
    if (mode == 0) audio_sample4 = audio_data4[audio_counter4];
    if (mode == 1) audio_sample4 = audio_data8[audio_counter4];
    if (mode == 2) audio_sample4 = audio_data12[audio_counter4];
    if (audio_counter4 > audio_lengths4[mode] - 1) {
      audio_counter4 = 0;
      audio_playing4 = 0;
    }
  }
 }
}
/* ---------------------------- (SystemClock_Config ) ----------------------------
* Configures the system clock to use MSI with no PLL, and sets clock dividers
* Inputs: None
* Outputs: None
* Local vars: RCC_OscInitStruct, RCC_ClkInitStruct
* -------------------------------------------------------------------------- */
void SystemClock_Config(void)
{
 RCC_OscInitTypeDef RCC_OscInitStruct = {0};
 RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
 if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK) {
   Error_Handler();
 }
 RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
 RCC_OscInitStruct.MSIState = RCC_MSI_ON;
 RCC_OscInitStruct.MSICalibrationValue = 0;
 RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_8;
 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
 if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
```

```
  Error_Handler();
 }
 RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK
                             | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
 RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_MSI;
 RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
 RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
 RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK) {
  Error_Handler();
 }
}
/* ---------------------------- (Error_Handler ) ----------------------------
* Handles unexpected system errors by disabling interrupts and entering infinite loop
* Inputs: None
* Outputs: None
* Local vars: None
* -------------------------------------------------------------------------- */
void Error_Handler(void)
{
 __disable_irq();
 while (1)
 {
  // Stay here for debugging
 }
}
/* ---------------------------- (assert_failed ) ----------------------------
* Called by assert macro when an error is detected
* Inputs: file (source file name), line (line number)
* Outputs: None
* Local vars: None
* -------------------------------------------------------------------------- */
#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line)
{
 // Optionally implement to log debug info
}
#endif /* USE_FULL_ASSERT */
```

MEM.c

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file          : main.c
 * @brief         : Main program body for SPI-based memory interaction
 ******************************************************************************
 */
/* USER CODE END Header */
```

```c
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "SPI.h"
#include "stm32l4xx.h"
#include <MEM.h>
/**
* @brief Configure GPIOB pins for SPI chip select (CS) control
*/
void GPIOB_Config(void)
{
  // Enable GPIOB clock
  RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
  // Configure PB5, PB6, PB8 as outputs (e.g., for chip select or control)
  CS->MODER &= ~(GPIO_MODER_MODE5 | GPIO_MODER_MODE6 | GPIO_MODER_MODE8);
  CS->MODER |=  (GPIO_MODER_MODE5_0 | GPIO_MODER_MODE6_0);
  // Set high speed for PB6 (optional depending on SPI speed)
  CS->OSPEEDR |=  (3 << GPIO_OSPEEDR_OSPEED6_Pos);
  // Set output type to push-pull (not open-drain) for PB6
  CS->OTYPER &= ~(GPIO_OTYPER_OT6);
}
/**
* @brief Send Write Enable (0x06) command to memory
*/
void write_enable(void)
{
  uint8_t cmd[1] = {0x06}; // Write Enable command
  GPIOB->BRR = GPIO_PIN_5;      // Set CS low
  SPI1_Transmit(cmd, 1);        // Send command
  GPIOB->BSRR = GPIO_PIN_5;     // Set CS high
}
/**
* @brief Wait until memory is no longer busy (polls the WIP bit)
*/
void wait_until_not_busy(void)
{
  uint8_t cmd[1] = {0x05}; // Read Status Register 1 command
  uint8_t status;
  do {
     GPIOB->BRR = GPIO_PIN_5;
     SPI1_Transmit(cmd, 1);
     status = SPI1_ReceiveByte(); // Get status register value
     GPIOB->BSRR = GPIO_PIN_5;
  } while (status & 0x01); // Loop while Write-In-Progress bit is set
}
/**
* @brief Erase a 4KB sector at a given address
*/
void erase_sector(uint32_t address)
{
  uint8_t cmd[4] = {
     0x20, // Sector Erase command
     (address >> 16) & 0xFF,
     (address >> 8) & 0xFF,
     address & 0xFF
  };
  write_enable();
  GPIOB->BRR = GPIO_PIN_5;
  SPI1_Transmit(cmd, 4);
  GPIOB->BSRR = GPIO_PIN_5;
  wait_until_not_busy();
}
```

MEM.h

```
/*
* MEM.h
*
*  Created on: Jun 5, 2025
*      Author: jrosu, bdaly
*/
#ifndef MEM_H_
#define MEM_H_
#include "main.h"
#include "SPI.h"
#include "stm32l4xx.h"
#define CMD_WRITE_ENABLE   0x06
#define CMD_READ_STATUS1   0x05
#define CMD_PAGE_PROGRAM   0x02
#define CMD_READ_DATA      0x03
#define CMD_SECTOR_ERASE   0x20
#define CMD_RESET_ENABLE   0x66
#define CMD_RESET_DEVICE   0x99
#define CMD_SECTOR_ERASE           0x20
#define CHIP_ERASE                                  0xC7
#define CS GPIOB
void GPIOB_Config(void);
void modify_sr1();
void modify_sr2();
void modify_sr3();
void global_unlock(void);
void reset(void);
void write_byte(uint32_t address, uint8_t data);
uint8_t read_byte(uint32_t addr);
void chip_erase(void);
void MEMCONFIG();
#endif /* INC_MEM_H_ */
```