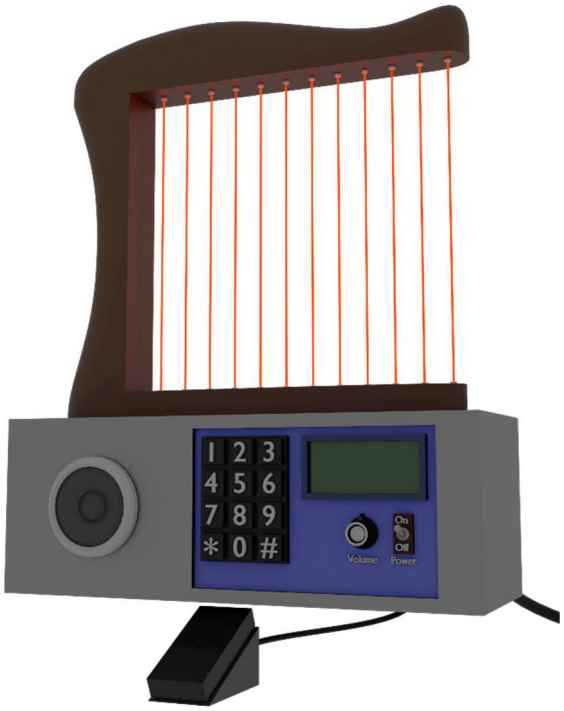
The Innovative Laser Harp (Build and Design)



Rohnin Menezes, Jahziel Ortega, Kathryn Talabucon
Electronics Engineering Technology Students
MacPhail School of Energy
Southern Alberta Institute of Technology

Source: Primary

The Innovative Laser Harp
(Build and Design)

Prepared for

Rysen Jordan, MacPhail School of Energy Academic Chair
SAIT

TAC Governing Board

Prepared by

Rohnin Menezes, Jahziel Ortega, Kathryn Talabucon
ENT Students
MacPhail School of Energy
SAIT

Requested by

Iouri Kourilov, PROJ-354 Instructor

Heather Wilson, PROJ-354 Instructor

15 April 2020

Rohnin Menezes, Jahziel Ortega, and Kathryn Talabucon
Electronics Engineering Technology Students
SAIT
Calgary, AB T2M 0L4

April 15, 2020

Rysen Jordan
MacPhail School of Energy Academic Chair
SAIT
Calgary, AB T2M 0L4

CC:
TAC Governing Board
10 Four Seasons Place
Suite 404
Toronto, ON M9B 6H7

Mr. Jordan:

We are submitting to you our Laser Harp (Build and Design) as part of our Capstone project. The purpose of writing this report is to give us an opportunity to demonstrate our skills and knowledge obtained throughout our ENT program.

Thank you for the opportunity to develop valuable skills in electronics design. With the help of our instructors, we have completed this project as intended. Despite the difficulties imposed by the COVID-19 pandemic, we persevered in order to ensure that this report is the best reflection of our abilities attained during our time at SAIT. The full design process, from Requirements Elicitation to Design, has been documented. Thank you for granting us this experience.

Sincerely,

Rohnin Menezes, Jahziel Ortega, Kathryn Talabucon
ENT Students
SAIT
rohninmenz@gmail.com; jdortega18@gmail.com; kathryntalabucon2@gmail.com

Executive Summary

This report details the full design process undergone to create a playable electronic Laser Harp. This harp contains the ability to play electronically generated notes with laser beams in place of strings. Songs can be recorded and stored for future playback, and a foot pedal will be used to manipulate the **octave**.

This project is to be completed as part of the **Electronics Engineering Technology** program at SAIT, in order to demonstrate the skills attained during the program.

The report details the many steps taken to create the final product, including Requirements Elicitation, Requirements Analysis, Requirements Specification, and the Design.

This report highlights the full design process undergone for the hardware design and prototyping phase, including the creation of the Human Machine Interface (HMI), the design of circuit schematics and PCBs, and the programming of software for the microcontroller.

In addition to explaining the processes undergone, this report also details the goals and objectives that drives the project, along with the challenges and problems we encounter.

As a direct result of the COVID-19 pandemic, SAIT was closed, and the classes were delivered online. Under those circumstances, the Laser Harp could not be constructed as planned. Despite this, we accomplished as much of the design process as possible. Despite the unforeseen circumstances, the results achieved from this project fully demonstrated the skills obtained from the ENT program.

Table of Contents

List of Illustrations

Figures

Tables

1.0 INTRODUCTION

This report is an introduction section for a Laser Harp instrument.

1.1 Purpose

The purpose of this project is to create a playable electronic harp with laser beams in place of strings. Users will be able to play and record songs onto an Electronically Erasable Programmable Read-Only Memory (EEPROM). The user can change the **octave** using a foot pedal.

1.2 Background

This project is completed by the 2nd year Electronic Engineering Technology (ENT) students in the Southern Alberta Institute of Technology (SAIT). The Capstone project is one of the requirements that needs be done as part the curriculum of the ENT program. Students need to have fundamental knowledge in electronic circuit design, programming, data communications, and control systems, all of which is learned during the program. This project is made by students to enhance their skills and abilities that they learned throughout the ENT program.

1.3 Scope

This report primarily focuses on the design and construction of the Laser Harp. The finished product must incorporate sound generation, programming, and electronic design. Some design prototyping will be required before constructing the final product. Moreover, as part of the Capstone project requirements, the harp must use an **STM32 ARM Cortex Microprocessor**, raw electronics components and sensors, and self-designed PCB circuit boards. The main limitations are the cost and production time of the project. An additional restriction is that the harp requires a darker area with smoke or a fog machine to see the laser LED strings.

1.4 Methods

All information used to build this project have been provided from courses in the program. The main concepts behind this project were obtain from the following courses: “Electronic Devices and Circuits,” “Microprocessor Fundamentals,” and “Wireless Communications.” The calculations and prototyping have been done over the reading week. The final product and this report must be completed at least two weeks before the submission.

1.5 Preview

The Elicited requirements and the problem domain description of this report explain the goals in mind when creating this project. The specification section briefly describes the connections of each subsystem. Furthermore, the system design section focuses on the construction and design of the hardware and software. Finally, the Costs and Required Resources section discusses the breakdown of pricing and supplies used to accomplish the goal of the project.

2.0 STAKEHOLDERS

List of stakeholders that will be interested to buy or invest in this project:

- Music Instructors
- Music Students
- Electronics Students
- Musicians
- Music Store Owners

3.0 ELICITED REQUIREMENTS (RAW REQUIREMENTS)

These requirements represent some of the basic specifications to be considered when designing the device. Listed below are the elicited requirements for the Laser Harp:

- The Laser Harp should generate a sound when one of the laser beams is blocked
- The Laser Harp should have a foot pedal to control the **octaves**
- The Laser Harp should have 12 laser diodes
- The Laser Harp should have a keypad to be able to record, save, and erase songs.
- The Laser Harp should be able record 20 songs.
- The Laser Harp should use the ARM Cortex MCU from STM manufacturer.
- The Laser Harp should be powered from a power supply.
- The Laser Harp size should be 60x20x80 (in centimeters).
- The Laser Harp should display the output information from the MCU
- The Laser Harp should possess a speaker to amplify sound.
- The Laser Harp should have an EEPROM to store recorded songs.
- The Laser Harp should operate within 0 through 50 degrees, Celsius.
- The development time should not exceed 4 months.
- The development cost should not exceed $1000 CAD
- The sounds from the harp should follow the relative shape shown in Figure 1 and 2.

In order to gain a basic understanding of audio signal generation, we analyzed the electrical waveform generated by playing notes on an electric guitar. The following figures show the waveform of a low E string, and an A string note B, captured using an oscilloscope.

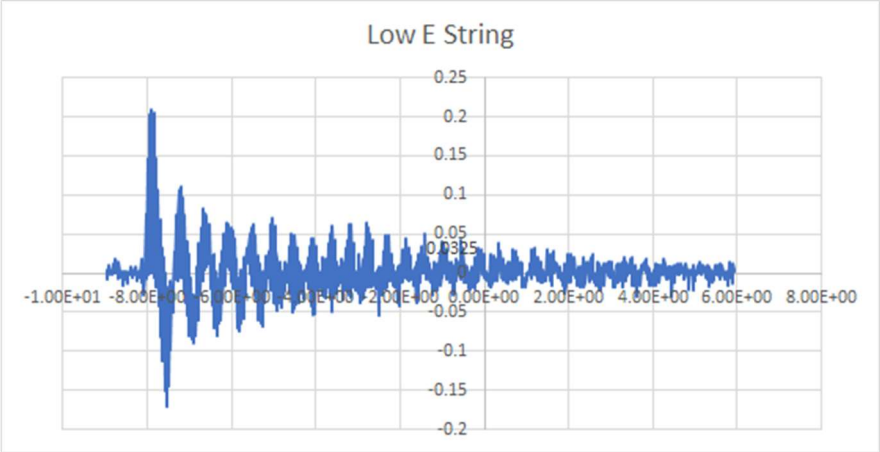


Figure 1: Waveform of a Low E string, open string.

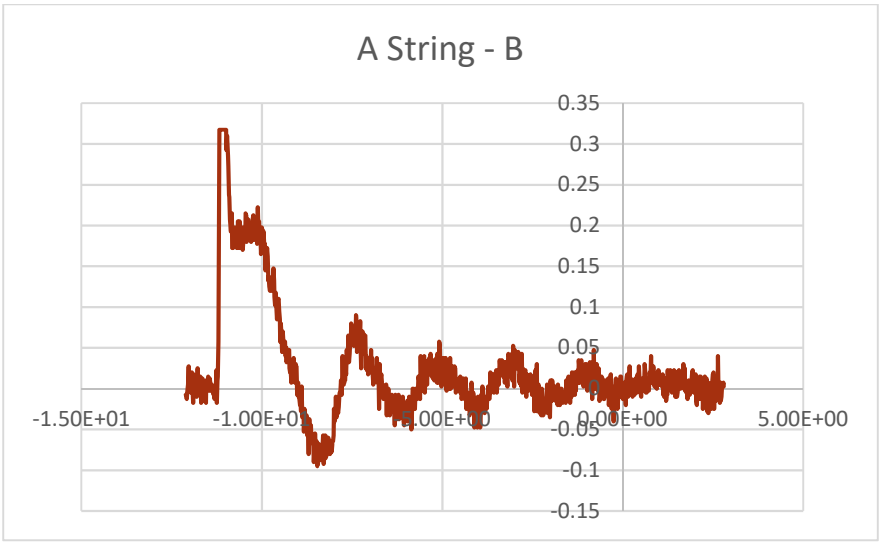


Figure 2: Waveform of the A string, second fret.

From Figure 1 and 2, we assessed that in order to generate different notes, we would have to design a system capable of generating waveforms with controllable frequency, and an amplitude that could decay exponentially over time.

4.0 PROBLEM DOMAIN DESCRIPTION

The problem domain defines the series of subsystems comprising the Laser Harp, and the relationships and interfaces between them.

4.1 Problem Frame Diagram

The problem frame diagram is illustrated below in Figure 3. The diagram contains the subsystem's relationship with one another.

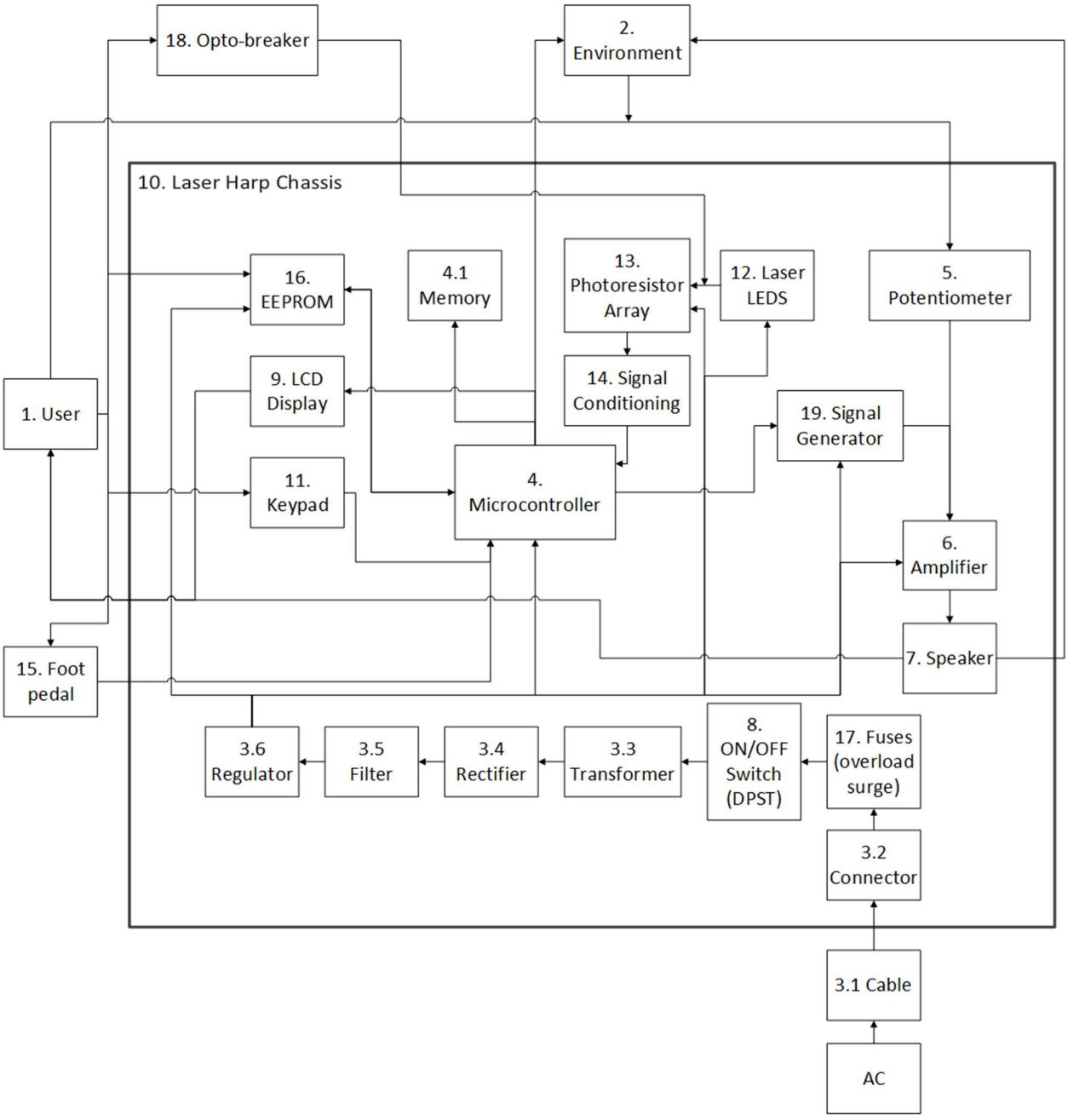


Figure 3: Problem Frame Diagram of the Laser Harp

4.1.1 Problem Frame Sub-domains

P1. User

User needs to have the following properties:

- Fingers to use the keypad, change the volume of the Laser Harp
- Feet to push the foot pedals and change the Laser Harp's **octave**

User has the following responsibilities:

- Turn on the instrument
- Swap fuses
- Save and erase the recorded songs
- Change the volume of the Laser Harp
- Select the desired function and display it on the LCD display
- Block the laser beams

P2. Environment:

- May receive EFM radiation Should be protected from EMF radiation caused by the harp and vice versa
- May cause thermo-effects on harp operation
- May cause damage if the harp is dropped
- May cause damage to harp from accidentally spilled liquids

P3. Power supply:

- Able to supply the max current and voltage from AC
- Powers the MCU
- Powers the Amplifier
- Powers the Laser LEDs
- Powers the **photoresistors**
- Powers the LCD Display

P4. MCU ARM Cortex-M4 by STMicroelectronics:

- Receives power from the dedicated power supply
- Obtains input from the keypad, **photoresistors**, and foot pedals
- Outputs sound signals to the volume control potentiometer
- Outputs information to the LCD screen
- Saves and erases data in the EEPROM
- The MCU will be initialized from data saved in its Flash memory
- Sends data for the sound frequency from the MCU as an Analog Voltage signal
- Sends data for the sound volume from the MCU as an Analog Voltage signal

P5. Potentiometer:

- Changes the desired volume based on user input
- Outputs attenuated signal to the amplifier

P6. Amplifier:

- Receives power from the dedicated Power Supply
- Obtains input from the MCU through the potentiometer
- Outputs amplified signal to the Speaker

P7. Speaker:

- Receives an input signal from the Amplifier
- Converts the electrical signal into sound
- The speaker should send signal to the user

P8. ON/OFF switch:

- Should enable/disable the power supply

P9. LCD screen:

- Receives power from the Power Supply
- Able to receive information from the MCU based on user input to the keypad
- Displays information to the user

P10. Chassis:

- Encloses the Laser Harp components, providing protection from the environment

P11. Keypad

Keypad properties:

- Keypad buttons

Keypad has the following responsibilities:

- Allows the user to save, load, and erase songs
- Sends input data to the MCU

P12. Laser LEDs:

- Activated and powered by the MCU
- Pointed at the **photoresistors**, changing their resistance values

P13. Photoresistor Array:

- Sends data signals to the MCU when the laser LEDs are blocked

P14. Signal Conditioning:

- Prevents the photoresistor from going over the current limit
- Converts the voltage across the photoresistor into an analog signal readable by the MCU

P15. Foot Pedal:

- Should be able to change the **octave** of the sound
- Should be able to send signals to the MCU

P16. EEPROM:

- Should be able to send and receive data signals from the MCU
- Should be able to store and erase data

P17. Fuses (overload surge):

- Protects the power supply from overvoltage
- Can be replaced by the user

P18. Opto-breaker:

- Any entity that breaks the laser beam between the LEDs and **photoresistors** (e.g. The user's fingers)

P19. Signal Generator:

- Powered by the Power Supply
- Generates sound signals to be sent to the Amplifier.
- Receives data for the sound frequency from the MCU as an Analog Voltage signal
- Receives data for the sound volume from the MCU as an Analog Voltage signal

4.1.2 Shared Phenomena

- The shared phenomena of the power button and the user is when user presses the button to turn on the Laser Harp.
- The shared phenomena between the user's finger and the keypad is when the user presses the keypad is the MCU function call.
- The shared phenomena between the user and the laser LED is when the user touches the beam is the measured light intensity.
- The shared phenomena between the power supply and the MCU is when the power supply sends current to the MCU.
- The shared phenomena between the user and the foot pedal is when the user presses the foot pedal to change the **octave** of the harp.
- The shared phenomena between the user and potentiometer is when the user's finger twist the potentiometer to the volume.

4.1.3 Action

The user needs to plug the Laser Harp.
The user needs to push the power supply switch on/off.
The user needs to press the keypad to select functions such as play, record, stop etc.
The user needs to touch the light emitted to produce a sound.
The user needs to step on the foot pedals to change the **octave** and sustain the notes.
The user needs to save and save and erase recorded songs.

4.1.4 Distortions and Delays

The Harp should not have a delay greater than 20µs between a laser being blocked and the sound being played.

4.1.5 Valid Commands

The power supply is pluged in by the user.
The foot pedal stepped on by the user.
The pins on the keypad will assigned differently based on the LCD menu.
The laser beam is blocked by the user.

4.1.6 Events Description based on the Problem Domain

Device is plugged-in/unplugged.
Device is turned on/off switch.
Laser beam is blocked.
A song is recorded.
A song is saved to the EEPROM.
A song is erased.
“Change **Octave**” foot pedal is pressed.
Device is unplugged/plugged in.
Dropped the instrument.
Laser LED blown.
No more storage to store songs.
Accidentally spilled liquids on the instrument.

4.2 Context Diagram

The context diagram is illustrated below in Figure 4. The diagram contains the domains directly connected to the MCU.

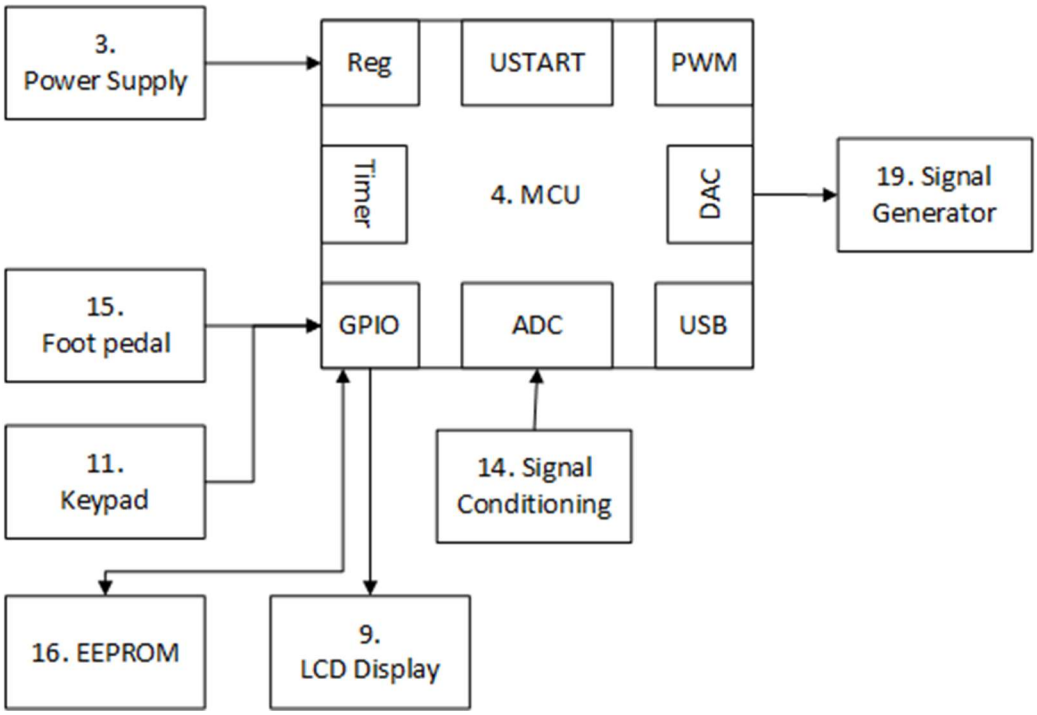


Figure 4: Context Diagram of the Laser Harp

4.2.1 Event-Responses based on the Context Diagram

The Laser Harp's event responses descriptions based on the context diagram is provided below in Table 1.

Table 1: Event Response Description based on the Context Diagram

Event #	Source	Destination	Event Diagram	Response Description	Signals Involved
1	3	4	Power Switch is turned on	A) The initial data is loaded from memory. B) The LCD screen displays the main menu.	DC
2	13	4	A Laser is blocked	A) The MCU selects the note to be sent based on the Laser blocked. B) The MCU sends a sound signal to the Signal Generator (Event 2).	Analog
3	4	19	A Sound signal must be sent.	A) The MCU loads data for selected note's frequency and volume from memory. B) The MCU sends the data to the Signal Generator.	Analog
4	11	4	The Record command is selected	A) The MCU records each note played along with its relative duration and the distance to the next note.	Digital
5	11	4	The Play command is selected	A) The MCU plays the selected song, outputting the audio to the Signal Generator.	Digital
6	11	4	The Select command is selected	A) The MCU loads the index of the current song.	Digital
7	11	4	The Delete command is selected	A) The MCU erases the song at the currently selected index from memory.	Digital
8	15	4	The Foot pedal is pressed	A) The MCU changes to the higher **Octave**.	Digital
9	15	4	The Foot pedal is released	A) The MCU reverts to the default **Octave**.	Digital

5.0 REQUIREMENTS

This section outlines the various elicited requirements for the Laser Harp that have been clearly defined and categorized.

5.1 General Design Constraints

The general design constraints of the Laser Harp are provided below in Table 2.

Table 2: General Design Constraints Description

GDC #	Software/Hardware	Description	Related FRs
GDC1	HW	The LCD display is to be used.	N/A
GDC2	HW	MCU ARM Cortex by STM is to be used.	N/A
GDC3	HW	A power switch is to be used.	N/A
GDC4	SW	The CrossWorks IDE is to be used.	N/A
GDC5	HW	The operating temperature range is from 0◦ to 50◦ Celsius.	N/A
GDC6	HW	The Laser Harp size should be 60x80x20 (in centimeters).	N/A
GDC7	HW	A keypad is to be used.	N/A
GDC8	HW	The Laser Harp should be able to function in all lighting conditions.	N/A
GDC9	HW	The Laser Harp should have a power supply.	N/A
GDC10	HW	The Laser Harp should possess a speaker to output sound.	N/A
GDC11	HW	The Laser Harp should possess an amplifier circuit to meet the speaker's power requirements.	N/A
GDC12	HW	The Laser Harp should have a pedal system	FR2
GDC13	HW	The power supply for the Laser Harp should be supplied by a 120 Vrms not exceeding 30W power consumption.	F12
GDC14	HW	The Laser Harp should have an EEPROM to record song data	N/A

5.2 Functional Requirements

The functional requirements of the Laser Harp are provided below in Table 3. The functional requirements outline what actions the harp should be capable of performing.

Table 3: Functional Requirements Description

FR#	**Software/ Hardware**	**Description**	**Related GDC**
FR1	HW/SW	The Laser Harp should generate a sound when one of the laser beams is blocked.	N/A
PR1.1	HW	The Laser Harp should produce 12 notes.	N/A
FR2	HW	The Laser Harp should switch between a lower and higher **Octave**.	GDC12
PR2.1	HW	The foot pedals should not exhibit functional damage after being pressed 50,000 times.	GDC12
FR3	HW	The Laser Harp should sustain a pressed and release note.	GDC2
FR4	HW	The generated notes should be displayed.	GDC1
PR4.1	HW	The sustain time should be within 10 ms and 3 seconds.	N/A
FR5	SW	The Laser Harp should record songs on the EEPROM.	GDC14
PR5.1	SW	The number of the recorded songs should be 20.	GDC14
FR6	SW	The Laser Harp should be able to display the list of recorded songs.	GDC1
PR6.1	SW	At least two songs should be displayed with their sequential numbers.	GDC1
FR7	HW	The Laser Harp should erase the unwanted song.	GDC14
FR8	HW	The Laser Harp should be able to function in all lighting conditions.	N/A
FR9	HW	The Laser Harp should receive power from a wall outlet	N/A

5.3 Commercial Constraints

C.1 The process of making the Laser Harp should be finished within 4 months.

C.2 The Laser Harp development cost must not exceed $1000 CAD.

C.3 The Laser Harp raw components should be available in Digi-key and Amazon.

C.3.1 Raw components from Digi-Key should be ordered and shipped in a month before the construction of the project.

6.0 PRODUCT SPECIFICATIONS

This section contains the functional block diagrams and their descriptions for the project. They outline how the subsystems operate and interface with each other from a technical standpoint.

6.1 Laser Harp Functional Diagram

The functional block diagram of the Laser Harp is illustrated below in Figure 5.

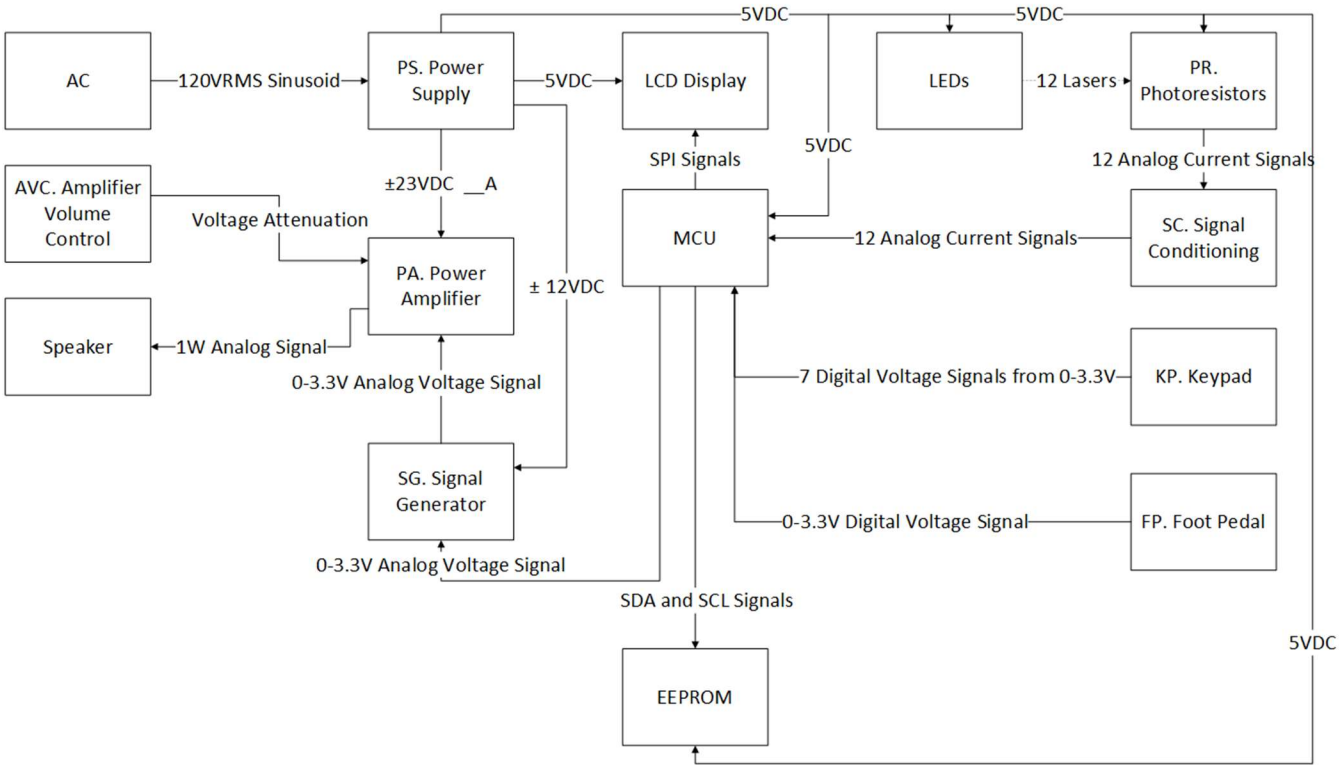


Figure 5: Laser Harp Functional Block Diagram

6.1.1 Laser Harp Functional Block Description

The functional description of the Laser Harp in Table 4 outlines the various subsystems present within the harp and their purposes.

Table 4: Laser Harp Functional Description

Functional Block	**Functions**
AC	Supply 120 VRMS to the power supply.
PS. Power Supply	Supply specified voltages and currents to MCU and other peripherals
PA. Power Amplifier	Amplify the power from the power supply to create signal and send it to the speaker
LCD Display	Display the function options of the Laser Harp
LEDs	Receives power to from the power supply
PR. **Photoresistors**	Receives power from the supply and send signal to the signal conditioning

Functional Block	Functions
EEPROM	It's used to erase and store songs.
AVC. Amplifier Volume Control	To control the voltage of the amplifier in order to control the volume
Speaker	Convert the signal from the power amplifier into a sound
SG. Signal Generator	An Oscillator that controls the frequency of each note.
SC. Signal Conditioning	Filtering the analog signal from the **photoresistors** to MCU
KP. Keypad	Entering the desired function option and send it to the MCU
FP. Foot Pedal	Changing the **octave** by sending signal to the MCU

6.1.2 Laser Harp Functional Signal Description

The functional signal description of the Laser Harp in Table 5 defines the signals sent between subsystems.

Table 5: Laser Harp Functional Signal Description

Source	Destination	Signal Name	Signal Description	Interface Description	Impedance
AC	PS	AC	120 VRMS Sinusoid	Power cord to screw terminals	TBD
PS	Laser LED	PS	5 VDC	2 wires terminal	TBD
PS	PR	PS	5 VDC	2 wires terminal	TBD
PS	EEPROM	PS	5 VDC	2 wires terminal	TBD
PS	MCU	PS	5 VDC	2 wires terminal	TBD
PS	EEPROM	PS	5 VDC	2 wires terminal	TBD
PS	LCD Display	PS	5 VDC	2 wires terminal	TBD
PS	PA	PS	±23 VDC	2 wires terminal	TBD
PS	SG	PS	±12 VDC	2 wires terminal	TBD
MCU	SG	MCU	0-3.3V Analog Voltage Signal	Jumper Wires	50kΩ
MCU	EEPROM	MCU	SDA AND SCL signals	Jumper Wires	50kΩ
MCU	LCD Display	MCU	SPI Signals (0VDC/3.3VDC)	Jumper Wires	50kΩ
PA	Speakers	PA	1W Analog Signals	2 Wires terminals	100Ω
AVC	PA	PA	Voltage Attenuation	2 wires terminal	TBD

Source	Destination	Signal Name	Signal Description	Interface Description	Impedance
SG	PA	SG	0-3.3V Analog Voltage Signal	2 wires terminal	TBD
PR	SC	PR	12 Analog current signals	2 wires terminal	20KΩ to 50KΩ
SC	MCU	SC	12 Analog current signals	Jumper Wires	TBD
KP	MCU	KP	7 Digital Voltage Signal	Header	50kΩ
FP	MCU	FP	0-3.3V Analog Voltage Signal	Header	TBD

6.2 Power Supply Functional Diagram

The functional block diagram for the Power Supply is illustrated below in Figure 6.

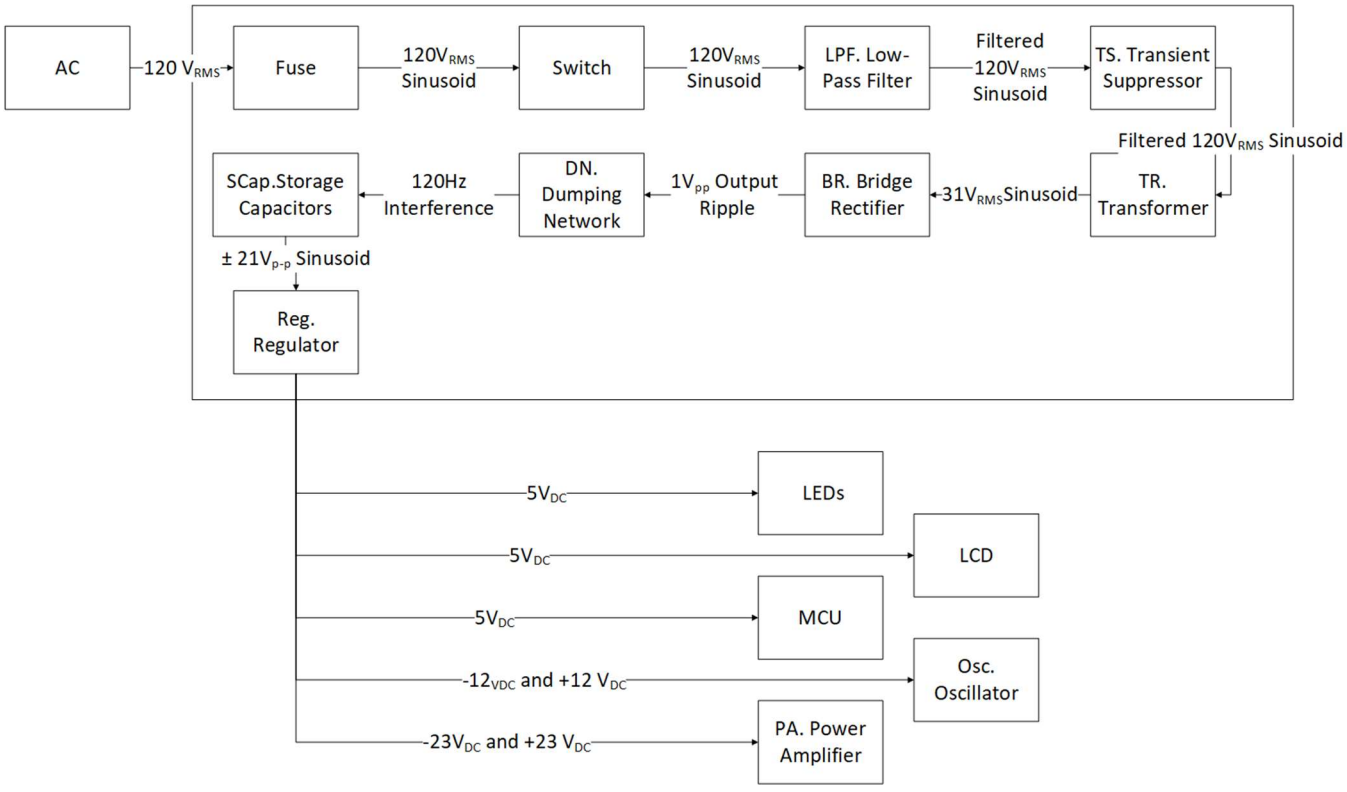


Figure 6: Power Supply Functional Diagram

6.2.1 Power Supply Functional Block Description

The functional description in Table 6 provides the characteristic of the signal going in the Power Supply.

Table 6: Power Supply Functional Description

Functional Block	**Functions**
AC	Provide 120VRMS to the power supply.
Fuse	Prevents the power supply from overcurrent.
Switch	The power entry of the power supply, which Allows to pass the current through the Low-Pass filter
LPF. Low- Pass Filter	Eliminates the possible radiation of radiofrequency interference (RFI) from the power line. It also allows current to pass the filtered current through the Line-Voltage Capacitor.
LVC. Line Voltage Capacitor	Provides a suitable low ripple voltage and enough voltage rating to handle the worst- case line voltage combination with no load.
TS. Transient Suppressor	It is used to block the terminal voltage to go over the specified limit voltage like a bi-directional high-power zener.
BR. Bridge Rectifier	The signal is converted into a sinusoid for the positive and negative half-wave rectified waveform.
DN. Damping Network	An RC series that generates 120 Hz interference in order to have a sharp spike periodically.
SCAP. Storage Capacitor	Converts a half-wave rectified sinusoid into a rippling DC voltage.
Reg. Regulator	Converts a rippling DC voltage into a stable DC voltage. Also provides power to LCD, LED, Signal Generator and Power Amplifier.

6.2.2 Power Supply Functional Signal Description

The functional signal description in Table 7 provides the signal direction in the Power Supply.

Table 7: Power Supply Functional Signal Description

Source	**Destination**	**Signal Name**	**Signals Description**	**Interface Description**	**Impedance**
AC	Fuse	AC	120 VRMS Sinusoid	Power cord to terminal	TBD
Fuse	Switch	Fuse	120 VRMS Sinusoid	2 Wire screw terminals	TBD
Switch	LPF	Switch	120 VRMS Sinusoid	2 Wire screw terminals	TBD

Source	Destination	Signal Name	Signals Description	Interface Description	Impedance
LPF	TS	LPF	120 VRMS Sinusoid	2 Wire screw terminals	TBD
TS	TR	TS	120 VRMS Sinusoid	2 Wire screw terminals	TBD
TR	BR	TR	120 VRMS Sinusoid	2 Wire screw terminals	TBD
BR	DN	BR	1 Vpp output Ripple	2 Wire screw terminals	TBD
DN	SCap	DN	24.8 VDC	2 Wire screw terminals	TBD
SCap	Reg	SCap	24.8 VDC	PCB Traces	TBD

6.3 Power Amplifier Functional Diagram

The functional block diagram for the Power Amplifier is illustrated below in Figure 7.

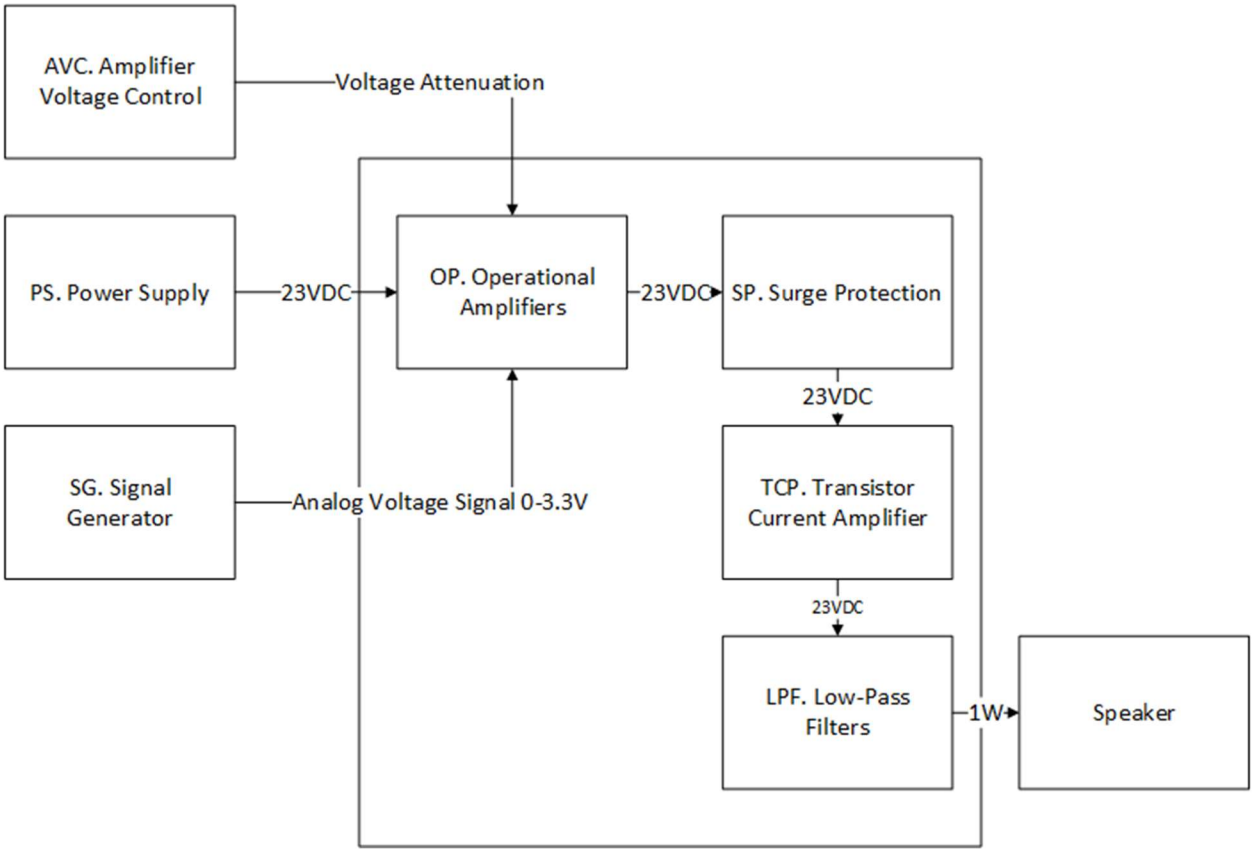


Figure 7: Power Amplifier Functional Diagram

6.3.1 Power Amplifier Functional Block Description

The functional block description in Table 8 defines the various subsystems within the Power Amplifier and their purposes.

Table 8: Power Amplifier Functional Description

Functional Block	**Functions**
OP. Operational Amplifiers	Amplify a triangle wave that is coming from the SG by increasing the input voltage.
SP. Surge Protection	Prevents damage to the components. SP prevents current from going the opposite direction of where it should be going.
TCP. Transient Current Amplifier	Takes the from the output current of the OP and increases the current through the TCA.
LPF. Low-Pass Filter	Eliminates the possible radiation of radiofrequency interference (RFI) from the power line. It also allows current to pass the filtered current through the Line- Voltage Capacitor.

6.3.2 Power Amplifier Functional Signal Description

The functional signal description in Table 9 defines the signals sent between subsystems within the Power Amplifier.

Table 9: Power Amplifier Functional Signal Description

Source	**Destination**	**Signal**	**Signals Description**	**Interface Description**	**Impedance**
AVC	OP	AVC	Voltage Attenuation	2 Wire screw terminals	TBD
PS	OP	PS	23 VDC	2 Wire screw terminals	TBD
SG	OP	SG	Analog Voltage signal 0-3.3 V	Jumper wires	TBD
OP	SP	OP	+23 VDC to - 23 VDC	PCB Traces	TBD
SP	TCP	SP	+23 VDC to - 23 VDC	PCB Traces	TBD
TCP	LPF	TCP	+23 VDC to - 23 VDC	PCB Traces	56KΩ
LPF	Speaker	LPF	+23 VDC to - 23 VDC	PCB Traces	56KΩ

6.4 MCU Connections Functional Diagram

The MCU's functional block diagram is illustrated below in Figure 8.

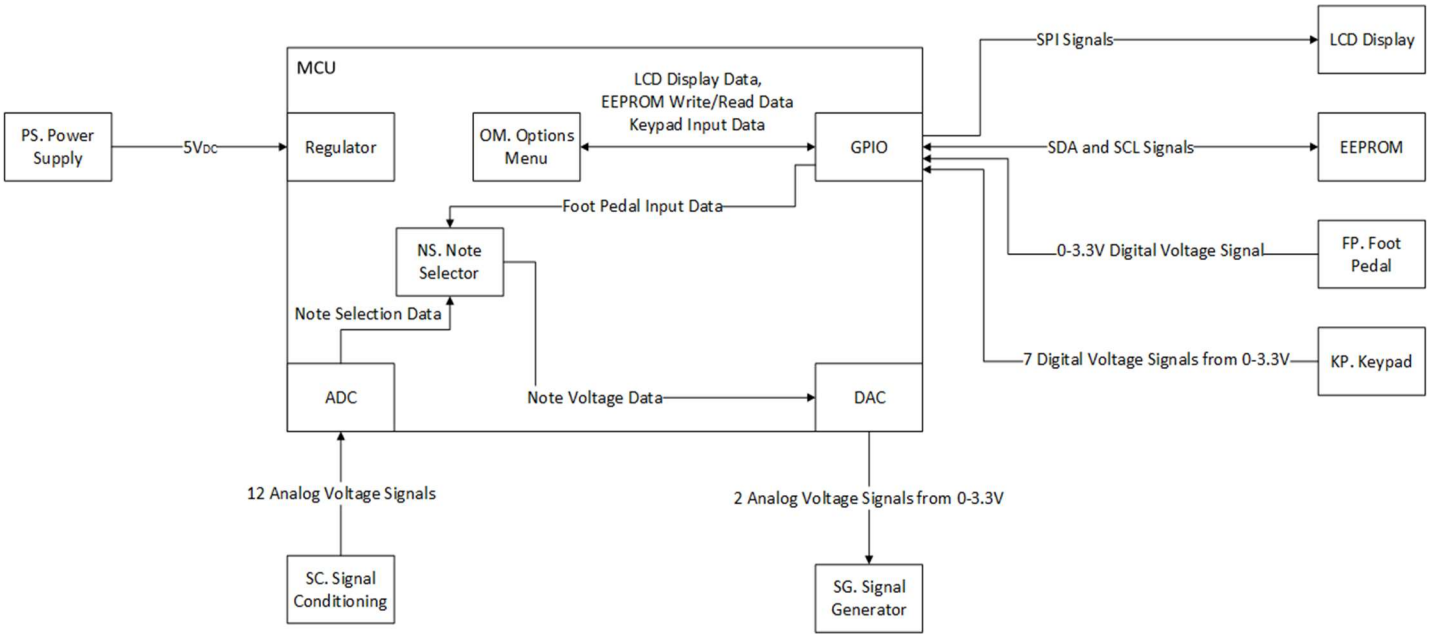


Figure 8: MCU Functional Diagram

6.4.1 MCU Block Description

The functional description in Table 10 defines the different elements of the MCU and their functions.

Table 10: MCU Functional Description

Functional Block	**Functions**
Regulator	Receives Power and supplies it to the MCU.
GPIO	Sends/Receives data signals from/to the EEPROM. Sends Display data to the LCD Display. Receives Input data from the Keypad. Receives Input data from the Foot pedal.
ADC	Receives data on the lasers being blocked from Signal Conditioning.
DAC	Sends data on the sound frequency and volume to the Signal Generator.
NS. Note Selector	Loads the frequency and volume of a note based on input from an ADC and the Foot Pedal, then outputs the values as Analog Voltages from the DAC.
OM. Options Menu	Contains a list of actions that can be executed, including "save song", "erase song", and "play song". The menu items are displayed to the LCD screen and selected using input from the keypad. The Options Menu can write/read data in the EEPROM based on the current option selected.

6.4.2 MCU Connections Functional Signal Description

The functional signal description in Table 11 defines the signals transmitted and received by the MCU.

Table 11: MCU Functional Signal Description

Source	Destination	Signal	Signals Description	Interface Description	Impedance
PS. Power Supply	Regulator	Power	5VDC. Main Power to the MCU.	2 Wire screw terminals	Low impedance source to varying impedance destination.
SC. Signal Conditioning	ADC	Photore-sistor Voltage Data	12 Analog Voltage Signals. The data that determines which lasers are blocked at any given time.	2 Wire screw terminals	2kΩ impedance source to very high impedance destination
ADC	NS. Note Selector	Note Selection Data	18 bytes. Information on which lasers are currently blocked.	N/A	Internal data transfer.
NS. Note Selector	DAC	Note Voltage Data	3 bytes. Information on the frequency and volume of the sound currently being sent.	N/A	Internal data transfer.
DAC	SG. Signal Generator	Sound Frequency Signal	0-3.3V Analog Voltage Signal. Information on the frequency of the sound being sent.	2 Wire screw terminals	Low impedance source to very high impedance destination.
DAC	SG. Signal Generator	Sound Volume Signal	0-3.3V Analog Voltage Signal. Information on the volume of the sound being sent.	2 Wire screw terminals	Low impedance source to very high impedance destination.
OM. Options Menu	GPIO	LCD Display Data	Varying data size. Information on to be shown on the LCD display.	N/A	Internal data transfer.
GPIO	LCD Display	LCD Display Signal	SPI Signals. Signals that set the LCD Display.	Jumper Wires	Low impedance source to high impedance destination.

Source	Destination	Signal	Signals Description	Interface Description	Impedance
OM. Options Menu	GPIO	EEPROM Write/Read Data	Varying data size. Data to be Written or Read to/from the EEPROM.	N/A	Internal data transfer.
GPIO	EEPROM	EEPROM Write Signals	SDA and SCL Signals. Song data sent to the EEPROM.	N/A	Low impedance source to high impedance destination.
EEPROM	GPIO	EEPROM Read Signals	SDA and SCL Signals. Song data received from the EEPROM.	N/A	Low impedance source to high impedance destination.
FP. Foot Pedal	GPIO	Foot Pedal Input Signal	0-3.3V Digital Voltage Signal. Input data received from the Foot Pedal.	2 Wire screw terminals	Low impedance source to high impedance destination.
GPIO	NS. Note Selector	Foot Pedal Input Data	1 byte. Determines which **octave** is selected based on input from the Foot Pedal.	N/A	Internal data transfer.
KP. Keypad	GPIO	Keypad Input Signal	7 Digital Voltage Signals from 0-3.3V. Input data received from the Keypad.	Jumper wires	Low impedance source to high impedance destination.
GPIO	OM. Options Menu	Keypad Input Data	12 bytes. The data that determines which keys on the keypad are pressed at any given time	N/A	Internal data transfer.

6.5 Signal Generator Functional Diagram

The functional block diagram for the Signal Generator is illustrated below in Figure 9.

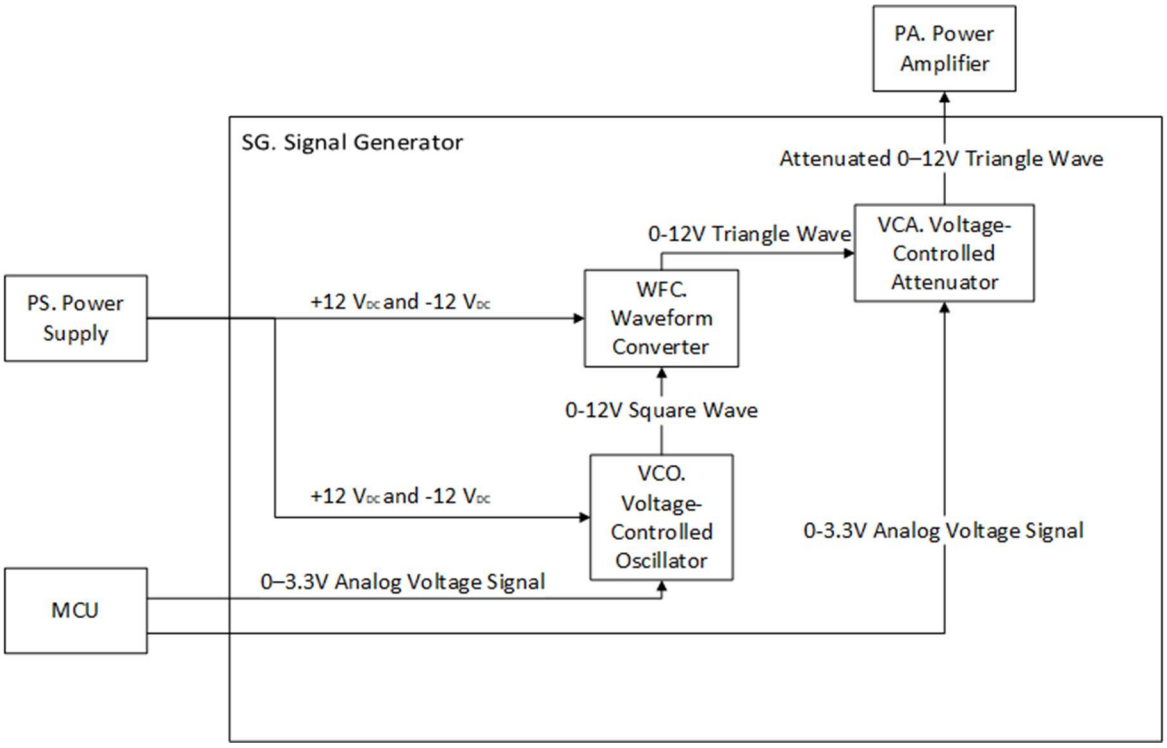


Figure 9: Signal Generator Functional Diagram

6.5.1 Signal Generator Functional Block Description

The functional description in Table 12 defines the subsystems comprising the Signal Generator and their purposes.

Table 12: Signal Generator Functional Description

Functional Block	**Functions**
VCO. **Voltage-Controlled Oscillator**	Outputs a square wave with a frequency based on the analog input voltage.
WFC. **Waveform Converter**	Composed of an integrator that converts the Output waveform of the VCO into a **sawtooth wave**.
VCA. Voltage-Controlled Attenuator	Controls the volume of the waveform by attenuating the signal based on an analog input voltage.

6.5.2 Signal Generator Block Diagram Signal Description

The functional signal description in Table 14 defines the signals sent between the subsystems of the Signal Generator.

Table 13: Signal Generator Functional Signal Description

Signal	**Source**	**Destination**	**Signals Description**	**Interface Description**	**Impedance**
VCO Power	PS. Power Supply	VCO. **Voltage-Controlled Oscillator**	+12 VDC and -12 VDC. Power to the Signal generator	2 Wire screw terminals	Low impedance source to varying impedance destination.
Waveform Converter Power	PS. Power Supply	VCO. **Voltage-Controlled Oscillator**	+12 VDC and -12 VDC. Power to the **waveform converter**	2 Wire screw terminals	Low impedance source to varying impedance destination.
Sound Frequency Signal	MCU	VCO. **Voltage-Controlled Oscillator**	0-3.3V Analog Voltage Signal. Signal defining the frequency of the sound waveform.	Jumper wires	Low impedance source to very high impedance destination.
Sound Volume Signal	MCU	VCO. **Voltage-Controlled Oscillator**	0-3.3V Analog Voltage Signal. Signal defining the volume of the sound waveform.	Jumper Wires	Low impedance source to very high impedance destination.
Square Wave	VCO. **Voltage-Controlled Oscillator**	WFC. **Waveform Converter**	0-12V Square Wave. The output signal of the VCO with a frequency set by the Sound Frequency Signal.	Jumper Wires	Low impedance source to fixed impedance destination.

Signal	**Source**	**Destination**	**Signals Description**	**Interface Description**	**Impedance**
Triangle Wave	WFC. **Waveform Converter**	VCA. Voltage-Controlled Attenuator	0-12V Triangle Wave. The output signal of the **waveform converter**.	Jumper Wires	Low impedance source to varying impedance destination.
Attenuated 0-12V triangle wave	VCA. Voltage-Controlled Attenuator	PA. Power Amplifier	Attenuated 0-12V Triangle Wave. The output sound from the Signal Generator, attenuated based on the Sound Voltage Signal.	2 Wire screw terminals	Low impedance source to very high impedance destination.

6.6 Signal Conditioning Functional Diagram

The functional block diagram for Signal Conditioning is illustrated below in Figure 10.

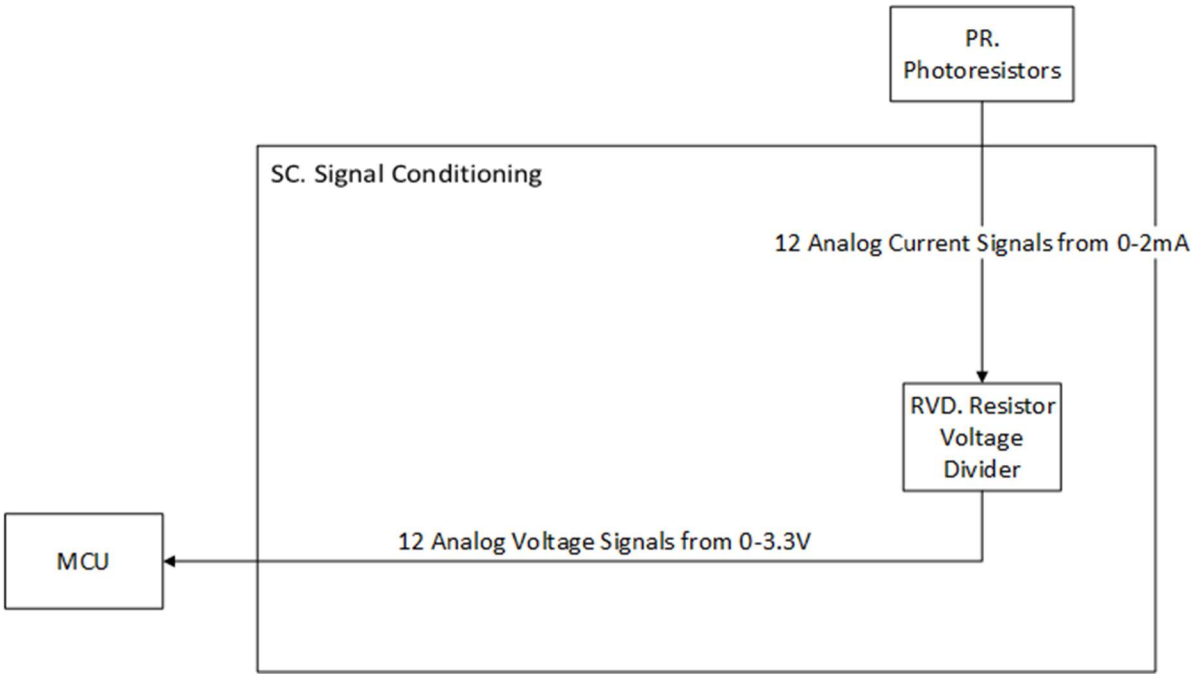


Figure 10: Signal Conditioning Functional Diagram

6.6.1 Signal Conditioning Functional Block Description

The functional description in Table 14 defines the subsystems that comprise the Signal Conditioning functional block.

Table 14: Signal Conditioning Functional Description

Functional Block	**Functions**
RVD. Resistor **Voltage Divider**	Converts the changing resistance of the **Photoresistors** into an Analog Voltage Signal that can be read by the MCU.

6.6.2 Signal Conditioning Block Diagram Signal Description

The functional signal description in Table 15 defines the signals sent between the subsystems within the Signal Conditioning functional block.

Table 15: Signal Conditioning Functional Signal Description

Signal	**Source**	**Destination**	**Signals Description**	**Interface Description**	**Impedance**
Photoresistor Current Data	PR. **Photoresistors**	RVD. Resistor **Voltage Divider**	12 Analog Current Signals from 0-2mA. Analog current signals created by the resistance of the p**hotoresistors**, which changes based on light exposure.	Jumper Wires	Varying impedance source (2-200kΩ) to a fixed impedance destination.
Photoresistor Voltage Data	RVD. Resistor **Voltage Divider**	MCU	12 Analog Voltage Signals from 0-3.3V. The output from the p**hotoresistors** converted into an analog voltage that can be read by the MCU's ADCs.	Jumper Wires	Variable impedance source (1-2kΩ) to a very high impedance destination.

7.0 HUMAN-MACHINE INTERFACE (HMI)

The human-machine interface defines the physical hardware of the harp accessible to the user, with interactable features labeled. The HMI is defined by a series of representation drawings from different angles, and descriptions of functionality for labeled parts.

7.1 Laser Harp Representation Drawing

7.1.1 Front View

The front view design of the Laser Harp is shown in Figure 11.

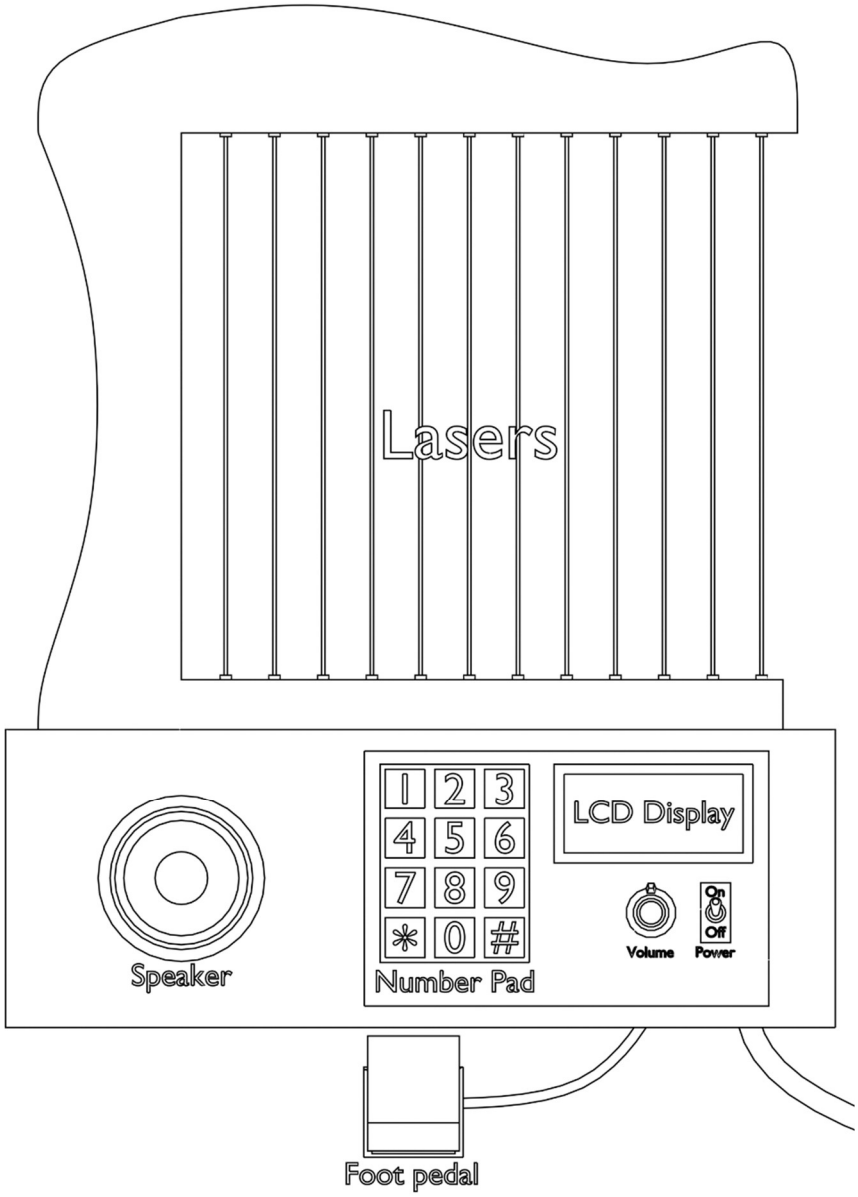


Figure 11: Front View of Laser Harp HMI

7.1.2 Back View

The back-view design of the Laser Harp is shown in Figure 12.

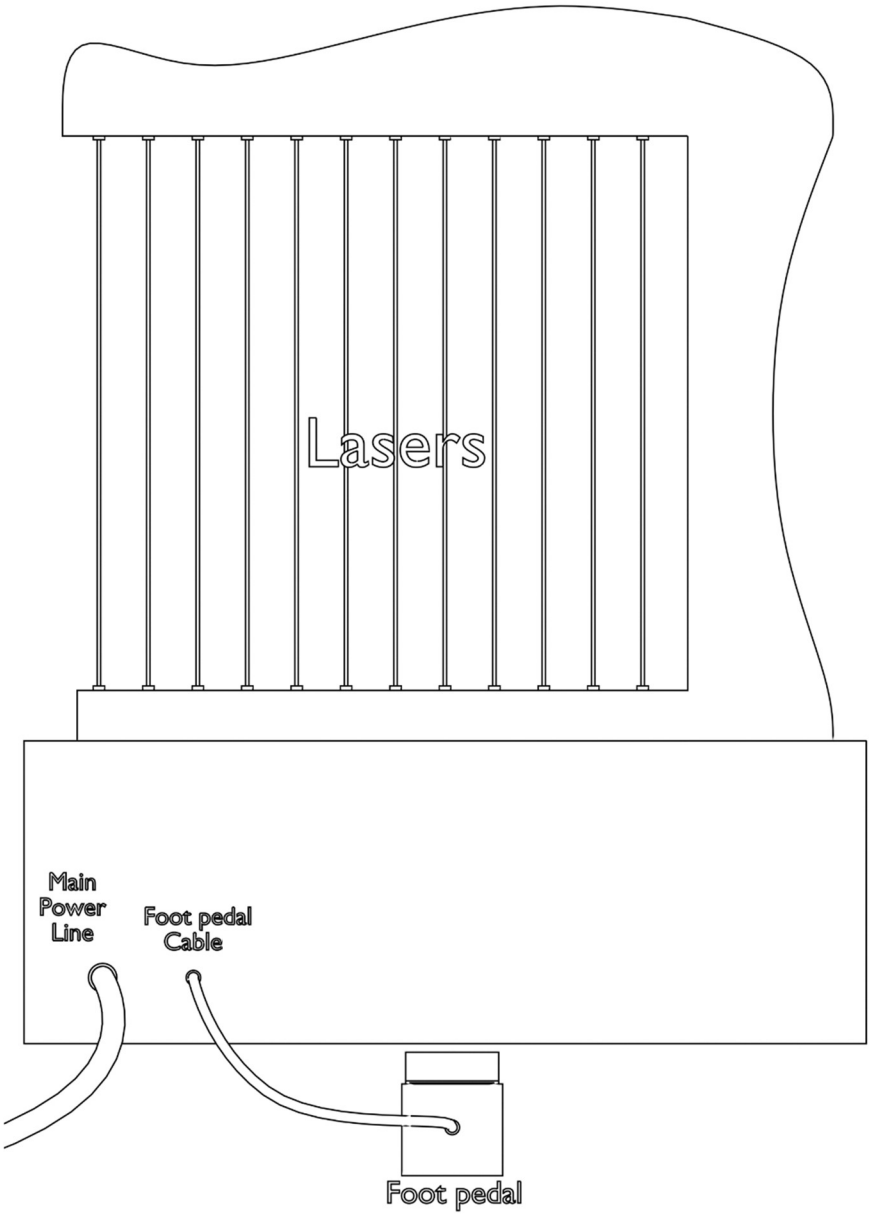


Figure 12: Back View of Laser Harp HMI

7.1.3 Top View

The top view design of the Laser Harp is shown in Figure 13.

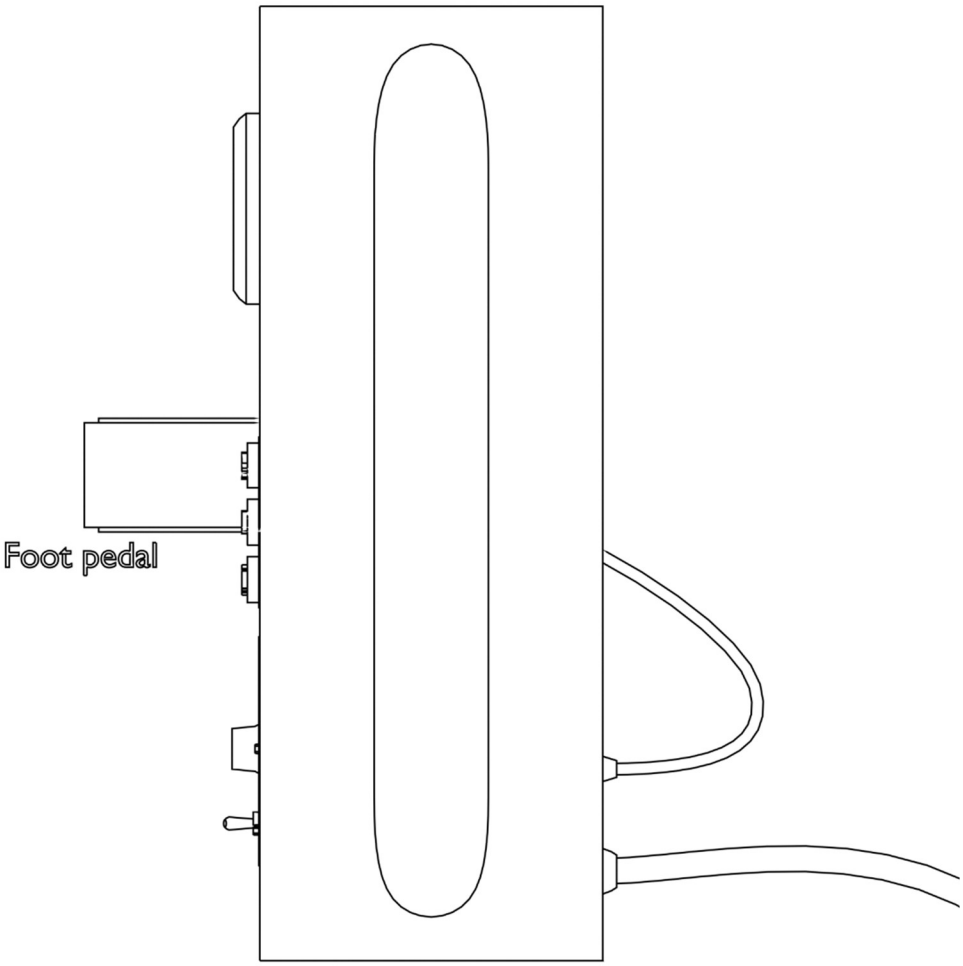


Figure 13: Top View of Laser Harp HMI

7.1.4 Left View

The left view design of the Laser Harp is shown in Figure 14.

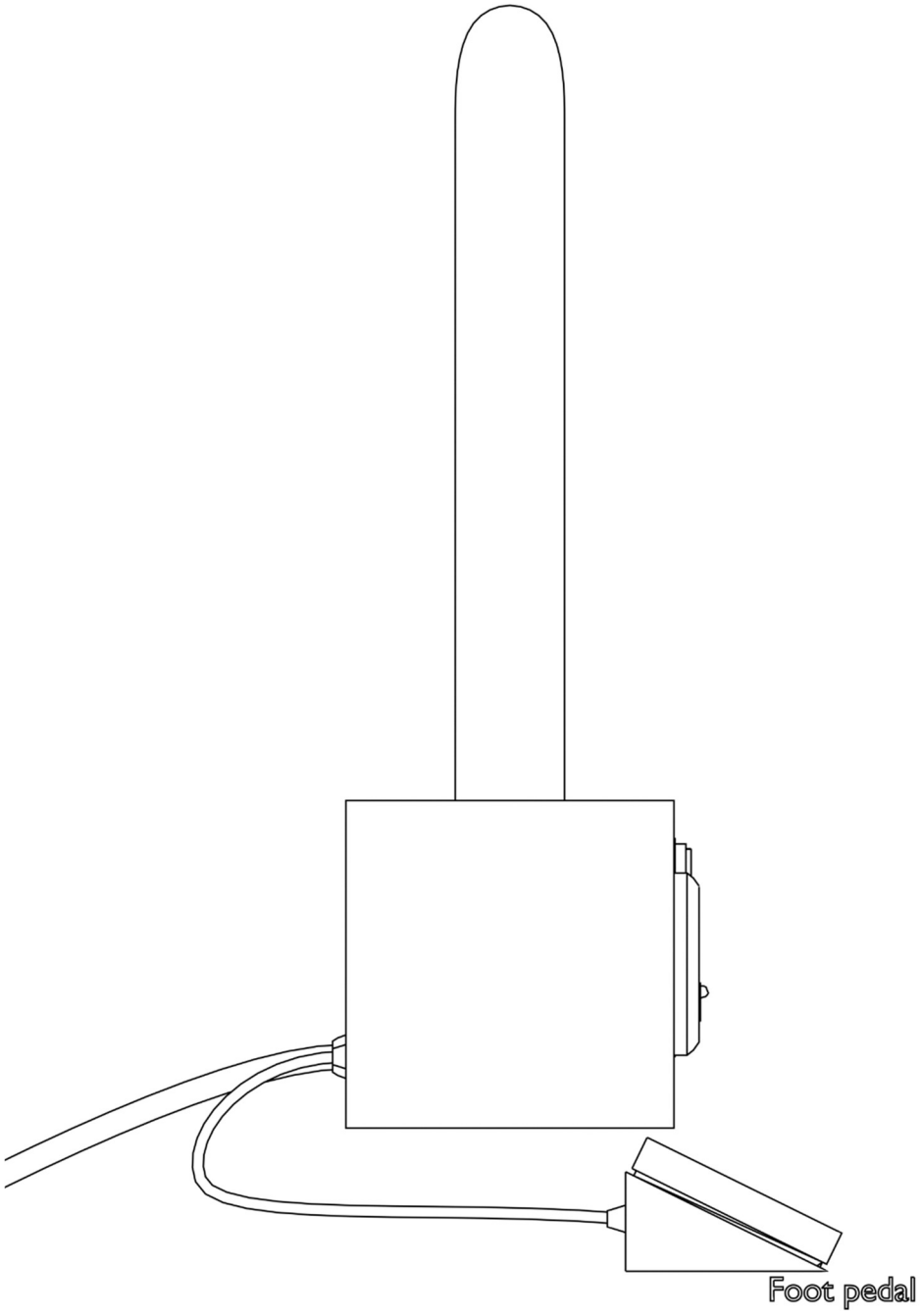


Figure 14: Left View of Laser Harp HMI

7.1.5 Right View

The right view design of the Laser Harp is shown in Figure 15.

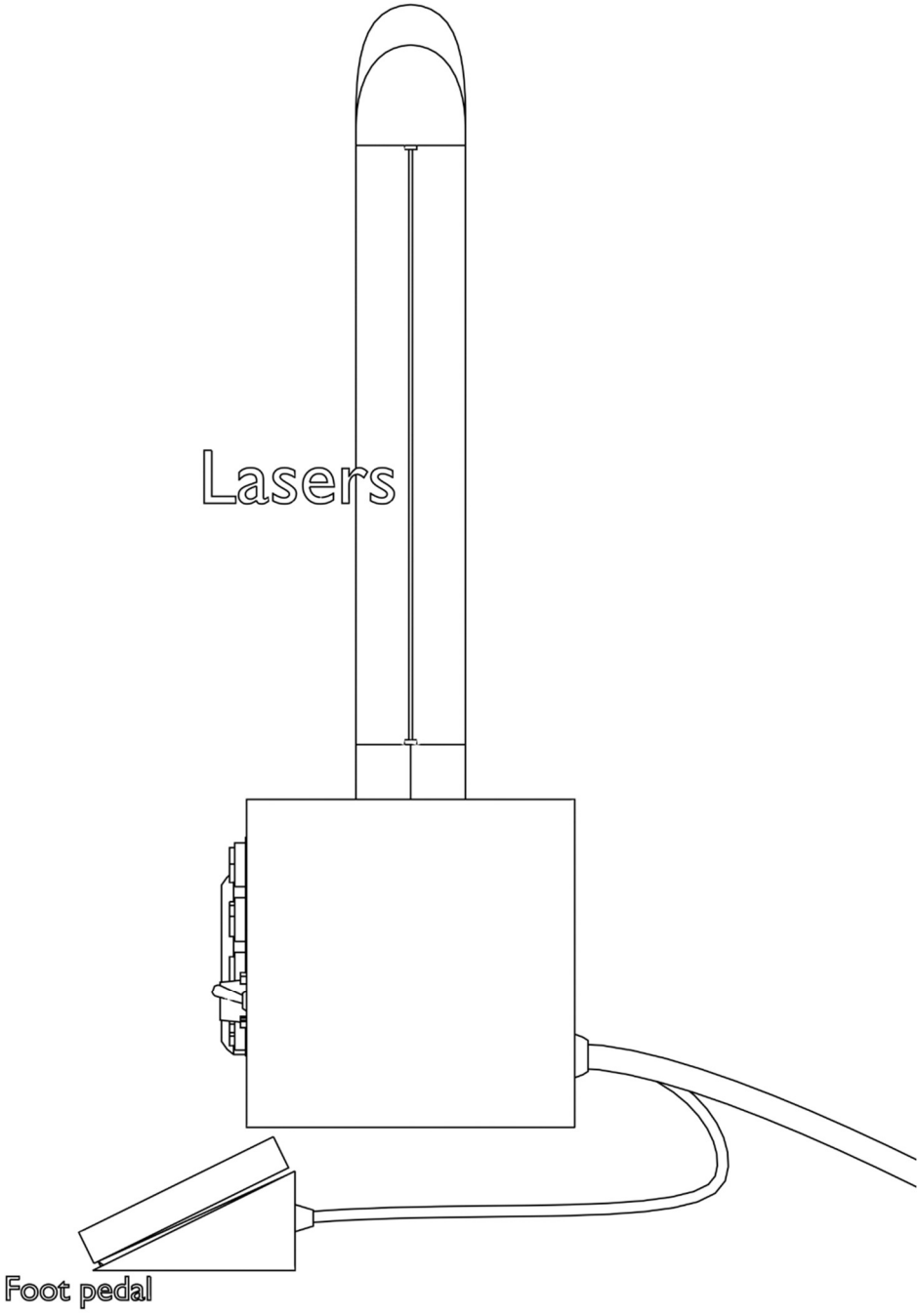


Figure 15: Right View of Laser Harp HMI

7.2 Functionality

Number Pad:

- Used to save, select, load, and delete songs.

Volume Knob:

- Adjusts the gain of the amplifier to manipulate the output volume of the harp.

Power Switch:

- Turns on/off the main power supply.

LCD Display:

- Displays the menu for selecting, saving, loading, and deleting songs.

Foot pedals:

- Swap between a higher or lower **octave**.

Foot pedal Cable:

- Connects the foot pedal to the harp.

Speaker:

- Outputs sound.

Lasers:

- Signals the MCU to apply sound when the beam is broken.

Main Power Line:

- Obtains power from a wall outlet and transfers it to the harp.

8.0 ELECTRONICS SUBSYSTEMS

This section outlines the full design process undergone for the PCBs in this project. The three circuits created were the Power Supply, Power Amplifier, and Signal Generator.

8.1 Overall Description of Electronics Subsystems

8.1.1 Power Supply Description

The Power Supply provides the required power for all subsystems within the Laser Harp.

8.1.2 Power Amplifier Description

The Power Amplifier boost the signal obtained from the Signal Generator by 3dB and outputs it to the Speaker.

8.1.3 Signal Generator Description

The Signal Generator produces a sawtooth waveform with volume and frequency based on the two DAC output signals from the MCU.

8.2 Power Supply Schematics

The final design of the 166J35 Power Supply is shown below in Figure 16.

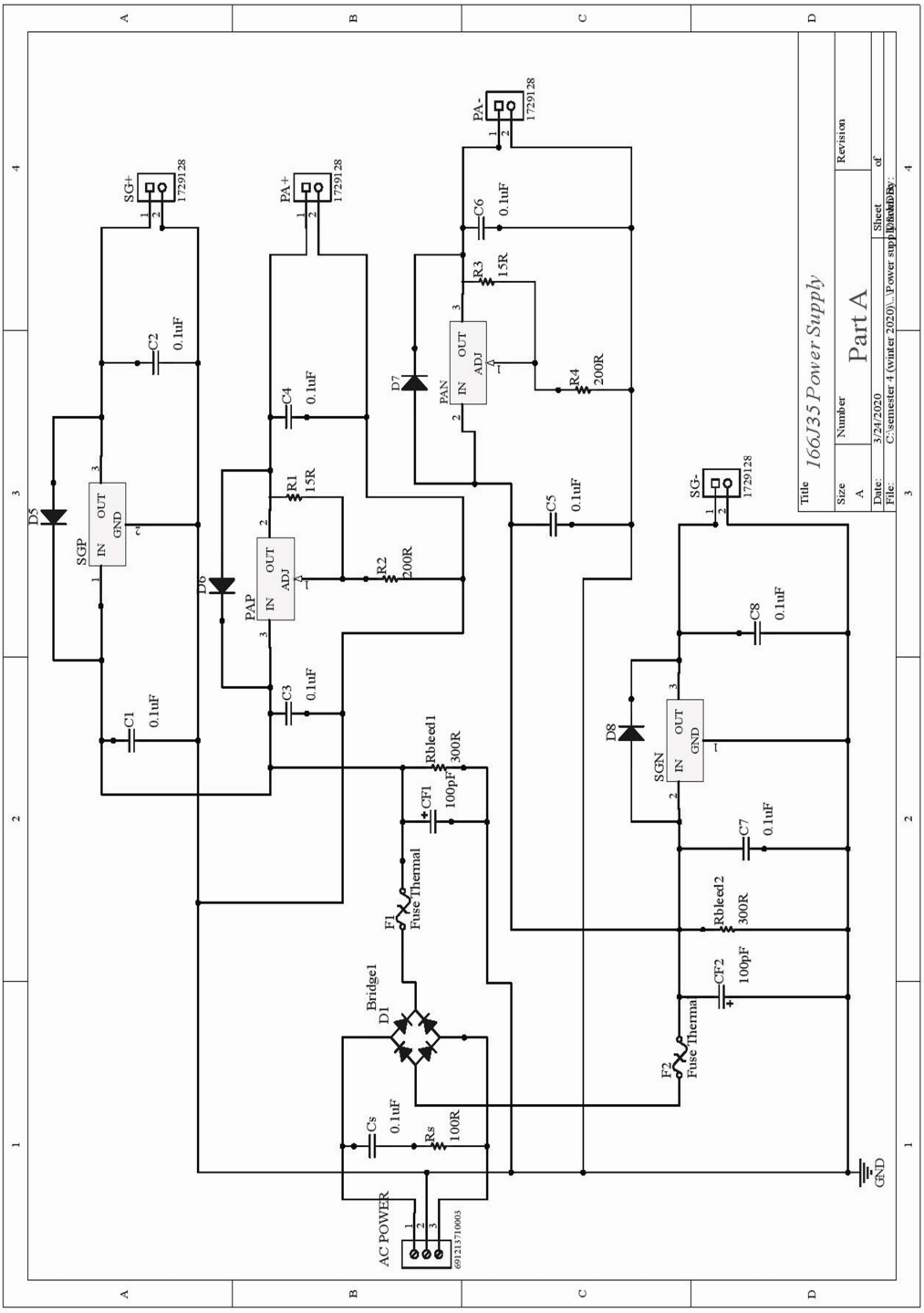


Figure 16: Final schematic design of the 166J35 Power Supply.

The final design of the 166J35 Power Supply is shown below in Figure 17.

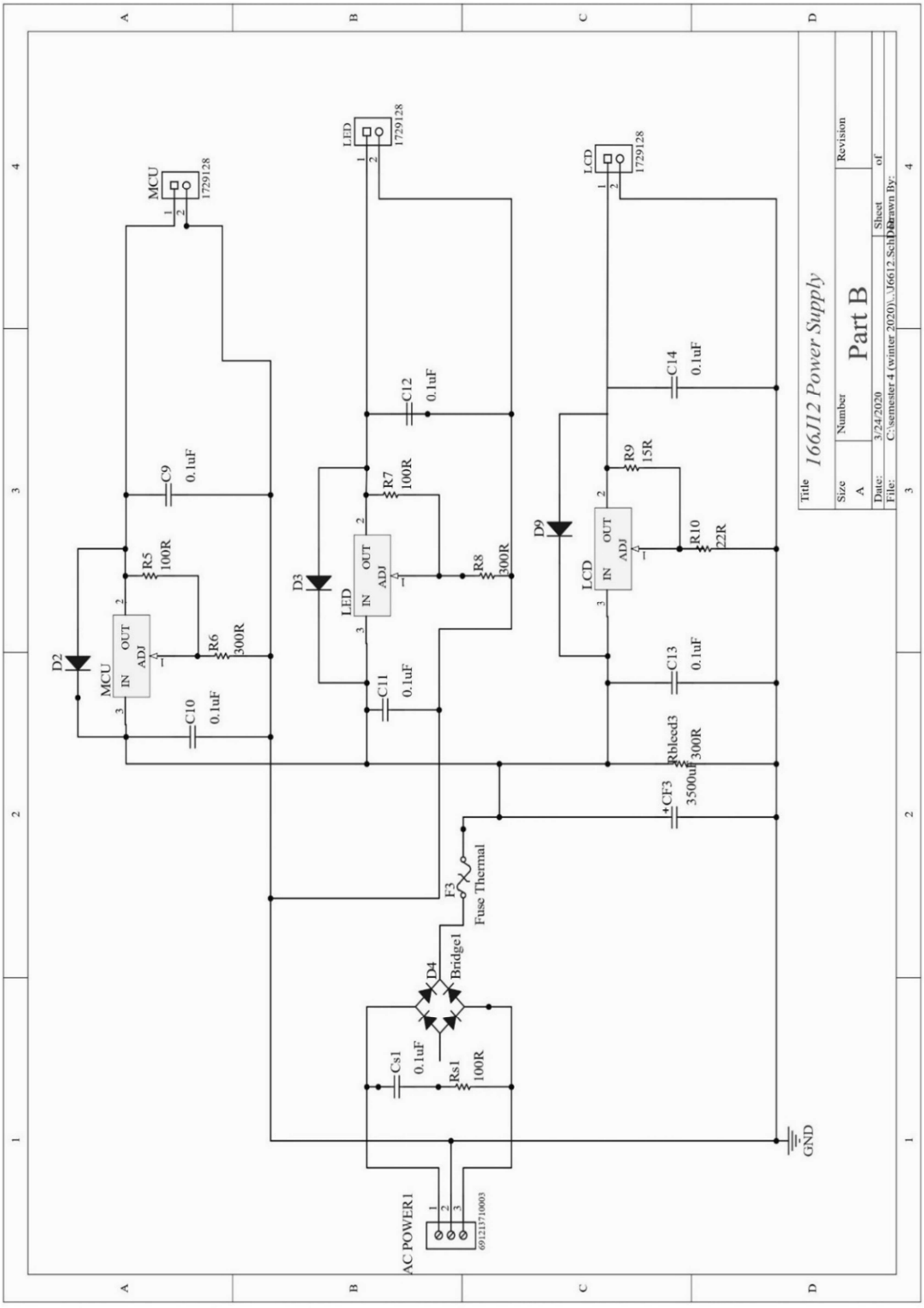


Figure 17: Final schematic design of the 166J12 Power Supply.

8.3 Power Supply Breadboard Design

This **bi-polar power supply** has two transformers with center taps, which are 166J35 and 166J12. Hammond Manufacturer 166J35 transformer has a voltage primary of 115 VRMS with a secondary voltage of 35 VRMS. [1] This transformer can handle a maximum current of 1.5 ARMS and a max power of 52.5 W. 166J35 transformer purpose is to supply the required power for the power amplifier and signal generator. [1] Meanwhile, Hammond Manufacturer 166J12 transformer has the same primary voltage but with a secondary voltage of 12 VRMS. [1] This transformer can supply current up to 1 ARMS and a max power of 12W. The purpose of this transformer is to supply the required power for laser LEDs, LCD and the MCU. This Bi-Polar power supply design exists based on the power supply lab from the ELTR. [2]

8.3.1 166J35 Transformer Calculations

The 166J35 has a primary voltage of 115VRMS with a secondary voltage of 35VRMS. The calculation for 166J35 begins by knowing the turns ratio. [1]

To calculate the turns ratio,

$$turns\ ratio = \frac{V_{pri}}{V_{SEC}} = \frac{115V_{RMS}}{35V_{RMS}} = 3.29$$

Then, the current load of each component is all measured or can be found on the data sheet. The total current can be determined by summing the current load coming from the positive and negative rails of the component.

$$I_{DC}(positive) = I_{PA} + I_{SG} = 500mA + 1.7mA = 501.7mA_{DC}$$

$$I_{DC}(negative) = 501.7mA_{DC}$$

$$I_{DC}Total = I_{DC}(Positive) + I_{Dc}(negative) = 501.7mA_{DC} + 501.7mA_{DC}$$

$$I_{DC}Total = I_{SEC} = 1.0034\ A_{RMS}$$

A fast- blow fuse is an overcurrent protection that limits the current draw from the voltage rails. This fuse is placed after the AC power to protect the transformer from supplying over the max limit current.

To calculate the fuse rating for the transformer,

$$I_{pri} = \frac{I_{SEC}}{turns\ ratio} = \frac{1.0034A_{RMS}}{3.29} = 304.98mA_{RMS}$$

$$Fuse\ rating = 1.5(304.98mA_{RMS}) = 457.47\text{m}A_{RMS}$$

Therefore, the fuse rating for the transformer will be 400mA.

This procedure is to verify the fuse rating is not exceeding the max current of the transformer which is 1.5mA.

$$I_{SEC} = Fuse\ Rating * turns\ ration = 400mA\ \times 3.29 = 1.316mA_{RMS}$$

8.3.2 PTC Resettable Fuse

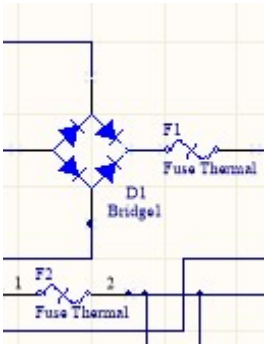


Figure 18: Full Bridge Rectifier with PTC resettable fuse

PTC resettable fuse is another kind of overcurrent protection that resets when the load reached the current limit or current spikes. [3] These fuses are placed on the positive and negative output of the bridge rectifier. The current ratings of the PTC fuses can be determined by knowing the current load on the positive and negative rail. From the previous calculation, the positive and negative rail has the same current draw of 501.7 mADC. Therefore, the PTC fuse current hold should be 600mADC with a voltage rating above 35VRMS.

8.3.3 Voltage Regulators (166J35)

Linear voltage regulators are used to supply a steady voltage on each component. The values of the voltage regulators are determined on the data sheet or design of each components. These values are included on the table below as Vout. [3]

These are the collected values from each component. All the shaded part on table 16 means are calculated, while the unshaded cells are predicted and measured.

Table 16: 166J35 Collected Values

Transformer	**166J35**				**Power from the regulator (W)**	**Power from the load (W)**
Parts Name	**Vout (V)**	**VReg (V)**	**Current Load (A)**	**Impedance (Ω)**		
Power Amplifier	18	6.05	0.5	36	3.025	1.0167
	-18	-6.05	-0.5	36	3.025	1.0167
Signal Generator	12	12.05	1.70E-03	7059	20.5E-3	20.6E-3
	-12	-12.05	-1.70E-03	7059	20.5E-3	20.6E-3
VA	24.05					
VB	-24.05					

8.3.4 Fixed Voltage Regulators (166J35)

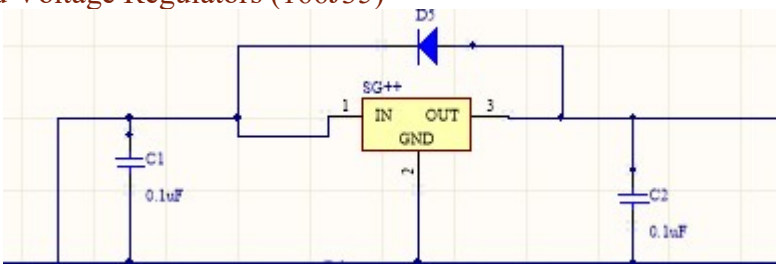


Figure 19: Voltage Regulator of the Signal Generator

Fixed voltage regulator is used to obtain a steady voltage supply to the signal generator, which is ±12V. The polarity of these voltage regulators can identify by looking at the first two digits of label. Positive voltage regulators label started with 78xx, while negative regulators started with 79xx. The last two digits of the label represent the value of the voltage regulator. It also has three pins for the input, output, and ground. [3, 4]

8.3.5 Adjustable Voltage Regulators (166J35)

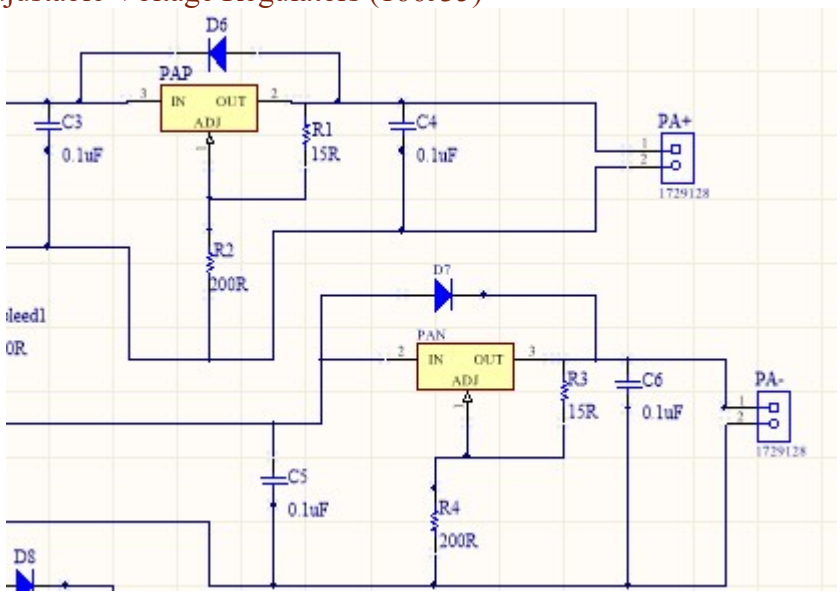


Figure 20: Positive and Negative Voltage Regulator for Power Amplifier

An adjustable voltage regulator is used for the power amplifier LED, LCD, and MCU. All of the component's values are acquired from the design of the power amplifier. Therefore, this component used an adjustable voltage regulator for easy changes. LM317 is the parts number for the positive adjustable voltage regulator and LM337 for the negative adjustable voltage regulator. These regulators can construct by changing the resistors based on the desired output voltage. This type of regulator has three pins for adjustment, input, and output.

Power Amplifier has an output voltage of ±18V and the voltage coming from the adjust pins is 1.25V.

Let R1 = R3 = 15Ω and solve for R2 = R4,

$$R_2 = R_4 = \frac{V_{out} - V_{ADJ}}{\frac{V_{ADJ}}{R_1 = R_3}} = \frac{18V - 1.25V}{\frac{1.25V}{15\Omega}} = 201\Omega \cong 200\Omega$$

8.3.6 Damping Networks (166J35)

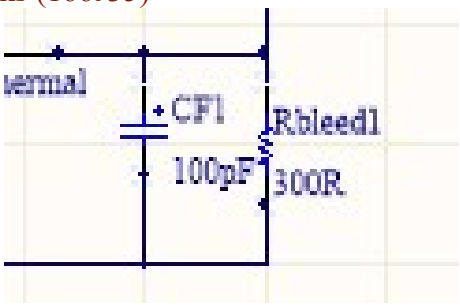


Figure 21: 166J35 RC circuit

The damping network circuit is a type of RC circuit to suppress a voltage spike at 120 Hz interface. [3] The filter capacitor in this circuit creates a $1V_{pp}$ ripple and removes the unwanted frequencies when the power is distributed. In contrast, the resistor in the circuit called RBleed absorbs the excess signal when the power comes out. [3]

Let $I_{DC(Positive)} = I_{Load\ peak}$ = 501.7mApk and CFilter = 3300µF to calculate the VRipple and R_{Bleed}.

$$V_{RIPPLE} = \frac{I_{load} \times \text{Diode offtime}}{C_{filter}} = \frac{501.7mA_P \times 85\% \times \frac{1}{120Hz}}{3300\mu F} = 1.08\ V_{PP}$$

$$R_{Bleed} = \frac{1}{\tau \times C_{Filter}} = \frac{1}{1s \times 3300\mu F} = 303\Omega \cong 300\Omega$$

Voltage Rails are the filtered voltages coming from the positive and negative output of the full-wave bridge rectifier. [3]

Secondary voltage must convert V_{RMS} to V_P to calculate the voltage rails.

$$V_{SEC(pk)} = V_{SEC(RMS)} \times \sqrt{2} = 35V_{RMS} \times \sqrt{2} = 49.5\ V_P$$

Then, V_{CT} means the voltage from the center tap.

$$V_{CT} = \frac{V_{SEC}}{2} = \frac{49.5V_P}{2} = 24.75V_P$$

V_A means the voltage coming from the positive output of the full wave bridge rectifier, while V_B is the voltage coming from the negative output of the full wave bridge rectifier.

$$V_A = V_{CT} - V_{DIODE} = +24.75V_P - 0.7V = +24.05\ V_P$$

$$V_B = V_{CT} - V_{DIODE} = -24.75V_P - (-0.7V) = -24.05\ V_P$$

8.3.7 Power Calculations (166J35)

Calculating power is essential to prevent the components from overheating and overpower. All of the calculated power on each voltage rail of the components are in table 16.

Calculation sample:

$$V_{REG} = V_A - V_{OUT_{signal\ generator}} = 24.05V_P - 12V_{DC} = 12.05V_{DC}$$

Use V_{REG} to calculate the power of the voltage regulator.

$$P_{REG} = V_{REG} \text{ ´ } I_{REG} = 12.05V \text{ ´ } 1.7mA_{DC} = 20.49mW$$

Use the I_{Load} and Vout of the signal generator to calculate the impedance. Then, calculate the power of the load.

$$R_{Signal\ Generator} = \frac{V_{out}}{I_{Load}} = \frac{12V_{DC}}{1.7mA_{DC}} = 7059\Omega$$

$$P_{LOAD} = \frac{(V_{OUT})^2}{R_{Signal\ Generator}} = \frac{(12V_{DC})^2}{7059\Omega} = 20.40\ mW$$

8.3.8 166J12 Transformer Calculations

The steps of calculating the values from 166J35 will be the same on 166J12. Therefore, all the values of each components will be in table 17.

Table 17: 166J12 Collected Values

Transformer	**166J12**				**Power from the regulator (W)**	**Power from the load (W)**
Parts Name	**Vout (V)**	**VReg (V)**	**Current Load (A)**	**Impedance (Ω)**		
MCU	5	2.79	150.0E-3	33.3333333	418.5E-3	750.0E-3
LED	5	2.79	360.0E-3	8000	1.0E+0	3.1E-3
LCD	3	4.79	135.0E-3	22.22222222	646.7E-3	405.0E-3
VA	7.79					

$$turns\ ratio = \frac{V_{pri}}{V_{SEC}} = \frac{115V_{RMS}}{12V_{RMS}} = 9.58$$

$$I_{DC}(positive) = I_{MCU} + I_{LED} + I_{LCD} = 150mA + 360mA + 135mA = 538mA_{DC}$$

$$I_{DC}Total = I_{DC}(Positive) + I_{Dc}(negative) = 538mA_{DC}$$

$$I_{DC}Total = I_{SEC} = 538mA_{DC}$$

Since, the I_{sec} is 538mA, the current hold of the PTC resettable fuse will be 550mA.

$$I_{pri} = \frac{I_{SEC}}{turns\ Ratio} = \frac{538A_{RMS}}{9.58} = 56.139mA_{RMS}$$

$Fuse\ rating = 1.5(56.139mA_{RMS})$= 84.208m$A_{RMS}$

Therefore, the fuse rating for the transformer will be 85mA.

8.3.9 Adjustable Voltage Regulators (166J12)

This calculation is for the MCU and LED, because they have the output voltage.

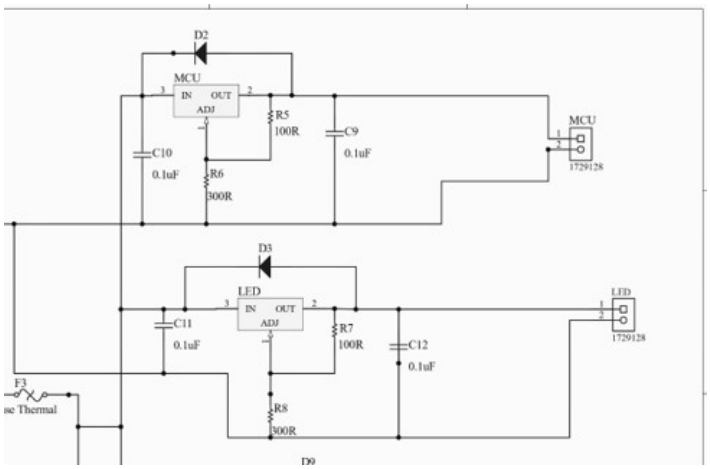


Figure 22: MCU and LED voltage rails

Let $R_5 = R_7 = 100\ \Omega$,

$$R_6 = R_8 = \frac{V_{out} - V_{ADJ}}{\frac{V_{ADJ}}{R_5 = R_7}} = \frac{5V - 1.25V}{\frac{1.25V}{100\Omega}} = 300\Omega$$

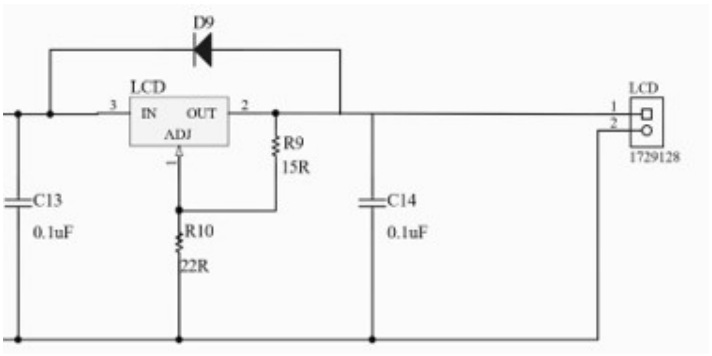


Figure 23: Voltage regulator for LCD

Let R_9 = 100 Ω,

$$R_{10} = \frac{V_{out} - V_{ADJ}}{\frac{V_{ADJ}}{R_9}} = \frac{3V - 1.25V}{\frac{1.25V}{15\Omega}} = 21\Omega \cong 22\Omega$$

8.3.10 Damping Network (166J12)

Let $I_{DC(Positive)} = I_{Load\ peak}$ = 538mApk and CFilter = 3500µF. Then, calculate the V_{RIPPLE} and R_{Bleed}.

$$V_{RIPPLE} = \frac{I_{load} \times \text{Diode offtime}}{C_{filter}} = \frac{538mA_P \times 85\% \times \frac{1}{120Hz}}{3500\mu F} = 1.09\ V_{PP}$$

$$R_{Bleed} = \frac{1}{\tau \times C_{Filter}} = \frac{1}{1s \times 3500\mu F} = 286\Omega \cong 300\Omega$$

The breadboard layout of 166J12 Power supply is in Figure 24.

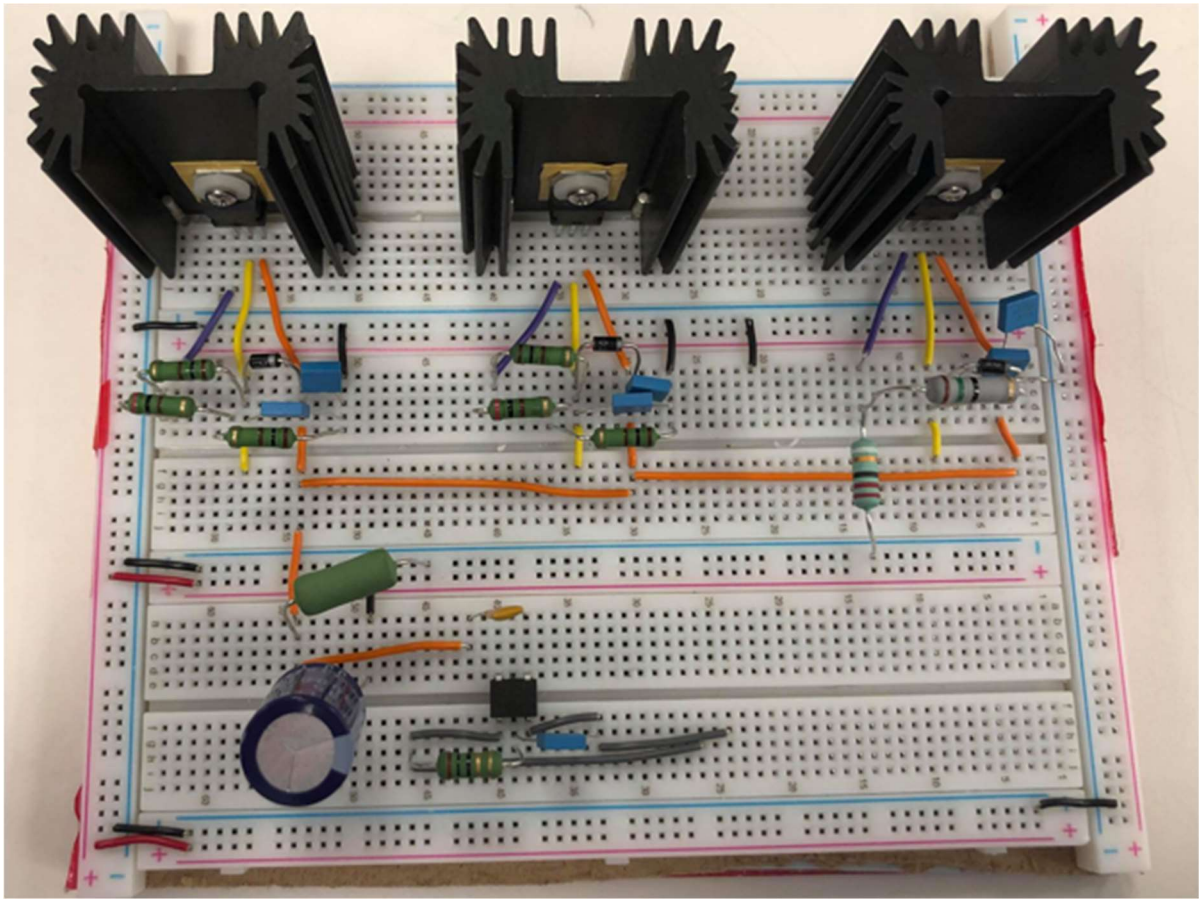


Figure 24: Breadboard layout of the 166J12 Power Supply.

8.4 Power Amplifier Schematic

The final design of the Power Amplifier is shown below in Figure 18.

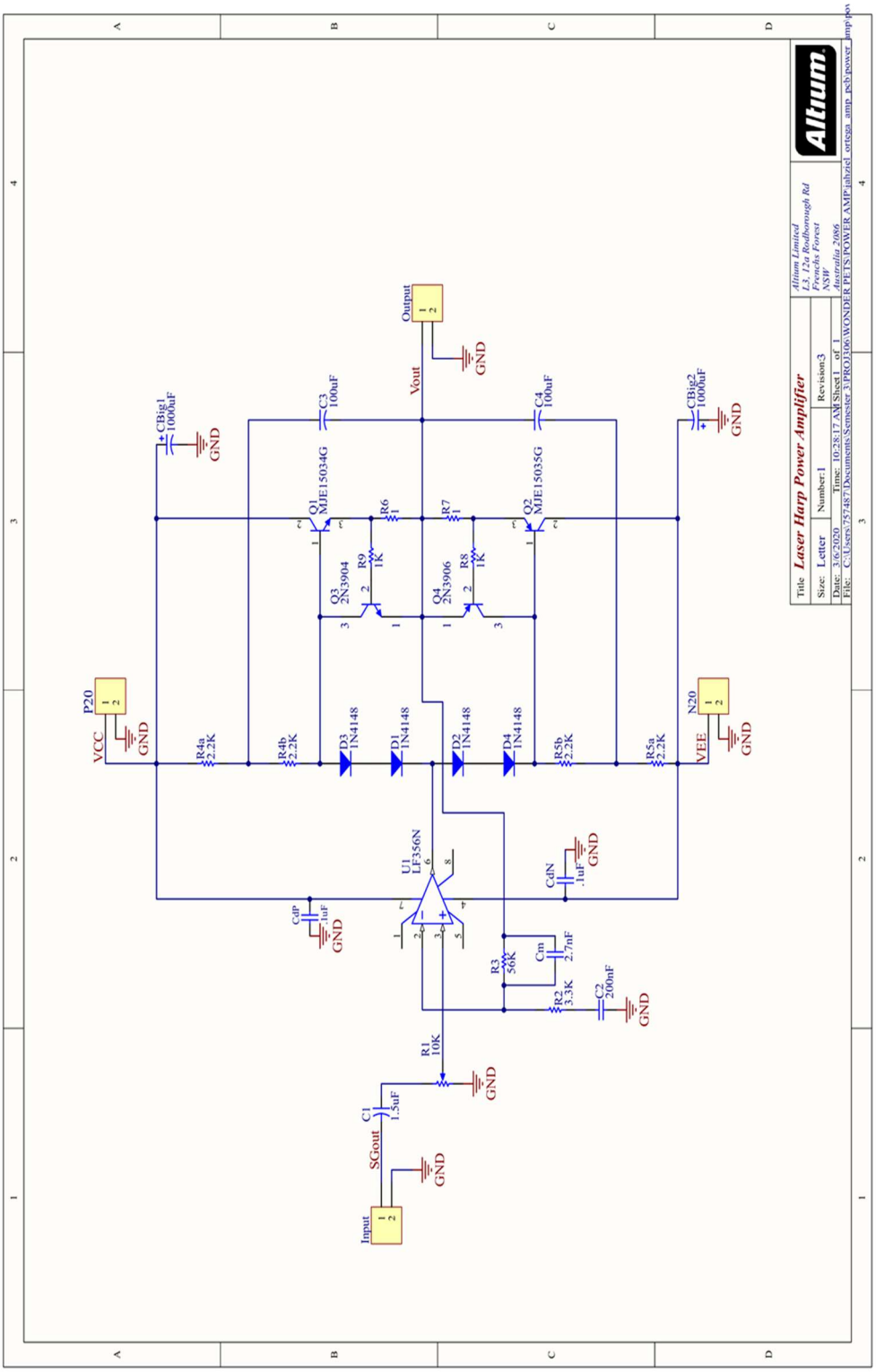


Figure 25: Final schematic design of the power amplifier.

8.5 Power Amplifier Breadboard Design

This is a BJT-boosted op-amp power amplifier, it is designed to have a gain of 3dB so that the signal generator would be able to produce a signal with less noise. Using an NE5534, we will have low noise on the signal. Since the operational amplifiers are compensated internally for a gain grater of equal to three, it is appropriate for the power amplifier. This circuit is based on a lab done previously in ELTR [5].

To design the power amplifier, we must pick the value of the load resistor and power:

$$P_L = 1W; R_L = 8\Omega;$$

$$P = \frac{V_{RMS}^2}{R}; V_{RMS} = \sqrt{PR} = \sqrt{(1W)(8\Omega)} = 2.83V_{RMS} = 4V_p$$
$$I_p = \frac{4V_p}{8\Omega} = 500mA$$
$$V_E = I_L \times R_G + V_{out_p} = (500mA)(1\Omega) + 4V_p = 4.5\text{Vp}$$
$$V_B = V_E + 0.7V = 4.5V + 0.7V = 5.2V_p$$
$$V_x = 4.5V_p$$

Choosing the power transistors, we needed a high base current rating. The MJE15034 and MJE15035 transistor are rated to 1ADC, these transistors will be a good choice as it would not overheat, and burn up.

Using the β from the MJE15034 and MJE15034 datasheets, we can approximate that $\beta = 180$ and having VCC and VEE as ±18, we can use it to find the current of I_B.

$$I_B = \frac{1}{\beta} \times I_p = \frac{1}{200} \times (500mA) = 2.5mA_{max}$$
$$R_4 = \frac{(V_{CC} - V_B)}{I_B} = \frac{(18V - 5.2V)}{2.78mA} = 4604\Omega$$
$$R_{4a,4b} = \frac{R_4}{2} = \frac{4604\Omega}{2} = 2302\Omega \approx 2200\Omega$$

Since $R_{4a,4b}$'s resistor values aren't common, we can use $2.2k\Omega$ resistors.

The circuit for R_4 and R_5 are similar, the only difference being the voltage rails, one uses the positive rail, while the other uses the negative rail. Therefore, we can assume that $R_4 = R_5$.

$$R_{5a,5b} = 2.2k\Omega$$

8.5.1 Voltage Follower

From the calculations above we can start calculating the resistors needed for a mid-band gain of 3dB.

$$A_{V_{mid}} = 3dB = 10^{\frac{3}{20}} = 1.4125\frac{V}{V}$$

With the $A_{V,mid}$ calculated, we can choose a value for R_2, and calculate for R_3:

Letting $R_2 = 3.9k\Omega$,

$$A_{V_{mid}} = 1 + \frac{R_3}{R_2}$$
$$1.4125 = 1 + \frac{R_3}{3.9k\Omega}$$
$$R_3 = (1.4125 - 1) * 3.9k\Omega = 1610\Omega$$

Use $R_3 = 1.5k\Omega$.

8.5.2 Bandwidth

The bandwidth needs to be at least between 200Hz and 1800Hz because the Laser Harp will have to notes C4 to B5 which is $261.626Hz$ to $987.767Hz$.

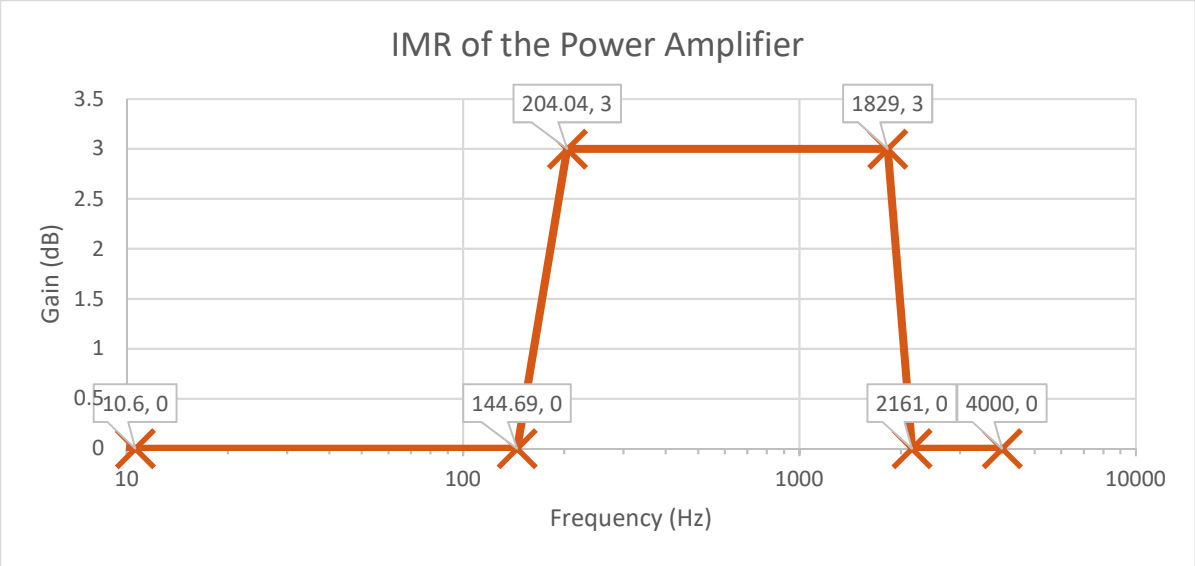


Figure 26: IMR graph of the Power Amplifier.

Using $C_2 = 200nF$, we have:

$$f_2 = \frac{1}{2\pi(R_2 + R_3)C_2} = \frac{1}{(2\pi(3.9k\Omega + 1.5k\Omega)200nF)} = 147.36Hz$$

We have to make sure that the mid band region should at least between 200Hz and 1800Hz.

To find the roll-up frequency (f_3) we have:

$$f_3 = \frac{1}{2\pi(R_2)(C_2)} = \frac{1}{2\pi(3.9k\Omega)(200nF)} = 204.04Hz$$

The roll-off frequency (f_4) should be about 2000Hz. To find C_m, we have

$$C_m = \frac{1}{2\pi(R_3)(f_3)} = \frac{1}{2\pi(1.6k\Omega)(1800Hz)} = 53.0nF$$

Using a $58nF$ capacitor will give me a frequency of $1829Hz$ for f_4's roll-off which is enough for the mid-band region.

To find f_5, we have:

$$f_5 = \frac{1}{2\pi(R_2//R_3)C_m} = \frac{1}{2\pi(3.9k//1.5k)68nF} = \frac{1}{2\pi(1.083k\Omega)68nF} = 2161Hz$$

With the calculations above, I was able to create an IMR graph, as seen in Figure 26.

8.5.3 Overcurrent Protection

The overcurrent protection consists of two transistors (Q_3 and Q_4) and two resistors (R_8 and R_9). This circuit protects the power transistors in case there is surge in current. In the case that there is excessive current going through the power transistors, the voltage across the R_6 and R_7 will be enough to turn on the protection transistors. These will then pull the extra current from the power transistor's base current to R_8 or R_9.

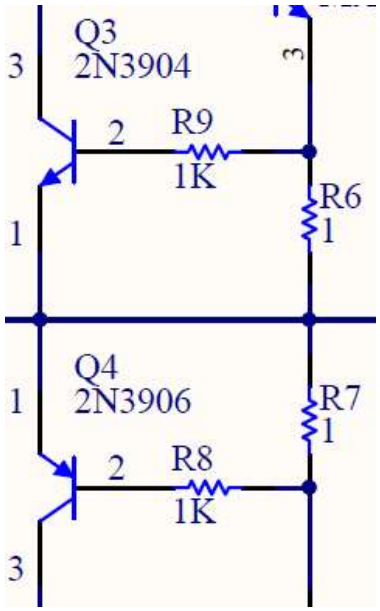


Figure 27: Power Amplifier overcurrent protection.

8.5.4 Diode Biasing

Diode biasing push-pull transistors allows the circuit's voltage to be stable even when the power supply is fluctuating its voltage.

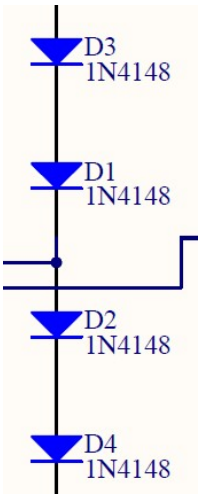


Figure 28: Power Amplifier's Diode Biasing circuit.

Figure 29 shows the breadboard layout used for the Power Amplifier.

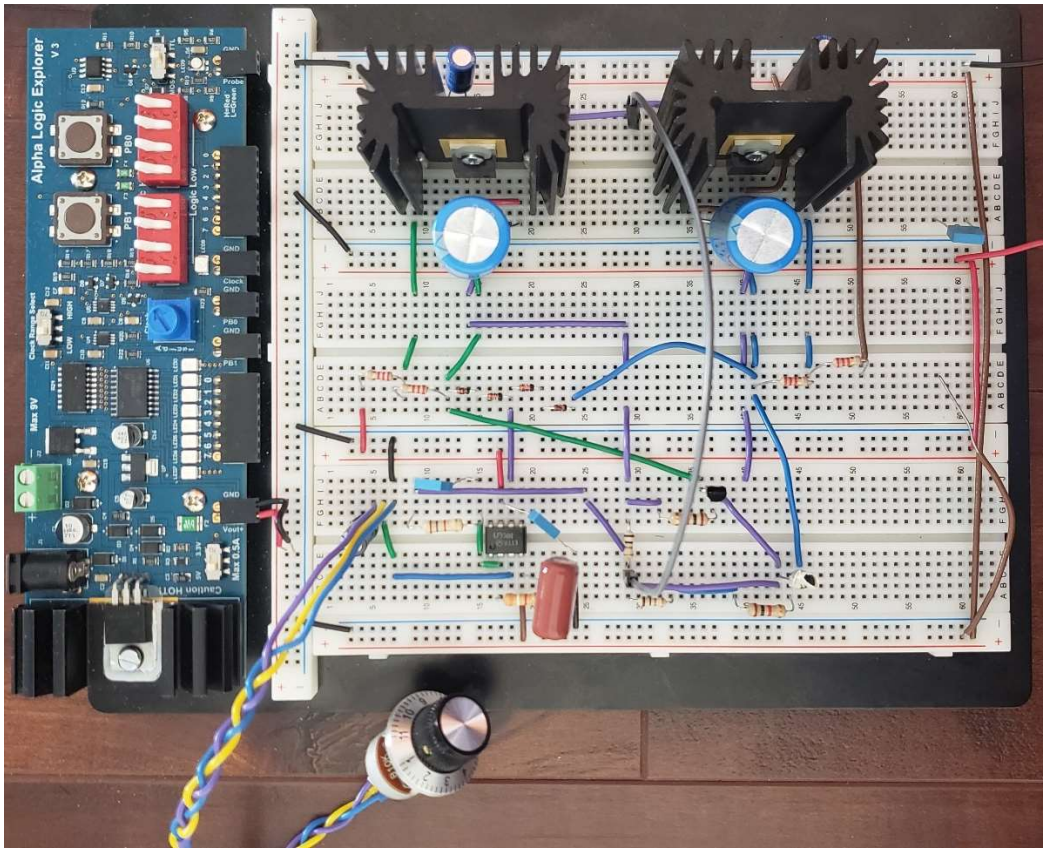


Figure 29: Breadboard layout of the Power Amplifier.

8.6 Signal Generator Schematic

The final design for the Signal Generator is shown below in Figure 30.

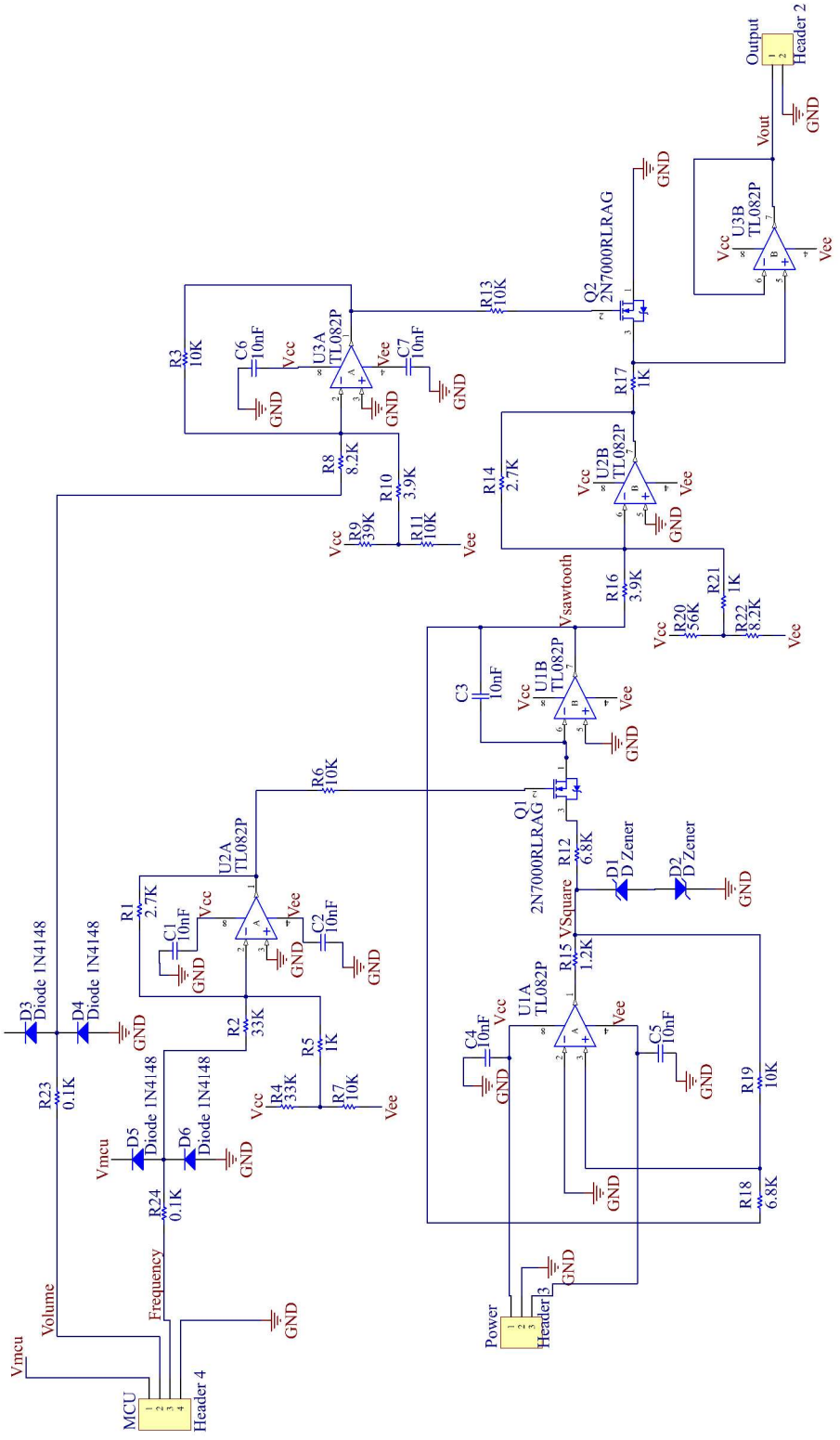


Figure 30: Final schematic design of the Signal Generator.

8.7 Signal Generator Breadboard Design

The simplified schematic for the main sawtooth oscillator is as shown in Figure 31.

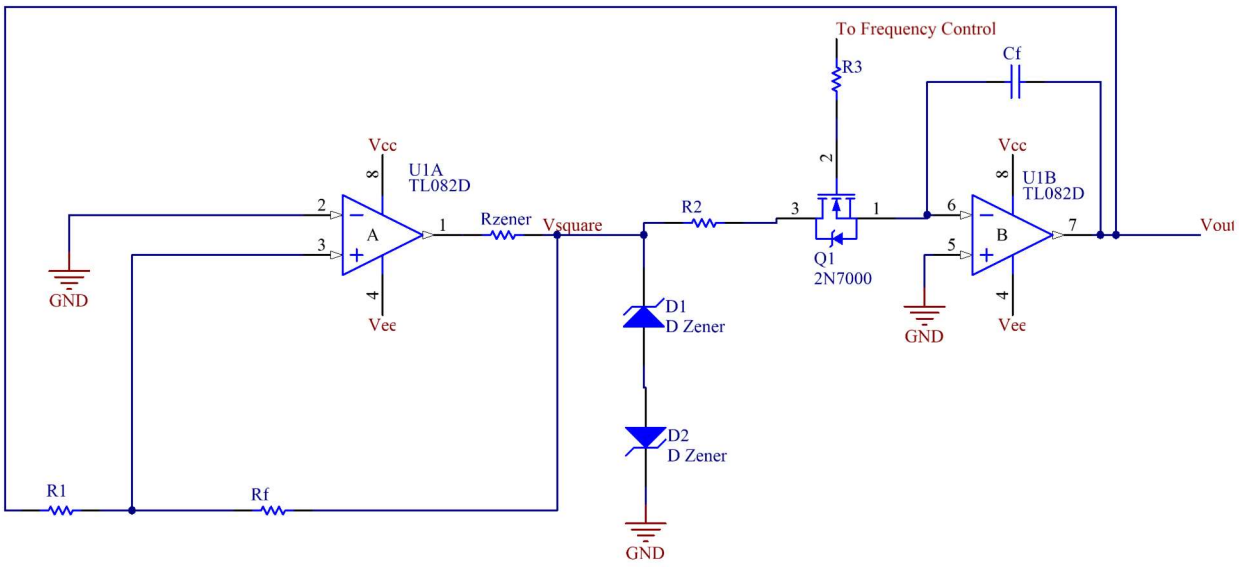


Figure 31: Simplified schematic for the main sawtooth oscillator.

The circuit is comprised of a non-inverting Schmitt trigger connected to an integrator. Since the integrator's capacitor is charged by the Schmitt trigger, V_{out} will continuously rise and fall between the threshold levels. The MOSFET alters the current flow in one direction, based on the gate-source voltage. This allows a **sawtooth wave**form to be generated with variable frequency.

8.7.1 Zener Resistor Choice

The part number for the Zener diodes is 1N5231. That Zener type has a Zener voltage of 5.1V and a forward voltage of 1.1V.

The Zener current to be used will be 5mA.

Assuming a +/- 12V saturation voltage from the operational amplifier:

$$R_{zener} = \frac{+V_{sat} - V_{zener} - V_{forward}}{I_{zener}} = \frac{12V - 5.1V - 1.1V}{5mA} = 1.160k\Omega$$

The standard resistor value 1.2kΩ will be used.

The Zener diodes will constrain V_{square} to the range of -6.2V to +6.2V.

8.7.2 Schmitt Trigger Design

Let R_f = 10kΩ.

For trigger levels of +/- 4V:

$$R_1 = \frac{V_{UTP}}{+V_{square}} R_f = \frac{+4V}{+6.2V}(10k\Omega) = 6.45161k\Omega$$

The standard resistor value of 6.8kΩ will be used.

Recalculating the trigger levels:

$$V_{UTP} = +V_{square}\frac{R_1}{R_f} = (+6.2V)\frac{6.8k\Omega}{10k\Omega} = +4.216V$$

$$V_{LTP} = -V_{square}\frac{R_1}{R_f} = (-6.2V)\frac{6.8k\Omega}{10k\Omega} = -4.216V$$

8.7.3 Integrator Design

The transfer function for an integrator circuit is as follows:

$$V_{out} = -\frac{1}{RC}\int V_{in}(t)dt$$

A **sawtooth wave**'s period contains a fast change in voltage and a slow change in voltage.

Since the TL082 has a max rated output current of 20mA, we need a small current value to create the fast change in voltage.

To minimize the current requirement, we will choose the low value of 10nF for Cf.

To make a fast change equal to 1/10 of our max frequency, the time from -4.216V to +4.216V must be:

$$T_{fast\ chang} = \frac{1}{10f_{max}} = \frac{1}{10(1kHz)} = 100.0\mu s$$

So, the V/s change must be:

$$m = \frac{+4.216V - (-4.216V)}{100.0\mu s} = 84.32kV/s$$

$$m = \frac{1}{RC_f}V_{in}$$

$$R_2 = \frac{1}{mC_f}V_{in} = \frac{1}{(84.32kV/s)(10nF)}(6.2V) = 7.35294k\Omega$$

The standard resistor value of 6.8kΩ will be used. This will reduce the rise time, so the error will not be a problem.

The changing gate-source voltage of the MOSFET manipulates the current flow to the capacitor, varying its charge rate and altering the frequency of the output waveform.

Calculating the max power dissipation of the MOSFET:

$$P_{MOSFET\ 1\ (max)} = \frac{\left(+V_{square}\right)^2}{4R_2} = \frac{(6.2V)^2}{4(6.8k\Omega)} = 1.413mW$$

The maximum power rating of the 2N7000 MOSFET used in this design is 350mW [6], so the transistor is operating within its specified power range.

R3 will be set to 10kΩ. The purpose of this resistor is to prevent unintended high frequency oscillations due to parasitic inductances and capacitances within the MOSFET. Since the gate of the MOSFET has a very high impedance, the resistor will have no effect on the gate-source voltage.

8.7.4 MCU Input Protection

The frequency and volume of the Signal Generator's output waveform will be controlled by an analog voltage supplied by the MCU's DAC outputs. In the event of circuit failure, the MCU may be exposed to voltages outside its rated range of 0-3.3V, which could cause damage. To protect the MCU, a protection system consisting of two 1N4148 diodes and a 100Ω resistor will be placed at each DAC output. The input protection will be connected as shown in Figure 32.

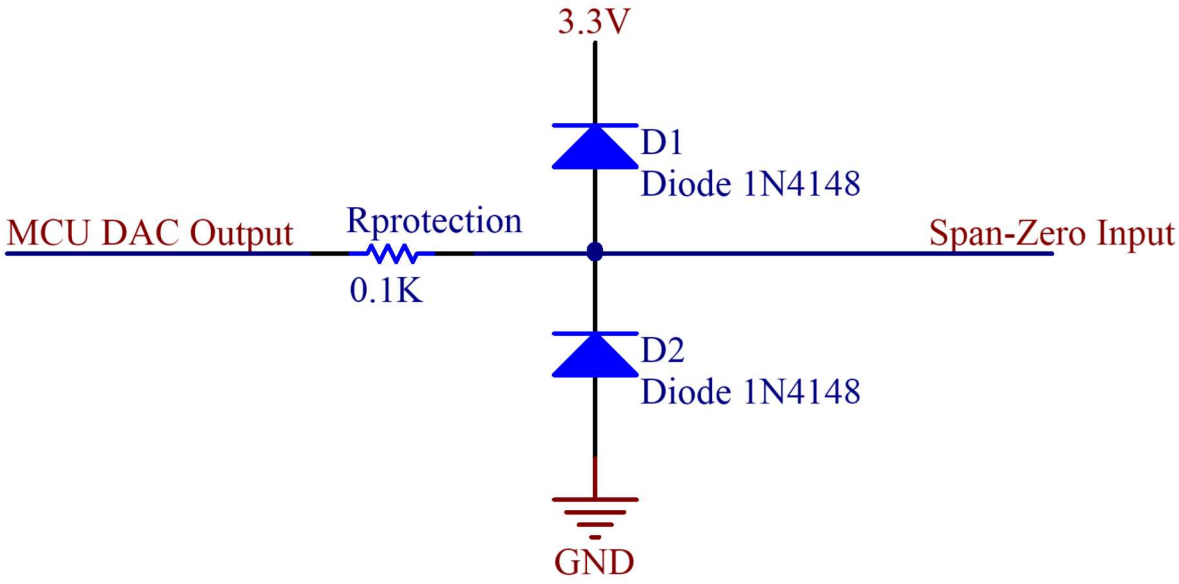


Figure 32: MCU Input Protection System

The diodes will short voltages outside of the range of 0-3.3V to ground through the MCU's power supply.

8.7.5 Span-Zero 1 Circuit

The MCU has two DACs capable of outputting 0-3.3V. By running tests on the oscillator, we found that a 1.75V – 2.0V analog signal produces the required frequency range of 200Hz to 1kHz with a reasonable margin of error. To map the voltage correctly, a span-zero operational amplifier circuit will be used. A simple span-zero circuit diagram is as shown in Figure 33.

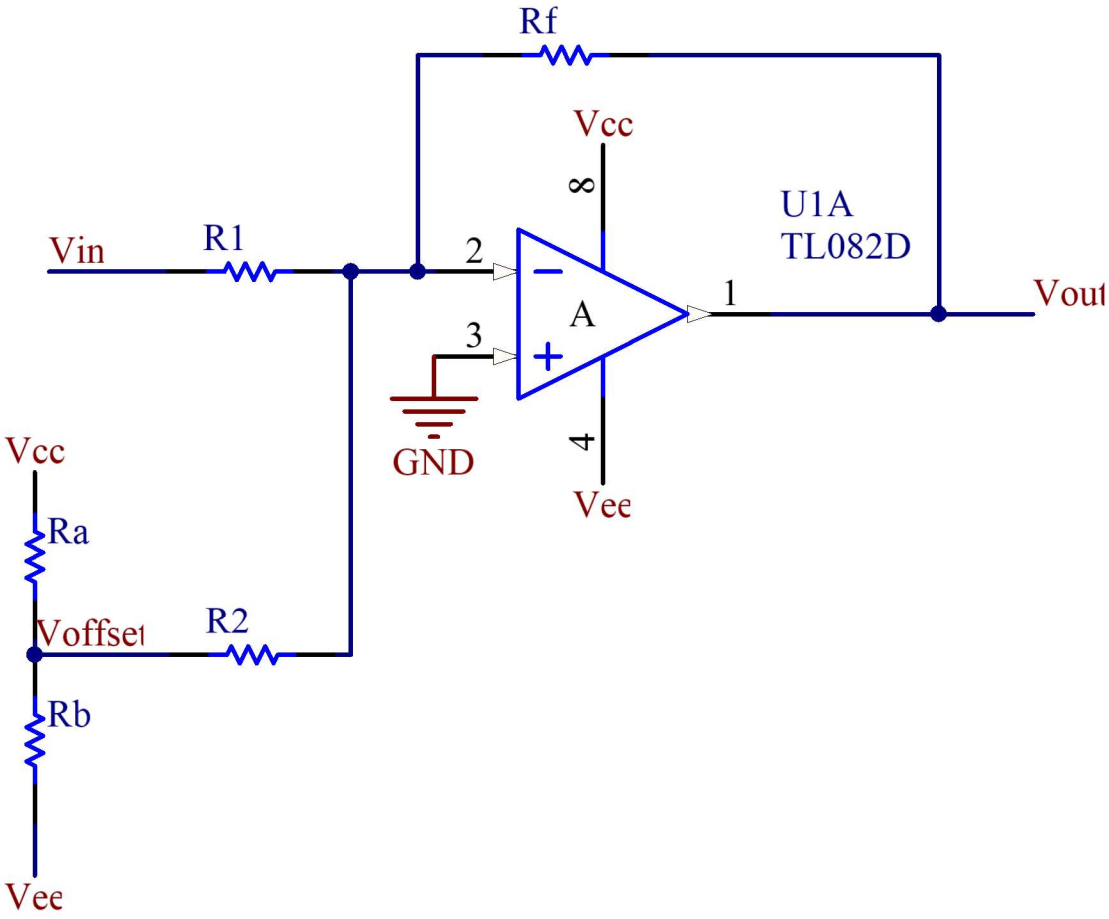


Figure 33: Span-Zero Circuit Design

Where the transfer function is:

$$V_{out} = mV_{in} + b$$

We will create an excel spreadsheet to calculate the resistor values.

Equations used:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_2 - mx_2$$

$$I_b = \frac{V_{offset} - V_{ee}}{R_b}$$

$$I_2 = \frac{V_{offset}}{R_2}$$

$$I_a = I_b + I_2$$

$$V_{offset} = \frac{\left(\frac{V_{cc}}{R_a} + \frac{V_{ee}}{R_b}\right)}{\left(\frac{1}{R_a} + \frac{1}{R_b} + \frac{1}{R_2}\right)}$$

$$R_a = \frac{V_{cc} - V_{offset}}{I_a}$$

$$R_1 = -\frac{R_f}{m}$$

Let R_b = 10kΩ, R_2 = 1kΩ, and R_f = 2.7kΩ.

Table 18: Span-Zero 1 Calculations

Span-Zero 1 Calculations:

Input Values:

Vcc	12
Vee	-12
Rb	10000
R2	1000
Rf	2700

Mapping Values:

y1	2
y2	1.75
x1	0
x2	3.3

Offset Calculations:

Voffset	-0.74074
Ib	0.001126
I2	-0.00074
Ia	0.000385

Slope and Intercept Calculated:

m	-0.07576
b	2

Calculated Resistor Values:

Ra	33076.92
R1	35640

Selected Resistor Values:

Ra	33000
R1	33100

Offset Recalculations:

Voffset	-0.73995
Ib	0.001126
I2	-0.00074
Ia	0.000386

Slope and Intercept Recalculated:

m	-0.08157
b	1.997855

Recalculated Output Voltage Range:

y1	1.997855
y2	1.728671

For Ra, the standard resistor value 33kΩ will be used. The error should increase the intercept value, so the minimum frequency of 200Hz should still be attainable.

For R1, the standard resistor value of 33kΩ will be used in series with a 100Ω resistor included as part of the input protection for the MCU. The selected values will increase the gain, so a minimum frequency range of 200Hz – 1000kHz should still be possible.

The maximum current draw from the MCU DAC1 to this circuit will be:

$$I_{DAC1\ (max)} = \frac{V_{in\ (max)}}{R_1} = \frac{3.3V}{33k\Omega + 100\Omega} = 99.70\mu A$$

8.7.6 Span-Zero 2 Circuit

To achieve volume control from the MCU using a MOSFET, the output triangle waveform must be offset to be entirely above 0V, in order to avoid reverse current flow. The waveform also must be attenuated to match the power amplifier's required input voltage. The output waveform will have a peak-to-peak voltage of 1Vpp. The same spreadsheet will be used to design a span-zero circuit capable of outputting the specified signal.

Let R_b = 8.2kΩ, R_2 = 1kΩ, and R_f = 2.7kΩ.

Table 19: Span-Zero 2 Calculations

Span-Zero 2 Calculations:

Input Values:

Vcc	12
Vee	-12
Rb	8200
R2	1000
Rf	2700

Mapping Values:

y1	5.66
y2	0
x1	-4
x2	4

Offset Calculations:

Voffset	-1.0481481
Ib	0.00133559
I2	-0.0010481
Ia	0.00028744

Slope and Intercept Calculated:

m	-0.7075
b	2.83

Calculated Resistor Values:

Ra	45393.7775
R1	3816.25442

Selected Resistor Values:

Ra	56000
R1	3900

Offset Recalculations:

Voffset	-1.0959113
Ib	0.00132977
I2	-0.0010959
Ia	0.00023386

Slope and Intercept Recalculated:

m	-0.69231
b	2.958961

Recalculated Output Voltage Range:

y1	5.72819141
y2	0.18972987

For R1, the standard resistor value of 3.9kΩ will be used. The reduced gain should still provide a waveform with acceptable amplitude for the power amplifier input.

For Ra, the standard resistor value 56kΩ will be used. The error should increase the intercept, which will create a greater margin of error between the output waveform and 0V.

Since the Power Amplifier is AC coupled, the DC offset will be ignored when the signal is amplified.

8.7.7 MCU Volume Adjustment

For the MCU volume adjustment, a **voltage divider** will be created using a MOSFET and a resistor. For the resistor in this network, 1kΩ should be an acceptable value to produce enough variation. By testing the MOSFET, we found that a gate-source voltage of 6V produces a 0V drain-source voltage, resulting in zero volume output. A gate-source voltage less than or equal to 2V cuts off current flow through the MOSFET, producing the maximum possible drain-source voltage and outputting the maximum volume. Another span-zero circuit will be needed to produce the required gate-source voltage range from the MCU's second DAC output. The same spreadsheet will be used to calculate resistor values.

Let Rb = 10kΩ, R2 = 3.9kΩ, and Rf = 10kΩ.

Table 20 Span-Zero 3 Calculations

Span-Zero 3 Calculations:

Input Values:

Vcc	12
Vee	-12
Rb	10000
R2	3900
Rf	10000

Mapping Values:

y1	6
y2	2
x1	0
x2	3.3

Offset Calculations:

Voffset	-2.34
Ib	0.000966
I2	-0.0006
Ia	0.000366

Slope and Intercept Calculated:

m	-1.21212
b	6

Calculated Resistor Values:

Ra	39180.33
R1	8250

Selected Resistor Values:

Ra	39000
R1	8300

Offset Recalculations:

Voffset	-2.33557
Ib	0.000966
I2	-0.0006
Ia	0.000368

Slope and Intercept Recalculated:

m	-1.20482
b	5.988642

Recalculated Output Voltage Range:

y1	5.988642
y2	2.012739

For Ra, the standard resistor value 39kΩ will be used. The error should decrease the intercept value, though the volume range should still be acceptable.

For R1, the standard resistor value of 8.2kΩ will be used. As part of the input protection for the MCU, a 100Ω resistor will be placed in series with R1. The error will not have a significant effect on the volume range.

The max current draw from the MCU DAC2 will be:

$$I_{DAC2\ (max)} = \frac{V_{in\ (max)}}{R_1} = \frac{3.3V}{8.2k\Omega + 100\Omega} = 397.6\mu A$$

The max power dissipation of the MOSFET must be determined.

Calculating the max voltage across the MOSFET (Using values from the Span-Zero 2 circuit):

$$V_{MOSFET\ 2\ (max)} = -\frac{R_f}{R_1}\left(V_{in\ (min)}\right) + b = -\frac{10k\Omega}{8.2k\Omega}(0V) + 6V = 6.000V$$

Calculating the max power dissipation of the MOSFET:

$$P_{MOSFET\ 2\ (max)} = \frac{\left(+V_{MOSFET\ 2\ (max)}\right)^2}{4R_{divider}} = \frac{(6.000V)^2}{4(1k\Omega)} = 9.000mW$$

This value is still well below 350mW, the max power rating for the 2N7000 [6].

As with the previous MOSFET, a 10kΩ resistor will be placed in series between the Span-Zero output and the gate to prevent unintended high frequency oscillations.

A voltage follower will be connected to the point between the MOSFET and resistor. This allows the voltage reading to be transmitted to the Power Amplifier without producing loading effects on either circuit.

8.7.8 Max MCU Current Draw

The max current drawn from the MCU by this circuit will be:

$$I_{MCU\ (max)} = I_{DAC1\ (max)} + I_{DAC2\ (max)} = 99.70\mu A + 397.6\mu A = 497.3\mu A$$

8.8 Printed Circuit Boards Design

8.8.1 Power Supply PCB Design

Figure 34 and Figure 35 shows the top and bottom layer of the final PCB design of the Power Supply.

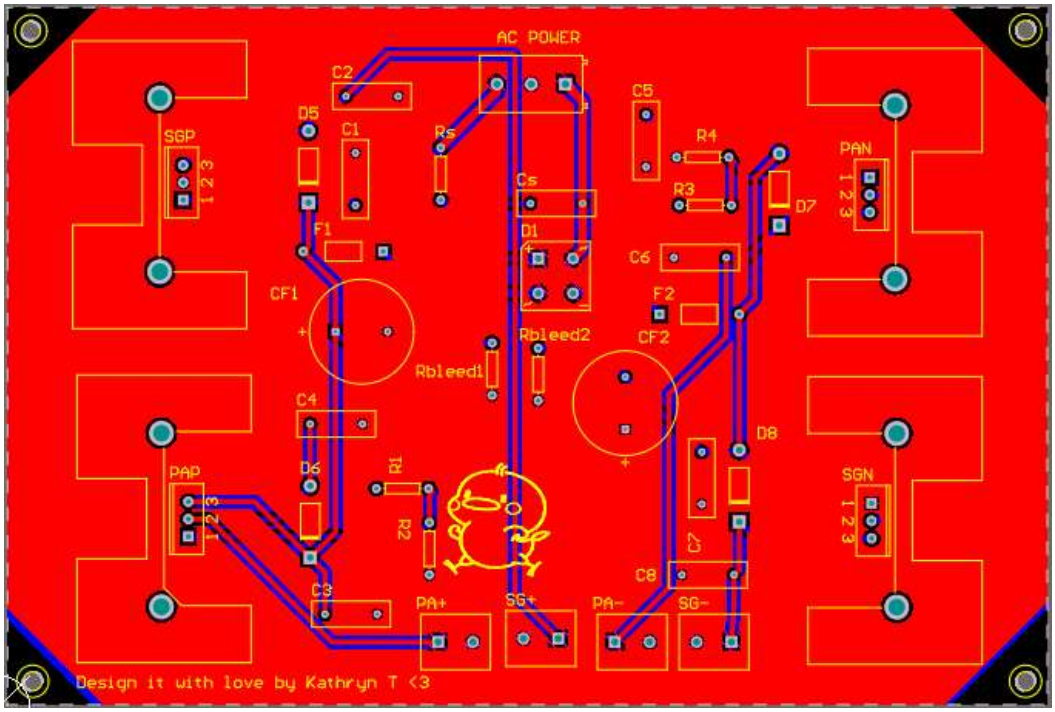


Figure 34: Printed Circuit Design (Top Layer) of the 166J35 Power Supply

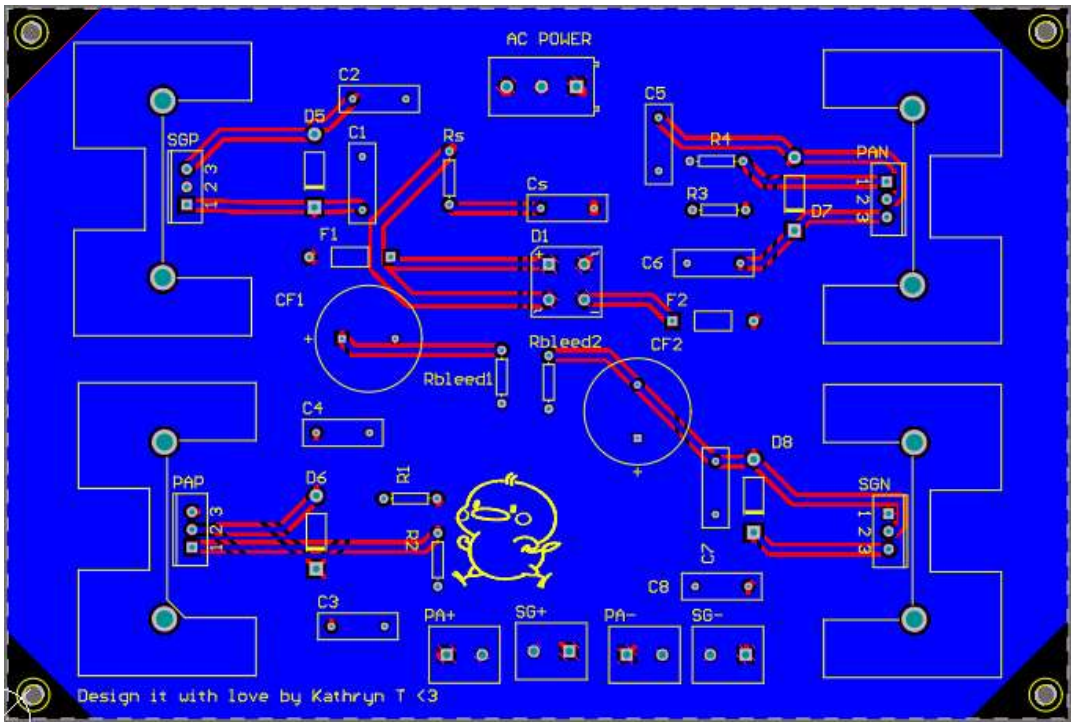


Figure 35: Printed Circuit Design (Bottom Layer) of the 166J35 Power Supply

8.8.2 Power Amplifier PCB Design

Figure 36 and Figure 37 shows the top and bottom layer of the final PCB design of the Power Amplifier.

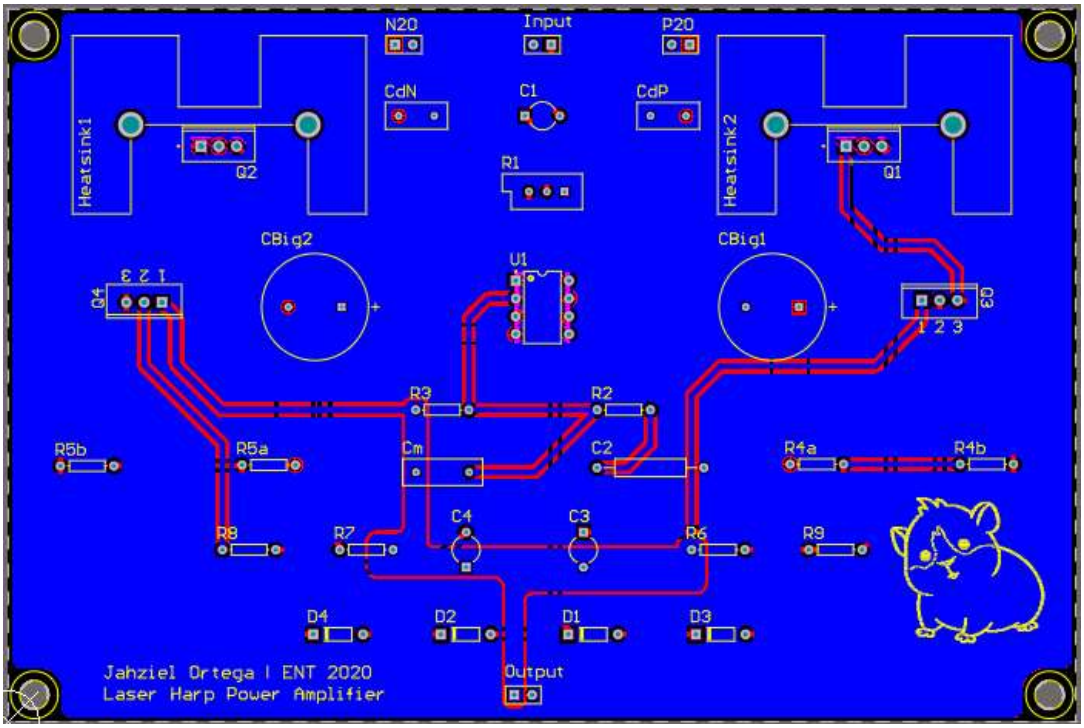


Figure 36: Printed Circuit Design (Bottom Layer) of the Power Amplifier

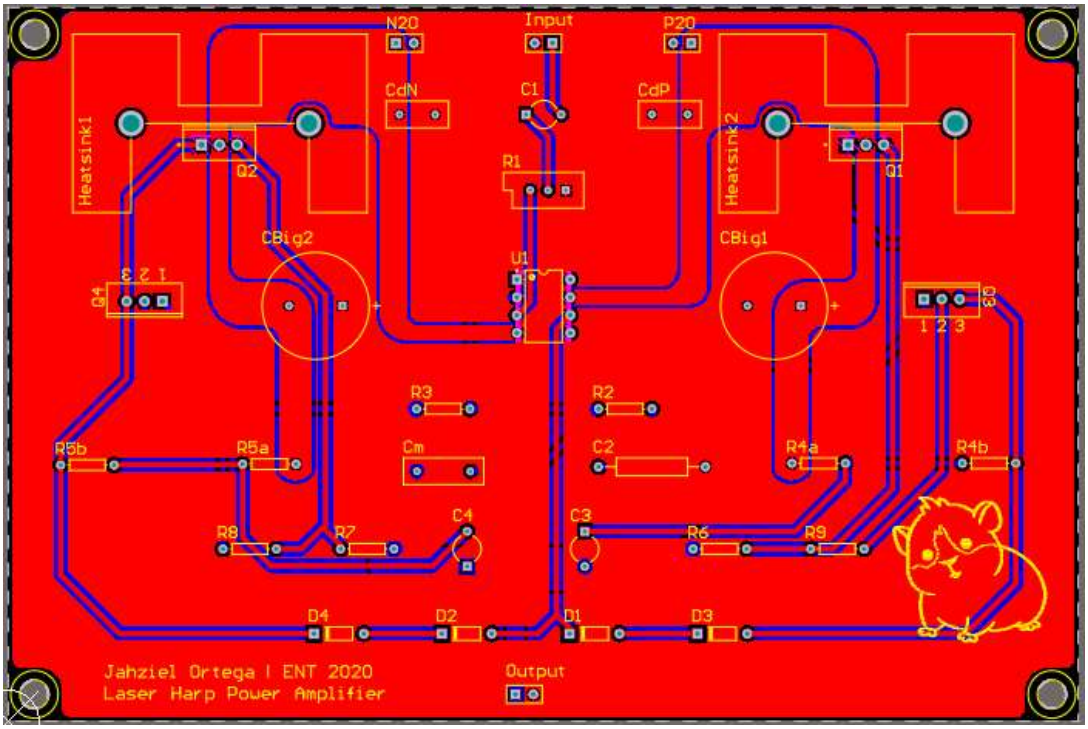


Figure 37: Printed Circuit Design (Top Layer) of the Power Amplifier

8.8.3 Signal Generator PCB Design

Figure 38 and Figure 39 shows the top and bottom layer of the final PCB design of the Signal Generator.

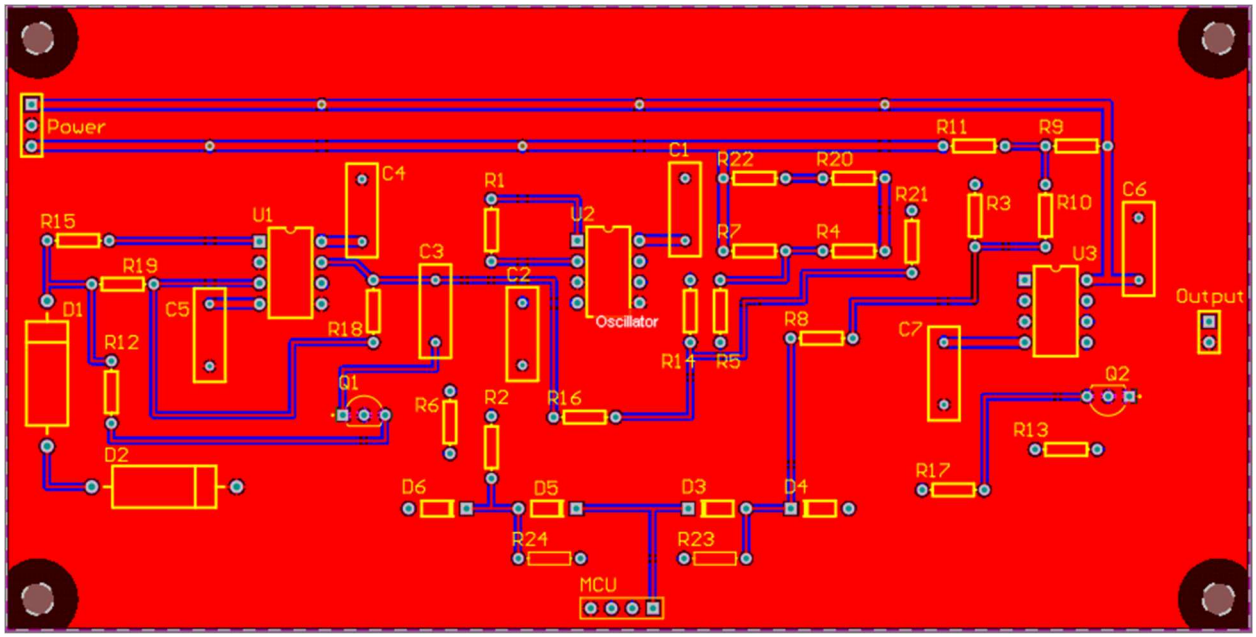


Figure 38: Printed Circuit Design (Top Layer) of the Signal Generator.

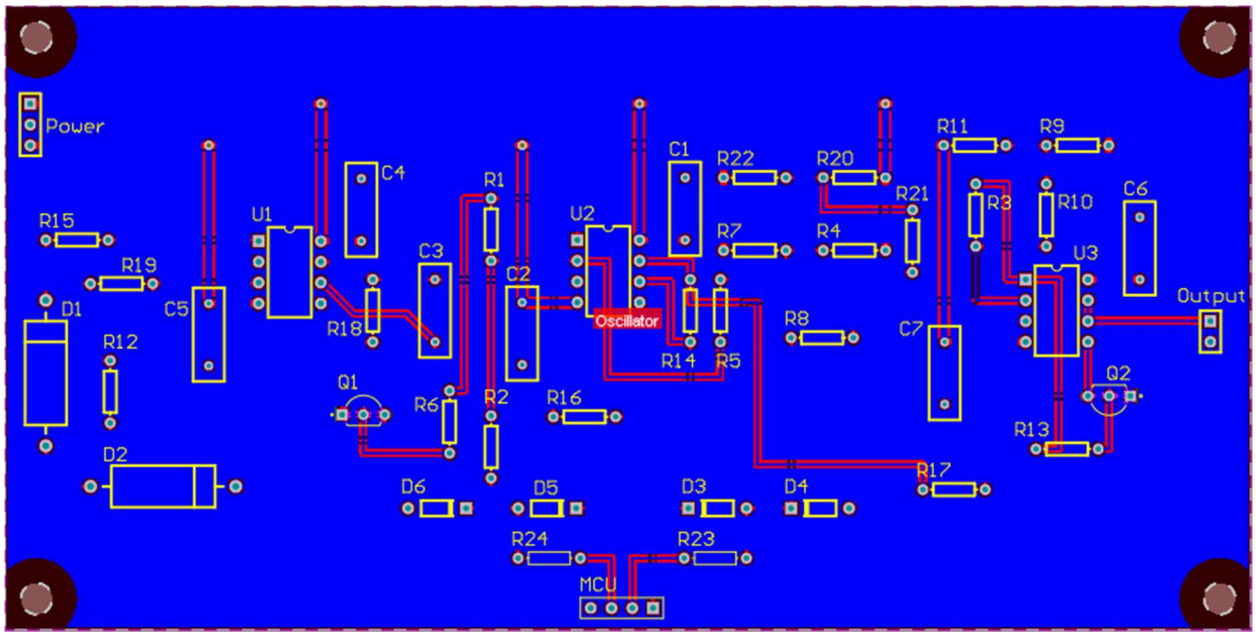


Figure 39: Printed Circuit Design (Bottom Layer) of the Signal Generator.

9.0 SOFTWARE DESIGN

This section details the processes undergone to program the software that allows the project to function. The functional block diagrams and pin allocations are all included.

9.1 Overall Software Functional Description

The functional block diagram for the overall software of the Laser Harp is illustrated below in Figure 40.

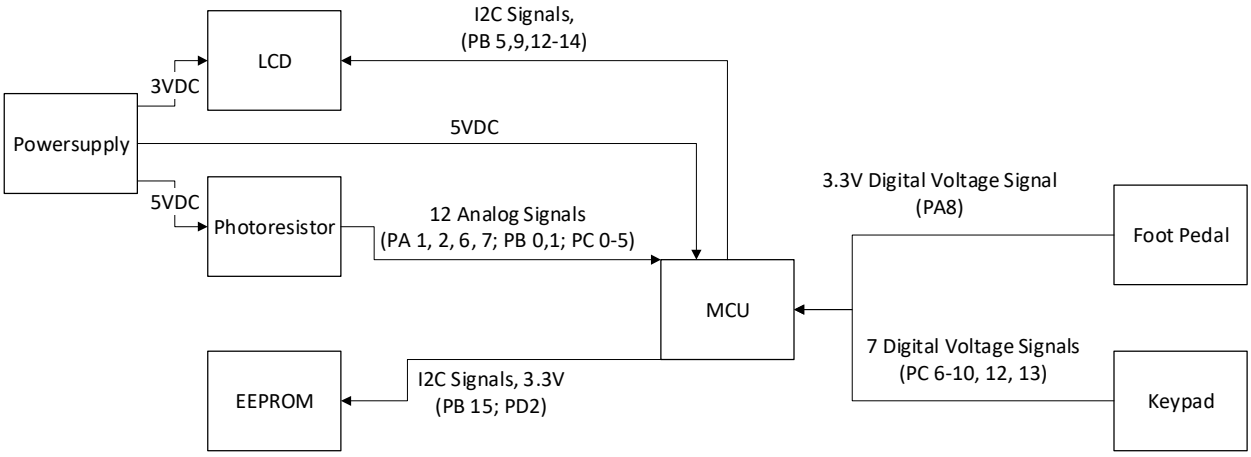


Figure 40: Overall Software Functional Block Diagrams with GPIO pins.

In Table 21 describes the specific hardware that will be connected directly to the MCU. The Laser Harp will be using an LCD, a keypad, a foot pedal, an EEPROM and 12 **photoresistors**.

Table 21: Specific hardware inputs and outputs from device and MCU.

Hardware	**Device Pin Name**	**Device Pin#**	**MCU Pin#**
LCD [7]	RST	1	PB12
	SCL	2	PB13
	SDA	3	PB14
	VSS	4	GND
	VDD	5	3.3V
	VOUT	6	GND
	C1+	7	LCD Pin 8
	C1-	8	LCD Pin 7

Hardware	Device Pin Name	Device Pin#	MCU Pin#
Keypad	Pin 3	1	PC6
	Pin 1	2	PC7
	Pin 5	3	PC8
	Pin 2	4	PC9
	Pin 7	5	PC10
	Pin 6	6	PC12
	Pin 4	7	PC13
Foot Pedal	Positive	1	3.3V
	Negative	2	PA8
EEPROM	CS	1	GND
	SO	2	GND
	WP	3	GND
	Vss	4	GND
	SI	5	PB15
	SCK	6	PD2
	HOLD	7	GND
	Vcc	8	3.3V
Photoresistors	Note #1	1	PA1
	Note #2	2	PA2
	Note #3	3	PA6
	Note #4	4	PA7
	Note #5	5	PB0
	Note #6	6	PB1
	Note #7	7	PC0
	Note #8	8	PC1
	Note #9	9	PC2
	Note #10	10	PC3
	Note #11	11	PC4
	Note #12	12	PC5

9.2 Keypad and Foot Pedal Program

This project is using four by three keypad, which is shown in figure 41. This keypad needs seven 220 ohms resistors for each pin to configure it as no pull-up and pull-down internally. [8]

Figure 41:4x3 Keypad from SparkFun Electronic Datasheet. [9]

The code for the keypad is keypad_function and keypad_demos. The keypad_function consists of assigned GPIOC for row and column pins of the keypad. Meanwhile, Keypad_demos consists of getting the ASCII value to send to USART ports and displays it on the LCD. This keypad is using PC6, PC7, and PC8 for the keypad column pins. Meanwhile, PC9, PC10, PC12, and PC13 for keypad row pins. The set up for this keypad is in Figure 42.

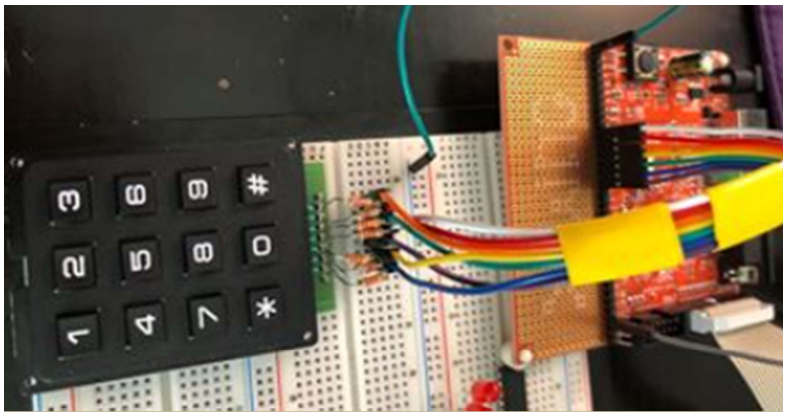


Figure 42: Laser Harp Keypad Set-up

USART_function consists of definitions for the functions files in the project to transmit it to the demo code of the keypad. It also includes the interrupts needed on every program such as the USART_IRQHandler.

USART function to enable interrupts description,

FUNCTION: void USARTx_Init(uint32_t USART_num, USART_TypeDef* USARTx,
uint32_t USARTx_CLK_EN_bit_mask,
uint32_t num_stop_bits, uint32_t parity,
uint32_t parity_select, uint32_t oversampling,
uint32_t baud_rate, uint32_t dma_tx, uint32_t dma_rx,
uint32_t interrupt_mask, uint32_t USARTx_IRQn,
uint32_t priority)

Foot Pedal Code:

The code foot pedal is GPIO_function and GPIO_demo. The purpose of this foot pedal is to change the **octave** of the Laser Harp. The LCD indicates whether the user is holding the note or releasing it. The foot pedal is like a regular push-button by wiring the brown wire to power and the blue wire to GPIOA pin 8 with a 220-ohm resistor. The set up for the foot pedal is in figure 43. The resistor is needed to set it up like an RC debouncer. [9] This debouncing effect reduces the bouncing effect of the signal coming from the hardware. [9] GPIO_Function consists of the characteristic function of the PA8 with the SYSTICK function delay for the output. Meanwhile, GPIO_demos consist of the program of how to hold and release the note signal produced by the Laser Harp.

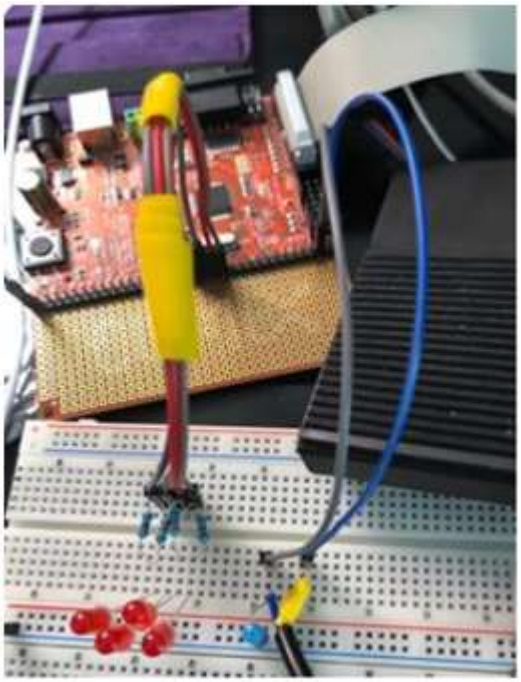


Figure 43:The Laser Harp Foot pedal set up

9.3 LCD Program

The LCD program is designed to work along side the keypad and EEPROM.The following functions describes how the LCD's display menu options to work along side the keypad and EEPROM. There will be two main menu options.

The LCD will be connected to the MCU, and the pin allocations can be seen in Table 21. Along with the MCU the LCD will also have two 1µF capacitors and two 10kΩ resistors. The schematic can be seen in Figure 44.

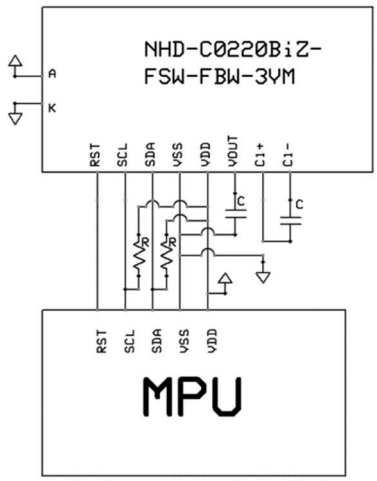


Figure 44: LCD Connections with the MCU.

Source: NHD-C0220BiZ-FSW-FBW-3V3M Datasheet [7]

The first menu option being "1. Record a Song." Once the '1 on the keypad is pressed the display will immediately show that it is recording and to stop recording the user will have to press '3' to stop recording. The recording will immediately be saved onto the EEPROM as "Recorded Song [#]"

The second menu option is "2. Recorded Songs." The user will have the option to choose the song they want to play, and immediately after the song is finished playing the display will immediately return to the main menu.

The following functions were use to program the LCD.

The gpio_functions.c file includes one sub-functions.

GPIO Initialization

This function configures GPIOx (x = A, B, C, D, etc.) pins in an arbitrary GPIO mode so that digital I/O pins can be used by any arbitrary STM32F405 peripheral.

The lcd_demos.c file includes two sub-functions.

LCD Show Characters

Writes a 20-char string to the RAM of the LCD

LCD Clear Display

Writes a 20-char string to the RAM of the LCD

LCD Initialization

This function initializes the LCD so that it is ready to display characters onto the screen.

The lcd_functions.c file includes three sub-functions.

I2C Start

Generates START condition for I2C communications. It configures two GPIOx pins as SDA (serial data) and SCL (serial clock). These two pins are configured in output mode and when the SCL pin is HIGH, the SDA pin goes from HIGH to LOW.

I2C Stop

Generates STOP condition for I2C communications. It configures two GPIOx pins as SDA (serial data) and SCL (serial clock). These two pins are configured in output mode and when the SCL pin is HIGH, the SDA pin goes from LOW to HIGH.

I2C Send Data

Sends a byte of data (address or message) using simulated I2C bus protocol. When the data bit is 1, the SDA output level is HIGH and when the data bit is 0, the SDA output level is LOW.

The usart_functions.c file includes three sub-functions.

USARTx Initialize

Configures and initializes any USART using a configurable interrupt.

USARTx Write String

Transmits a string of characters using USART.

USART2 IRQ Handler

Define ISR for USART2 RXNE interrupt.

Finished LCD program

Figure 45 shows a demonstration of the finished program of the LCD's main menu.

Figure 45:LCD Main Menu Demonstration

9.4 EEPROM Program

This program allows notes to be played using the keyboard using the function Keyboard_Piano(). Songs can be recorded, played back, and erased. Songs can from an EEPROM. The EEPROM to be used is the 25LC256. On the EEPROM, Pins 1,2,3,4, and 7 are connected to ground, Pin and Pin 8 is connected to the MCU's 3.3V pin. The main functions of this program can be found in note_functions.c, the file harp_debug_functions.c contains code used to run tests on the hardware. The original design for the harp was **photoresistors** (connected to standard resistors to form **voltage dividers**) using the ADC inputs, in order to detect when the function Harp_using_ADC() allows songs to be played using the MCU's ADCs as inputs instead of the keyboard, however the USART2_TX pin as an ADC input, and therefore cannot be tested using USART2.

In the main Keyboard_Piano() function, the following actions can be performed by using the following inputs:

Press q, w, e, r, t, y, u, 2, 3, 5, 6, 7 to play notes.
Press x to toggle recording.
Press c to playback recorded songs.
Press v to write songs to the EEPROM.
Press b to read songs from the EEPROM.
Press n to play a song by number.
Press m to erase a song by number.

Figure 46 shows a simple demonstration of the finished program writing three recorded notes to the EEPROM.

Figure 46: EEPROM program Write demonstration.

Figure 47 shows the three notes being read from the EEPROM and played back.

Figure 47: EEPROM program Read demonstration.

Figure 48 shows a single song being selected and played.

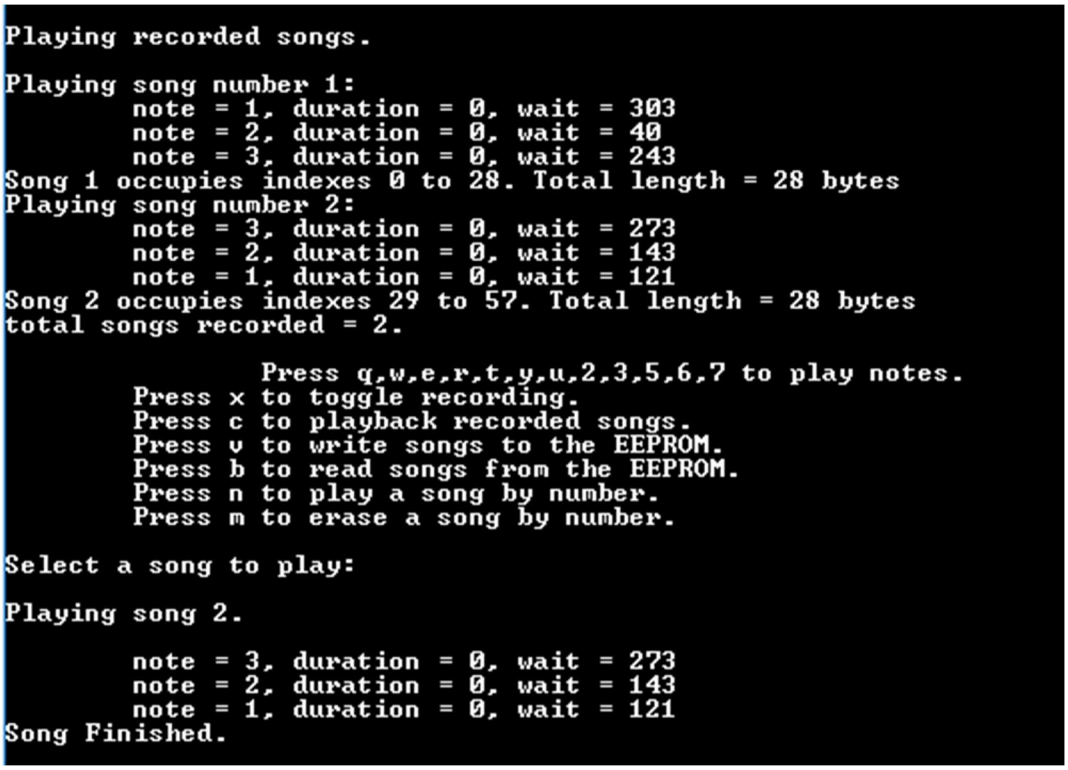


Figure 48: EEPROM program single song playback demonstration.

Figure 49 shows a song being selected and erased.

```
Playing recorded songs.

Playing song number 1:
        note = 1, duration = 0, wait = 578
        note = 2, duration = 0, wait = 207
        note = 3, duration = 0, wait = 223
Song 1 occupies indexes 0 to 28. Total length = 28 bytes
Playing song number 2:
        note = 3, duration = 0, wait = 1032
        note = 2, duration = 0, wait = 264
        note = 1, duration = 0, wait = 192
Song 2 occupies indexes 29 to 57. Total length = 28 bytes
Playing song number 3:
        note = 1, duration = 0, wait = 364
        note = 3, duration = 0, wait = 314
        note = 5, duration = 0, wait = 460
        note = 7, duration = 0, wait = 593
Song 3 occupies indexes 58 to 95. Total length = 37 bytes
total songs recorded = 3.

                Press q,w,e,r,t,y,u,2,3,5,6,7 to play notes.
        Press x to toggle recording.
        Press c to playback recorded songs.
        Press v to write songs to the EEPROM.
        Press b to read songs from the EEPROM.
        Press n to play a song by number.
        Press m to erase a song by number.

Select a song to erase:

Song 2 Erased.


                Press q,w,e,r,t,y,u,2,3,5,6,7 to play notes.
        Press x to toggle recording.
        Press c to playback recorded songs.
        Press v to write songs to the EEPROM.
        Press b to read songs from the EEPROM.
        Press n to play a song by number.
        Press m to erase a song by number.

Playing recorded songs.

Playing song number 1:
        note = 1, duration = 0, wait = 578
        note = 2, duration = 0, wait = 207
        note = 3, duration = 0, wait = 223
Song 1 occupies indexes 0 to 28. Total length = 28 bytes
Playing song number 2:
        note = 1, duration = 0, wait = 364
        note = 3, duration = 0, wait = 314
        note = 5, duration = 0, wait = 460
        note = 7, duration = 0, wait = 593
Song 2 occupies indexes 29 to 66. Total length = 37 bytes
total songs recorded = 2.
```

Figure 49: EEPROM program song erasing demonstration.

10.0 GANTT Chart

Figure 50 shows the GANTT Chart for this project, outlining the work completed and deadlines with their corresponding dates.

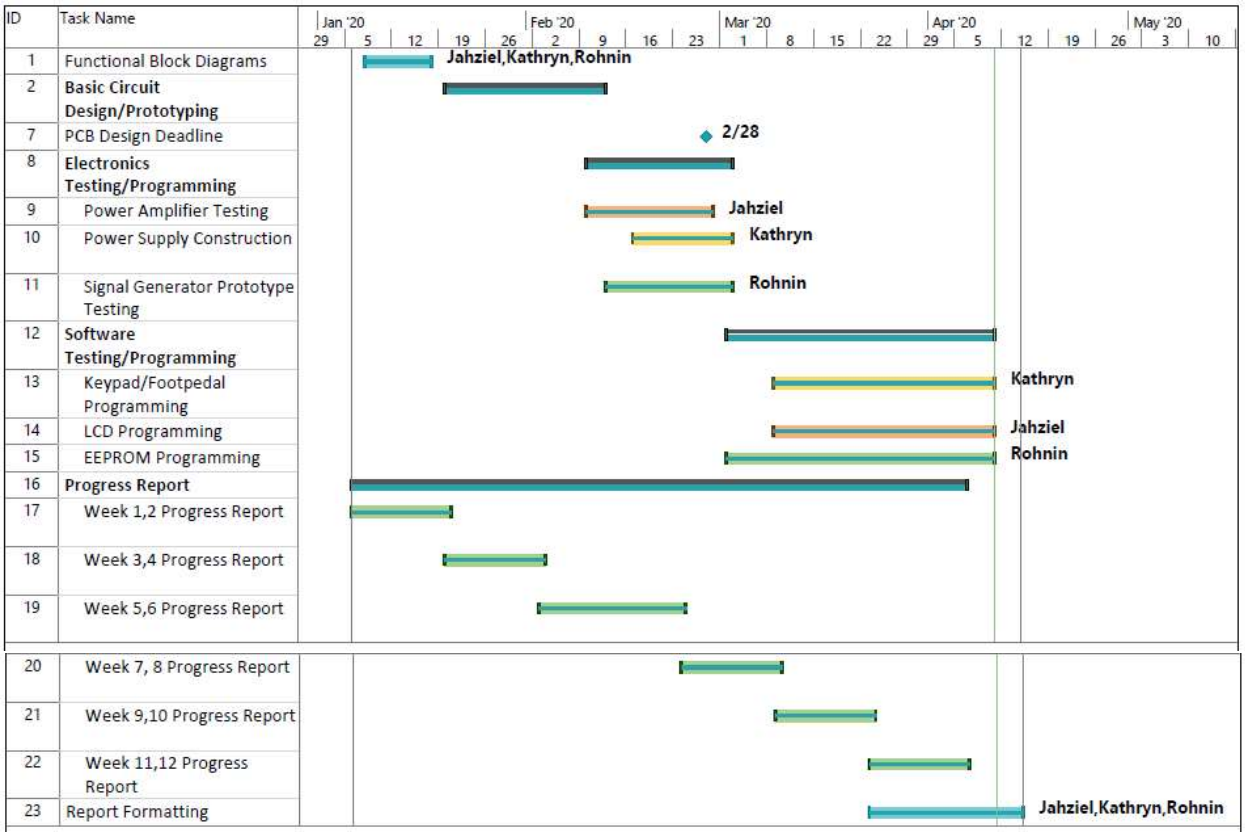


Figure 50: Gantt Chart for this project.

11.0 Costs and Required Resources

This section outlines all parts and resources used during the project, including labor. The cost of each resource is also shown, and the total cost of the project has been calculated.

Power Supply				
Manufacturer Parts #	Designator	Quantity	Unit Price	Amount (CAD)
LM324N	N/A	2	$0.73	$1.46
0.1uF Capacitor	C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, Cs, Cs1	16	$0.31	$4.90

Power Cord (Male)	N/A	2	$5.00	$10.00
LM317T	PA+	1	$0.98	$0.98
LM337TG	PA-	1	$1.15	$1.15
TO-220 Heatsink	N/A	7	$2.93	$20.49
Thermal Pad	N/A	7	$0.39	$2.70
22 Ohm Resistor	R10	4	$0.56	$2.24
Fuse Holder	N/A	2	$3.00	$6.00
166JA12 Transformer	N/A	1	$17.49	$17.49
166K35 Transformer	N/A	1	$51.67	$51.67
1N4004-TP	D2, D3, D5, D6, D7, D8, D9	7	$0.16	$1.10
Glass Fuse	N/A	1	$0.51	$0.51
3- Pin Terminal Block	AC Power	2	$0.62	$1.24
LM317T	MCU, LED, LCD	3	$0.97	$2.91

Power Entry Connector Module	N/A	1	$39.07	$39.07
Power Cord (Female)	N/A	2	$7.68	$15.36
Varistor	N/A	2	$2.90	$5.80
3500uF Capacitor	CF	1	$3.36	$3.36
15 Ohm 2W Resistor	R3, R9	2	$1.28	$2.56
100 Ohm 2W Resistor	R5, R7	2	$1.25	$2.50
200 Ohm 2W Resistor	R2, R4	3	$1.47	$4.41
300 Ohm 2W Resistor	R6, R8	2	$1.47	$2.94
300 Ohm 5W Resistor	Rebleed	1	$0.81	$0.81
220 Ohm 5W Resistor	RBleed1, RBleed 2	2	$5.12	$10.24
10 Ohm 2W Resistor	RS, RS1	2	$1.25	$2.50
Bridge Rectifier	D1, D4	3	$0.86	$2.58
3300uF Capacitor	CF1, CF2	2	$2.73	$5.46

PTC Resettable Fuse	F1, F2	4	$1.49	$5.96
2-Pin Terminal Block	PA+, PA-, SG+, SG-, MCU, LED, LCD	5	$1.57	$7.85
Shoulder Washer	PA+, PA-, SG+, SG-, MCU, LED, LCD	14	$0.17	$2.37
MC7812CTG	SG+	5	$0.93	$4.65
MC7912CTG	SG-	5	$0.91	$4.55
Subtotal				**$247.80**

Power Amplifier				
Manufacturer Parts #	Designator	Quantity	Unit Price	Amount (CAD)
0.1uF Capacitor	CdP, CdN	2	$0.31	$0.61
10k Ohm 0.5 W Potentiometer	R1	1	$1.13	$1.13
Knob Knurl with Skirt	N/A	1	$1.45	$1.45
NE5534	U1	1	$1.50	$1.50
8 Pin Dip IC Socket	U1	1	$0.22	$0.22

0.056uF Capacitor	Cm	1	$0.61	$0.61
TO-220 Heatsink	N/A	2	$2.93	$5.85
MJE15034G	Q1	1	$1.89	$1.89
MJE15035G	Q2	1	$1.89	$1.89
Thermal Pads	N/A	2	$0.39	$0.77
56k Resistor	R3	1	$0.17	$0.17
1 Ohm Resistors	R6, R7	2	$0.51	$1.02
1N4148	D1, D2, D3, D4	4	$0.08	$0.33
0.2 uF Capacitor	C2	1	$2.60	$2.60
1.5uF Capacitor	C1	1	$0.80	$0.80
2.2k Resistor	R4a, R4b, R5a, R5b	4	$0.16	$0.64
2N3904	Q3	1	$0.28	$0.28
2N3906	Q4	1	$0.28	$0.28

2-Pin Screw Terminal	P20, N20, Output	3	$0.52	$1.56
1000uF	CBig1, CBig2	2	$1.56	$3.12
100uF	C3, C4	2	$0.35	$0.70
Shoulder Washers	N/A	2	$0.17	$0.34
#6 Nylon Washers	N/A	2	$0.16	$0.32
1k Ohm Resistor	R8, R9	2	$0.16	$0.32
8 Ohm 1W Speaker	N/A	$1.00	$8.98	$8.98
Subtotal				**$21.49**

Signal Generator				
Manufacturer Parts #	Designator	Quantity	Unit Price	Amount (CAD)
TL082	U1, U2, U3	3	$1.09	$3.27
1N4148	D3, D4, D5, D6	4	$0.15	$0.60
Capacitors	C1, C2, C3, C4, C5, C6, C7	7	$0.07	$0.49

1N5231	D1, D2	2	$0.19	$0.38
2N7000	Q1, Q2	2	$0.23	$0.46
Resistors	R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24	24	$0.07	$1.68
Header 2	Output	1	$0.17	$0.17
Header 3	Power	1	$0.27	$0.27
Header 4	MCU	1	$0.35	$0.35
Sub-total				**$7.67**

LCD				
Manufacturer Parts #		# of items	Unit Price	Amount (CAD)
LCD	N/A	1	$17.64	$17.64
10k Ohm	N/A	3	$0.07	$0.22

1uF	N/A	3	$0.37	$1.11
Subtotal				**$18.97**

Other Material				
Manufacturer Parts #	Designator	# of items	Unit Price	Amount (CAD)
Laser LEDs	LD1, LD2, LD3, LD4, LD5, LD6, LD7, LD8, LD9, LD10, LD11, LD12	12	$1.40	$16.78
Photoresistors	PR1, PR2, PR3, PR4, PR5, PR6, PR7, PR8, PR9, PR10, PR11, PR12	12	$0.11	$1.32
Foot pedal	N/A	1	$8.55	$8.55
1/4" Female Headphone Jack	N/A	1	$3.83	$3.83
Laser LEDs	LD1, LD2, LD3, LD4, LD5, LD6, LD7, LD8, LD9, LD10, LD11, LD12	12	$1.40	$16.78
Photoresistors	PR1, PR2, PR3, PR4, PR5, PR6, PR7, PR8, PR9, PR10, PR11, PR12	12	$0.11	$1.32

Keypad	N/A	1	$7.31	$7.31
PCB	N/A	3	100	$300.00
Subtotal				**$355.88**
			Total Cost:	**$680.97**

The total cost of all materials is $680.97.

11.1 Total Expenses

With an average of 20 hours of work done per week, per person. The total hours spent working on the Laser Harp is 720 hours. With an hourly rate of $22.00 (CAD) per hour. The total labour cost is $15,840.

The total production cost of the Laser Harp is $16,520.97.

12.0 Conclusion

The Electronic Laser Harp was successfully designed after a series of calculations and testing prototypes of each subsystem. The Laser Harp has laser LEDs and can generate sound by blocking the light beam and an amplifier. The harp is also able to record, save, and erase songs using the EEPROM. The Laser Harp could change the **octave** by pressing and holding the foot pedal. The information from the MCU could be controlled by the keypad and displayed on the LCD. The Laser Harp performs in accordance with the requirements and specifications defined in this report. The IEEE format of Laser Harp documentation with and marks the completion of the Capstone Project.

Before the Laser Harp could be constructed, the COVID-19 pandemic resulted in the shut-down of all labs within SAIT. The progress made during the project has been documented, despite the inability to present a final product. In accordance, we programmed each hardware device individually as if they were connected. This programmed software was completed on the 14th week of this semester.

Reference

[1] "*Low Voltage/Filament - Economical Single Primary 166 Series*", Manufacturer Hammond, [Online]. Available: https://www.hammfg.com/electronics/transformers/power/166.pdf. [Accessed March 2020].

[2] G. Smith, "*ELTR 300 Lab 9 Dual Bipolar Power Supply*," SAIT, 2018.

[3] W. Hill and P.Horowitz , "*From ac line to unregulated supply,"* in The Art of Electronics Third Edition, New York: Cambridge University Press, 2015, pp. 630-636.

[4] T. L. Floyd, "*Fixed Positive Linear Voltage Regulators*," in Electronic Devices: Convetional Current, 10th Edition. England: Pearson Education, 2018, pp.878-879..

[5] G. Smith, *"ELTR 270 Lab 10 Power Amplifiers I: The BJT Boosted Op-Amp"*, SAIT, 2018.

[6] *"2N7000G"*, ON Semiconductor, 2011. [Online]. Available: https://www.onsemi.com/pub/Collateral/2N7000-D.PDF. [Accessed 13 4 2020].

[7] *"NHD-C0220BiZ-FSW-FBW-3V3",* New Haven Display International, Elgin, IL.[Online].Available: http://www.newhavendisplay.com/specs/NHD-C0220BiZ-FSW-FBW-3V3M.pdf

[8] *"Keypad 12- Button: COM-14662",* Sparkfun Electronics.[Online].Available: https://media.digikey.com/pdf/Data%20Sheets/Sparkfun%20PDFs/COM-14662_Web.pdf

[9] Y. Zhu, "*14.9 Keypad Scan,"* in Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, 3rd Edition. USA: E-Man Press LLC,2017.pp.365.

[10] H. Islam, *Lecture 1 Inter-Integrated Circuit (I^2C)*, SAIT, 2020

Glossary

Bi-Polar Power Supply	A standard power supply that has a positive and negative terminal.
Diode Biasing	Push-pull transistors allows the circuit's voltage to be stable even when the power supply is fluctuating its voltage.
Electronically-Erasable Programmable Read-Only Memory	Memory that can be electronically erased and written to multiple times.
Electronics Engineering Technology	A program at SAIT
Inter-Integrated Circuit	A bus interface protocol that enables bi-directional serial communications between the microprocessor and their peripheral devices using two wires (SDA and SCL) [10]
Metal Oxide Semiconductor Effect Transistor	An electronic component capable of varying current flow based on a voltage applied to one of its inputs.
Octave	The frequency range between two notes, where one is twice the frequency of the other. Eight notes exist in this range, including the upper and lower notes.
Photoresistors	Resistors capable of changing resistance based on its exposure.
Sawtooth Wave	A waveform composed of an oscillating ramp voltage with a fast rise time in one direction only, so that the waveform resembles a saw blade.
STM32 ARM Cortex Microprocessor	A model of microcontroller used in this project.
Voltage Divider	A series of resistors connected together to divide a voltage source into a lower measurable value.
Voltage-Controlled Attenuator	A device used in this project to change the amplitude of a waveform based on an analog input voltage.
Voltage-Controlled Oscillator	A device that generates a waveform with a frequency that can be manipulated by an analog voltage input.
Waveform Converter	A device used in the project to convert a square wave into a Sawtooth Wave.

Appendix A: List of Abbreviations

Abbreviation	Definition
AVC	Amplifier Volume Control
BR	Bridge Rectifier
DN	Dumping Network
ENT	Electronics Engineering Technology
FP	Foot pedal
FR	Functional Requirements
GDC	General Design Constraints
GPIO	General-Purpose Input/Output
HW	Hardware
I2C	Inter-Integrated Circuit
KP	Keypad
LVC	Line Voltage Capacitor
LPF	Lowpass Filter
MOSFET	Metal Oxide Semiconductor Effect Transistor
NS	Note Selector
OP	Operational Amplifier
OM	Options Menu
Osc	Oscillator
PR	Photoresistors
PA	Power Amplifier
PS	Power Supply
Reg	Regulator
RVD	Resistor Voltage Divider
RXNE	Receiver Not Empty
SC	Signal Conditioning
SCL	Serial Clock
SDA	Serial Data
SG	Signal Generator
SW	Software
SAIT	Southern Alberta Institute of Technology
SCap	Storage Capacitors
SP	Surge Protection
TR	Transformer
TCP	Transient Current Amplifier
TS	Transient Suppressor
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
VCA	Voltage-Controlled Attenuator
WFC	Waveform Convertor

Appendix B: Code

Keypad code and Foot Pedal Code:

This code allows the keypad work with the LCD, EEPROM and foot pedal Code:

```
/*-----------------------------------------------------------------------------
FILENAME: gpio_demos.c
HEADER: mcro_include.h
DESCRIPTION: This file contains different GPIO applications.
It calls the functions defined in the file gpio_functions.c with the parameters required for a particular application.
HARDWARE: Olimex development board STM32-P405
REFERENCES: STM32F405 datasheet
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
-----------------------------------------------------------------------------*/
#include "mcro_include.h"
/*-----------------------------------------------------------------------------
The following function initialized GPIOC pin PC12 as output pin and turns on STAT LED of the Olimex board by driving PC1 low by setting Bit[28] in GPIOC_BSRR register
-----------------------------------------------------------------------------*/
void LED_ON(void)
{
/* Configure GPIOC registers to initialize the LED pin PC12 */
GPIOx_Init(GPIOC,              /* Define a pointer that points to the base address of GPIOC */
GPIOC_CLK_EN_BIT_MASK,  /* Enable the clock of GPIOC by setting Bit[2] in RCC_AHB1ENR */
12,                            /* GPIOC pin number 12 */
1,                             /* Set pin 12 in output mode by writing 0x01 to Bits[25:24] in GPIOC_MODER */
0,                             /* Since the mode is not AF, use any number except 2 */
0,                            /* Set output type to push-pull by clearing Bit[12] in GPIOC_OTYPER */
2,                            /* Set GPIO speed as HIGH speed by writing 0b10 to Bits[25:24] GPIOC_OSPEEDR */
                    0)        /* Select "no pull-up pull-down by clearing Bits[25:24] in GPIOC_PUPDR */
;
GPIOC->BSRR |= 1 << 28;    /* Drive PC12 LOW by setting Bit[28] in GPIOC_BSRR register */
}
/*-----------------------------------------------------------------------------
The following function initialized GPIOC pin PC12 as output pin and turns off STAT LED of the Olimex board by driving PC1 high by setting Bit[16] in GPIOC_BSRR register
-----------------------------------------------------------------------------*/
void LED_OFF(void)
{
/* Configure GPIOC registers to initialize the LED pin PC12 */
GPIOx_Init(GPIOC,              /* Define a pointer that points to the base address of GPIOC */
GPIOC_CLK_EN_BIT_MASK, /* Enable the clock of GPIOC by setting Bit[2] in RCC_AHB1ENR */
12,                            /* GPIOC pin number 12 */
1,                             /* Set pin 12 in output mode by writing 0x01 to Bits[25:24] in GPIOC_MODER */
0,                             /* Since the mode is not AF, use any number except 2 */
0,                            /* Set output type to push-pull by clearing Bit[12] in GPIOC_OTYPER */
2,                            /* Set GPIO speed as HIGH speed by writing 0b10 to Bits[25:24] GPIOC_OSPEEDR */
                    0)        /* Select "no pull-up pull-down by clearing Bits[25:24] in GPIOC_PUPDR */
;
GPIOC->BSRR |= 1 << 12;    /* Drive PC12 high by setting Bit[12] in GPIOC_BSRR register */
}
/*-----------------------------------------------------------------------------
The following demo uses the functions LED_ON() and LED_OFF() to blink the STAT LED
```

```
of the Olimex board by turning it ON or OFF by a specified amount of time. This demo is run from mcro_main.c file.
-------------------------------------------------------------------------------------*/
void LED_Blink(void)
{u
int8_t msg[] = "\n\r\t\tRecording press * to stop and save\n\r\n";
/* Display message on TeraTerm */
USARTx_Write_Str(USART2, msg);
while(1)
{
/* If the user hits Esc key, return to the main menu */
if(return_to_main()) return;
/* Turn on LED */
LED_ON();
delay_ms_systick(500);
/* Turn off LED */
LED_OFF();
/* Introduce a short amount of delay */
delay_ms_systick(500);
}
}
/*-------------------------------------------------------------------------
The following demo turns STAT LED ON or OFF based on foot pedal input. If the user presses the foot pedal, the LED is
turned OFF. If the user releases the foot pedal, the LED is turned ON. This demo is run from mcro_main.c file.
-------------------------------------------------------------------------*/
void LED_control_using_footpedal(void)
{
/* Message prompt for the user */
uint8_t msg[] = "\n\r\t\tPress the footpedal and observe the board\n\r\n";
/* Display message on TeraTerm */
USARTx_Write_Str(USART2, msg);
/* Configure GPIOA registers to initialize the foot pedal pin PA8 */
GPIOx_Init(GPIOA,             /* Define a pointer that points to the base address of GPIOA */
GPIOA_CLK_EN_BIT_MASK, /* Enable the clock of GPIOA by setting Bit[0] in RCC_AHB1ENR */
8,                             /* GPIOA pin number 8 */
8,                              /* Set pin 8 in input mode by clearing Bits[17:16] in GPIOA_MODER*/
0,                              /* Since the mode is not AF, use any number except 2 */
8,                             /* Set output type to push-pull by clearing Bit[0] in GPIOA_OTYPER*/
2,                             /* Set GPIO speed as HIGH speed by writing 0b10 to Bits[1:0] in GPIOA_OSPEEDR */
16                              /* Select "no pull-up pull-down by clearing Bits[1:0] in GPIOA_PUPDR */
);
while(1)
{
/* If the user hits * key, return to the main menu */
if(return_to_main()) return;
/* Check GPIOA_IDR register if the push button is pressed. If the foot pedal is pressed, Bit[8] of GPIOA_IDR would be set.
*/
if(GPIOA->IDR & (1 << 8))
{
uint8_t on[] = "\n\r\t\tHolding the note!! \n\r\n";
/* If Bit[8] of GPIOA_IDR is set, turn on LED */
LED_ON();
USARTx_Write_Str(USART2, on);
delay(0xff555);
}
/* Else if Bit[8] of GPIOA_IDR is not set, turn off LED */
```

```
else {
uint8_t off[] = "\n\r\t\tReleasing the note!! \n\r\n";
LED_OFF();
USARTx_Write_Str(USART2, off);
delay(0xff555);
}
}}
/*-------------------------------------------------------------------------
-------------------------------------------------------------------------*/
void four_LED_control_using_footpedal(void)
{
/* Message prompt for the user */
uint8_t msg[] = "\n\r\t\t Connect 4 LED to PB12, PB13, PB14 and PB15.Then, Press the footpedal and
observe the board\n\r\n";
/* Display message on TeraTerm */
USARTx_Write_Str(USART2, msg);
/* Configure GPIOA registers to initialize the foot pedal pin PA8 */
GPIOx_Init(GPIOA,              /* Define a pointer that points to the base address of GPIOA */
GPIOA_CLK_EN_BIT_MASK, /* Enable the clock of GPIOA by setting Bit[0] in RCC_AHB1ENR */
8,                             /* GPIOA pin number 8 */
8,                             /* Set pin 8 in input mode by clearing Bits[17:16] in GPIOA_MODER*/
0,                              /* Since the mode is not AF, use any number except 2 */
8,                             /* Set output type to push-pull by clearing Bit[0] in GPIOA_OTYPER*/
2,                              /* Set GPIO speed as HIGH speed by writing 0b10 to Bits[1:0] in GPIOA_OSPEEDR */
16                             /* Select "no pull-up pull-down by clearing Bits[1:0] in GPIOA_PUPDR */
);
*(unsigned int*)(0x40023830) |= 1<<1;            // enable GPIOB clock
*(unsigned int*)(0x40020400) &= ~(0xff<<24); // clear GPIOB moder
*(unsigned int*)(0x40020400) |= 0x55<<24;    // configure GPIOB12…15 as output
int i,j =0, reset = 0;
int r= 0, l=0;
while(1)
{
/* If the user hits * key, return to the main menu */
if(return_to_main()) return;
/* Check GPIOA_IDR register if the push button is pressed. If the foot pedal is pressed, Bit[8] of GPIOA_IDR would be set.
*/
if(GPIOA->IDR & (1 << 8))
{
uint8_t on[] = "\n\r\t\tHolding the note!! \n\r\n";
for (i=0; i<4;i++)
{
reset = (0xf<<28);
*(unsigned int*)(0x40020400 + 0x18) = reset;
*(unsigned int*)(0x40020400 + 0x18) = 1<<i+12; // set bit i high
delay(0x5ffff);
}
USARTx_Write_Str(USART2, on);
}
/* Else if Bit[8] of GPIOA_IDR is not set, turn off LED */
else
{
uint8_t off[] = "\n\r\t\tReleasing the note!! \n\r\n";
for (i=3;i>=0;i--)//rotate right
{
```

```
reset = (0xf<<28);
*(unsigned int*)(0x40020400 + 0x18) = reset;
*(unsigned int*)(0x40020400 + 0x18) = 1<<i+12; // set bit i high
delay(0x5ffff);
}
USARTx_Write_Str(USART2, off);
}
}
}

/*-----------------------------------------------------------------------------
FILENAME: gpio_functions.c
HEADER: __cross_studio_io.h, stm32f4xx.h
DESCRIPTION: This file contains the definitions of all the functions that are used to control input and output pins by
writing to or reading from GPIOx registers.
REFERENCES: STM32F405 Datasheet
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
-----------------------------------------------------------------------------*/
#include "mcro_include.h"
/*-------------------------------------------------------------------------------------------------
FUNCTION: void GPIOx_Init(GPIO_TypeDef* GPIOx,
uint32_t GPIOx_CLK_EN_bit_mask,
uint32_t GPIOx_pin_number,
uint32_t GPIOx_mode,
uint32_t periph_AF_number,
uint32_t GPIOx_output_type,
uint32_t GPIOx_output_speed,
uint32_t GPIOx_pupd)
DESCRIPTION: This function configures GPIOx (x = A, B, C, D, etc.) pins in an arbitrary GPIO
mode
so that digital I/O pins can be used by any arbitrary STM32F405 peripheral.
PARAMETERS: GPIOx - Pointer to the base address of GPIOC
GPIOx_pin_number - GPIOx pin number
GPIOx_mode - Selected Mode (e.g. Input, Output, Analog, Alternate Function)
periph_AF_number - AF number associated with the selected peripheral
GPIOx_output_type - GPIOx output type (e.g. push-pull, open-drain)
GPIOx_output_speed - GPIOx output speed (e.g. low, medium, high)
GPIOx_pupd - GPIOx pull-up/pull-down mode
RETURNS: None
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
-------------------------------------------------------------------------------------------------*/
void GPIOx_Init(GPIO_TypeDef* GPIOx,
uint32_t GPIOx_CLK_EN_bit_mask,
uint32_t GPIOx_pin_number,
uint32_t GPIOx_mode,
uint32_t periph_AF_number,
uint32_t GPIOx_output_type,
uint32_t GPIOx_output_speed,
uint32_t GPIOx_pupd)
{
/* Enable GPIOx clock by setting corresponding EN bit in RCC_AHB1ENR register */
RCC->AHB1ENR |= GPIOx_CLK_EN_bit_mask;
/* Clear mode of the selected pin by clearing corresponding bit in MODER register */
GPIOx->MODER &= ~(3 << 2*GPIOx_pin_number);
```

```
/* Set the desired mode for the selected GPIOx pin */
GPIOx->MODER |= GPIOx_mode << 2*GPIOx_pin_number;
/* If GPIOx mode is Alternate Function, configure GPIOx_AFRL (AFR[0])
and GPIOx_AFRH (AFR[1]) registers for the GPIOx pin */
if (GPIOx_mode == 2)
{
/* If the GPIOx pin number is 0 to 7 */
if ((GPIOx_pin_number >= 0) && (GPIOx_pin_number <= 7))
{
/* Clear the 4 bits in AFRL register associated with the GPIOx pin */
GPIOx->AFR[0] &= ~(0xF << 4*GPIOx_pin_number);
/* Write the AF number in corresponding 4 bits in GPIOx_AFRL register */
GPIOx->AFR[0] |= periph_AF_number << 4*GPIOx_pin_number;
}
/* If the GPIOx pin number is 8 to 16*/
else if ((GPIOx_pin_number >= 8) && (GPIOx_pin_number <= 16))
{
/* Clear the 4 bits in AFRL register associated with the GPIOx pin */
GPIOx->AFR[1] &= ~(0xF << (4*GPIOx_pin_number - 32));
/* Write the AF number in corresponding 4 bits in GPIOx_AFRL register */
GPIOx->AFR[1] |= periph_AF_number << (4*GPIOx_pin_number - 32);
}
}
/* Clear output type by clearing the bit associated with GPIOx pin in GPIOx_OTYPER register */
GPIOx->OTYPER &= ~(1 << GPIOx_pin_number);
/* Select the desired output type by setting the corresponding bit in GPIOx_OTYPER register */
GPIOx->OTYPER |= GPIOx_output_type << GPIOx_pin_number;
/* Clear output speed by clearing the two bits associated with GPIOx pin in GPIOx_OSPEEDR register
*/
GPIOx->OSPEEDR &= ~(3 << 2*GPIOx_pin_number);
/* Select the desired output speed by writing the desired speed to the corresponding bits in GPIOx
_OTYPER register */
GPIOx->OSPEEDR |= GPIOx_output_speed << 2*GPIOx_pin_number;
/* Clear pull-up/pull-down by clearing the two bits associated with GPIOx pin in GPIOx_PUPDR regist
er */
GPIOx->PUPDR &= ~(3 << 2*GPIOx_pin_number);
/* Select the desired pull-up/pull-down type by writing the desired type to the corresponding bits
in GPIOx_OTYPER register */
GPIOx->PUPDR |= GPIOx_pupd << 2*GPIOx_pin_number;
} /* End of GPIOx_Init */
/*-----------------------------------------------------------------------------------------------
FUNCTION: void GPIOx_EXTI_Init(uint32_t GPIOx_pin_number,
uint32_t EXTICR_EXTIx_mask,
uint32_t EXTICR_EXTIx_PORTx_mask,
uint32_t EXTI_RTSR_TRx_mask,
uint32_t EXTI_FTSR_TRx_mask,
uint32_t EXTI_IMR_MRx_mask,
uint32_t NVIC_EXTIx_IRQn,
uint32_t NVIC_EXTIx_priority)
DESCRIPTION: This function configures an arbitrary GPIOx (x = A, B, C, D, etc.) pin in
input interrupt mode.
PARAMETERS: GPIOx_pin_number - GPIOx pin/line number (0, 1, 2,....,15)
EXTICR_EXTIx_mask - EXTIx bit mask in SYSCFG_EXTICR register (EXTI0_mask 15 << 0
```

```
, EXTI1_mask 15 << 4, EXTI2_mask = 15 << 8,....EXTICR_EXTIx_PORTx_mask - EXTIx port mask in SYSCFG_EXTICR register
(PAx = 0, PBx = 1, PCx = 3, ......) EXTI_RTSR_TRx_mask - TRx bit mask in EXTI_RTSR register (TR0: 1 << 0, TR1: 1 <<1, TR2: 1
<< 3, ......)
EXTI_FTSR_TRx_mask - TRx bit mask in EXTI_FTSR register (TR0: 1 << 0, TR1: 1 <<
1, TR2: 1 << 3, ......)
EXTI_IMR_MRx_mask - MRx bit mask in EXTI_IMR register (MR0: 1 << 0, MR1: 1 << 1,
MR2: 1 << 3, ......)
NVIC_EXTIx_IRQn - IRQ number for EXTIx line
NVIC_EXTIx_priority - Interrupt priority for EXTIx line
RETURNS: None
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
------------------------------------------------------------------------------------------*/
void GPIOx_EXTI_Init(uint32_t GPIOx_pin_number,
uint32_t EXTICR_EXTIx_mask,
uint32_t EXTICR_EXTIx_PORTx_mask,
uint32_t EXTI_RTSR_TRx_mask,
uint32_t EXTI_FTSR_TRx_mask,
uint32_t EXTI_IMR_MRx_mask,
uint32_t NVIC_EXTIx_IRQn,
uint32_t NVIC_EXTIx_priority)
{
/* Enable system configuration clock by setting Bit[14] of RCC_APB2ENR */
RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN_Msk;
/* For GPIO pins 0 t0 3, use SYSCFG_EXTICR1 register */
if((GPIOx_pin_number >= 0) && (GPIOx_pin_number <= 3))
{
/* Clear EXTIx bits of EXTICR1 register */
SYSCFG->EXTICR[0] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[0] |= EXTICR_EXTIx_PORTx_mask;
}
/* For GPIO pins 4 t0 7, use SYSCFG_EXTICR2 register */
else if((GPIOx_pin_number >= 4) && (GPIOx_pin_number <= 7))
{
/* Clear EXTIx bits of EXTICR2 register */
SYSCFG->EXTICR[1] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[1] |= EXTICR_EXTIx_PORTx_mask;
}
/* For GPIO pins 8 t0 11, use SYSCFG_EXTICR3 register */
else if((GPIOx_pin_number >= 8) && (GPIOx_pin_number <= 11))
{
/* Clear EXTIx bits of EXTICR3 register */
SYSCFG->EXTICR[2] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[2] |= EXTICR_EXTIx_PORTx_mask;
}
/* For GPIO pins 8 t0 11, use SYSCFG_EXTICR4 register */
else if((GPIOx_pin_number >= 12) && (GPIOx_pin_number <= 15))
{
/* Clear EXTIx bits of EXTICR4 register */
SYSCFG->EXTICR[3] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[3] |= EXTICR_EXTIx_PORTx_mask;
```

```
}
/* Enable/disable interrupt trigger for rising edge by setting/clearing the
corresponding TRx bit in EXTI_RTSR register*/
EXTI->RTSR |= EXTI_RTSR_TRx_mask;
/* Enable/disable interrupt trigger for falling edge by setting/clearing the
corresponding TRx bit in EXTI_FTSR register*/
EXTI->FTSR |= EXTI_FTSR_TRx_mask;
/* Unmask the interrupt request from line x by setting
the MRx bit in EXTI_IMR register */
EXTI->IMR |= EXTI_IMR_MRx_mask;
/* Set the priority for the external interrupt */
NVIC_SetPriority(NVIC_EXTIx_IRQn, NVIC_EXTIx_priority);
/* Enable NVIC level interrupt for EXTIx */
NVIC_EnableIRQ(NVIC_EXTIx_IRQn);
} /* End of GPIOx_Read_interrupt() */
/*-------------------------------------------------------------------------------------------
-------
FUNCTION: void EXTI0_IRQHandler(void)
DESCRIPTION: Define external interrupt handler for GPIO line 0 (EXTI0). Each time the selected edge arrives on the
external interrupt line 0, a global variable count g_exti0_count is incremented. This value of g_exti0_count can be used
to conditionally run any application.
PARAMETERS: None
RETURNS: None
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
-------------------------------------------------------------------------------------------
--------*/
uint32_t g_exti0_count = 0;
void EXTI0_IRQHandler(void)
{
/* If the pending bit PR0 in EXTI_PR register is set */
if ((EXTI->PR & (1 << 0)) == (1 << 0)) // if ((EXTI->PR & EXTI_PR_PR0) == EXTI_PR_PR0)
{
/* Increment the interrupt event count */
g_exti0_count++;
/* Clear the pending bit by setting the PR0 bit */
EXTI->PR |= 1 << 0; // EXTI->PR |= EXTI_PR_PR0;
} /* End of if ((EXTI->PR & (1 << 0)) == (1 << 0)) */
} /* End of EXTI0_IRQHandler() */
/* -------------------------------------------------------------------------------------------
-----------*/

/*-------------------------------------------------------------------------------
FILENAME: keypad_demos.c
DESCRIPTION: Gets the ASCII value of a key pressed on a 12 button keypad and sends it out of the RS-232 USART serial
port for display on a terminal emulator program. In keypad_testing(), external interrupts are not used. In
keypad_testing_interrupt(), external interrupts are implemented PC9 to PC13 are set up as outputs and connected to the
4 keypad rows. PC6, PC7 and PC8 are set up as inputs and connected to the 3 keypad columns.
HARDWARE: Olimex prototyping board with STM32F405RG microcontroller
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
-------------------------------------------------------------------------------*/
#include "mcro_include.h"
void keypad_testing(void)
{
/* Declare a char variable to store the value of a keypad key */
```

```
uint8_t keypad_key;
/* Declare the message to be displayed on TeraTerm */
uint8_t msg[] = "\r\n****Keypad Test Program****\r\n\r\n\
5- Play the recorded song,\r\n\
6- Saved song ,\r\n\
7- delete song \r\n\
* - return to main\r\n";
/* Write message to USART_DR to be displayed on TeraTerm */
USARTx_Write_Str(USART2, msg);
{
unsigned char key;
/* Displaying the desired functions */
uint8_t play[] = "\r\n****Playing Record 1****\r\n\r\n" ;
uint8_t saved[] = "\r\n****Record 1 saved ****\r\n\r\n";
uint8_t deleted[] = "\r\n****Record 1 deleted ****\r\n";
/* Begin infinite loop */
uint8_t pressed_key[] = "k\r\n";
/* Initialize keypad row and column pins */
Keypad_Init();
while(1)
{
/* If the user hits the ESC button, return to the main menu */
if(return_to_main()) return;
key = keypad_get_key();
/* If no key is pressed, ignore successive commands */
if(key == 0)continue;
switch(key)
{
case '5': USARTx_Write_Str(USART2, play);break;
case '6': USARTx_Write_Str(USART2, saved); break;
case '7': USARTx_Write_Str(USART2, deleted); break;
case '*': return_to_main();break;
}
}
}
}

/*-----------------------------------------------------------------------------------------------
-----------------
FILENAME: keypad_functions.c
HEADER: mcro_main.h
DESCRIPTION: This file contains the definitions of all the functions that are used for testing the keypad.
REFERENCES: STM32F405 Datasheet and Keypad Schematic
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
-----------------------------------------------------------------------------------------------
---------------*/
#include "mcro_include.h"
/*-----------------------------------------------------------------------------------------------
---------------
FUNCTION: void Keypad_Init(void)
DESCRIPTION: Configures and initializes PC0, PC1, and PC2 as keypad column pins and PC3, PC4, PC5, and PC6 as keypad
row pins
PARAMETERS: None
RETURNS: None
AUTHOR: Habib Islam
```

```
Modified by Kathryn Talabucon
-------------------------------------------------------------------------------------------------
----------*/
void Keypad_Init(void)
{
/* Turn on GPIOC port clock by setting GPIOCEN field (Bit[2]) in RCC_AHB1ENR register */
RCC->AHB1ENR |= 1 << 2;
/* Clear MODER bits of PC0, PC1, PC2, PC3, PC4, PC5, and PC6 to set them all in input mode */
GPIOC->MODER &= ~(3 << 0 | 3 << 2 | 3 << 4 | 3 << 6 | 3 << 8 | 3 << 10 | 3 << 12);
/* Set column pins in pull-up mode */
GPIOC->PUPDR |= (1 << 0) | (1 << 2) | (1 << 4); /* Set colum pins in Pull-up mode */
}
/*-------------------------------------------------------------------------------------------------
----------
FUNCTION: void keypad_get_key(void)
DESCRIPTION: Identifies the row and column of pressed key gets the value of the key
PARAMETERS: None
RETURNS: Character
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
-------------------------------------------------------------------------------------------------
----------*/
uint8_t keypad_get_key(void)
{
/* Declare variables to hold row and colum numbers */
uint32_t row, col;
/* Declare a character variable that would store the value of the key */
uint8_t key;
/* Define key map */
uint8_t key_map[4][3] ={{'1','2','3'},
{'4','5','6'},
{'7','8','9'},
{'*','0','#'}};
/* Declare the combined mask for the column pins PC0, PC1, and PC2 */
uint32_t col_mask = 1 << 0 | 1 << 1 | 1 << 2;
/* Declare an array that contains individual mask for column pins PC0, PC1, and PC2 */
const uint32_t read_col[] = {1 << 0, 1 << 1, 1 << 2};
/* Declare an array that contains individual masks for setting row pins PC3, PC4, PC5, and PC6 as o
utput pins */
const uint32_t set_row_output[] = {1 << 6, 1 << 8, 1 << 10, 1 << 12};
/* Declare an array that contains individual masks for driving row pins PC3, PC4, PC5, and PC6 LOW
*/
const uint32_t set_row_low[] = {1 << 19, 1 << 20, 1 << 21, 1 << 22};
/* Declare an array that contains individual masks for driving row pins PC3, PC4, PC5, and PC6 HIGH
*/
const uint32_t set_row_high[] = {1 << 3, 1 << 4, 1 << 5, 1 << 6};
/* Set row pins PC3, PC4, PC5, and PC6 as output pins by writing 0b01 to MODER3 (Bits[7:6]),
MODER4 (Bits[9:8]), MODER5 (Bits[11:10]), and MODER6 (Bits[13:12]), respectively */
GPIOC->MODER = (1 << 6) | (1 << 8) | (1 << 10) | (1 << 12);
/* Drive row pins PC3, PC4, PC5, and PC6 LOW by setting BR3 (Bit[19]),
BR4 (Bit[20]), BR5 (Bits[21]), and BR6 (Bits[22]), respectively */
GPIOC->BSRR = (1 << 19) | (1 << 20) | (1 << 21) | (1 << 22);
/* Introduce a short amount of delay */
delay(0xF);
/* Scan one row at a time */
```

```
for (row = 0; row < 4; row ++)
{
/* Disable driving all row pins by setting them as input pins */
GPIOC->MODER &= ~(3 << 6 | 3 << 8 | 3 << 10 | 3 << 12);
/* Drive one row pin at a time by setting it as output pin */
GPIOC->MODER = set_row_output[row];
/* Drive the active row pin LOW */
GPIOC->BSRR = set_row_low[row];
/* Introduce a short amount of delay */
delay(0xF);
/* Scan one column pin at a time */
for (col = 0; col < 3; col++)
{
/* If the column pin voltage is zero */
if((GPIOC->IDR & read_col[col]) == 0)
{
/* Read the key from key_map array */
key = key_map[row][col];
/* Wait until the key is released */
while((GPIOC->IDR & col_mask) != col_mask);
/* Return the value of the key */
return key;
} /* End of if((GPIOC->IDR & read_col[col]) == 0) */
} /* End of for (col = 0; col < 3; col++) */
/* Drive the active row HIGH */
GPIOC->BSRR = set_row_high[row];
} /* End of for (row = 0; row < 4; row ++) */
/* Drive all row pins High before disabling them */
GPIOC->BSRR = 1 << 3 | 1 << 4 | 1 << 5 | 1 << 6;
/* Disable driving all row pins by setting them as input pins */
GPIOC->MODER &= ~(3 << 6 | 3 << 8 | 3 << 10 | 3 << 12);
} /* End of keypad_get_key(void) */
```

File name: systick_demos

Author: Habib Islam

Modified by Kathryn Talabucon

```
#include "mcro_include.h"
void delay_ms_systick(uint32_t num_ms)
{
SysTick_Init(SysTick_CTRL_ENABLE_Msk|
SysTick_CTRL_TICKINT_Msk|
0, /* Clock source = 42/8 MHz */
42000000/8000,
0
);
for(uint32_t i = 0; i < num_ms; i++)
{
while(!(SysTick->CTRL & SysTick_CTRL_COUNTFLAG_Msk));
}
SysTick->CTRL = 0;
}

#include "mcro_include.h"
```

```
void SysTick_Init(uint32_t STK_CTRL_mask,
uint32_t STK_RELOAD_val,
uint32_t STK_interrupt_priority)
{
/* Disable SysTick before configuring */
SysTick->CTRL = 0;
/* Clear current value register */
SysTick->VAL = 0;
/* Write the desired reload value */
SysTick->LOAD = STK_RELOAD_val - 1;
NVIC_SetPriority(SysTick_IRQn, STK_interrupt_priority);
SysTick->CTRL |= STK_CTRL_mask;
}
uint32_t g_systick_count = 0;
uint32_t g_seconds = 0;
uint32_t g_minutes = 0;
uint32_t g_hours = 0;
uint32_t g_days = 0;
void SysTick_Handler(void)
{
g_systick_count++;
g_seconds++;
if(g_seconds == 60)
{
g_minutes++;
g_seconds = 0;
}
if(g_minutes == 60)
{
g_hours++;
g_minutes = 0;
}
if(g_hours == 24)
{
g_days++;
g_hours = 0;
}
}

/*-----------------------------------------------------------------------------
FILENAME: usart_functions.c
HEADER: mcro_main.h
DESCRIPTION: This file contains the definitions of all the functions that are used for transmitting receiving data using
USART.This file also contains the USART2 interrupt service routine named USART2_IRQHandler()
REFERENCES: STM32F405 Datasheet
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
-----------------------------------------------------------------------------*/
#include "mcro_include.h"

/*-----------------------------------------------------------------------------
FUNCTION: void USARTx_Write_Str(USART_TypeDef * USARTx, uint8_t * str)
DESCRIPTION: Transmits a string of characters using USART
PARAMETERS: USARTx - Pointer to the base address of USARx
str - Pointer to the string to be sent
RETURNS: None
```

```
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
------------------------------------------------------------------------------*/
/* Function prototype declaration */
void USARTx_Write_Str(USART_TypeDef *USARTx, uint8_t *str)
/* Beginning of function definition */
{
uint32_t i = 0; /* Declare and initialize an index for the string elements */
for(i = 0; i < strlen(str); i++)
{
/* Wait until TC (Transmission Complete) bit (Bit[6]) in USART_SR register is set */
while(!(USARTx->SR & USART_SR_TC)); /* while((USARTx->SR & (1 << 6) == 0) */
/* Write a character to be transmitted to the Data Register (USART_DR) */
USARTx->DR = (str[i] & 0xFF);
} /* End of for(i = 0; i < strlen(str); i++) loop */
} /* End of USARTx_Write_Str */
/*------------------------------------------------------------------------------
FUNCTION: uint8_t USARTx_Read_Poll(USART_TypeDef * USARTx)
DESCRIPTION: Reads a character received by USART_DR
PARAMETERS: USARTx - Pointer to the base address of USARx
RETURNS: Character
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
------------------------------------------------------------------------------*/
uint8_t USARTx_Read_Poll(USART_TypeDef * USARTx)
{
/* Wait until RXNE (Bit[5]) in USARTx_SR is set */
while(!(USARTx->SR & (1 << 5)));
/* Read and return the data in USARTx_DR */
return (uint8_t)USARTx->DR;
}
/*------------------------------------------------------------------------------
FUNCTION: void USART2_IRQHandler(void)
DESCRIPTION: Define ISR for USART2 RXNE interrupt
PARAMETERS: None
RETURNS: None
------------------------------------------------------------------------------*/
/* Initialize the number of characters received by USARTx_DR */
uint32_t g_keypad_read_count = 0;
/* Initialize the array that would store the characters read from USARTx_DR */
uint8_t g_keypad_read_buffer[256] = {};
void USART2_IRQHandler(void)
{
/* If RXNE (Bit[5]) flag is set in USART2_SR */
if(USART2->SR & (1 << 5))
{
/* Increment the read count */
g_keypad_read_count++;
/* Read data into read buffer array */
g_keypad_read_buffer[g_keypad_read_count - 1] = USART2->DR;
/* If the read buffer is full */
if(g_keypad_read_count >= 256)
{
/* Clear read buffer array */
memset(g_keypad_read_buffer, 0, 256);
```

```
/* Reset the read count */
g_keypad_read_count = 0;
}
}
}
/*-------------------------------------------------------------------------
FUNCTION: uint32_t return_to_main(void)
DESCRIPTION: This function allows the user to exit from a demo function and return to the main function. After the user
hits ESC on keyboard, the main menu items are displayed on TeraTerm terminal.
PARAMETERS: None
RETURNS: Integer
AUTHOR: Iouri Kourilov
Modified and commented by Habib Islam and Kathryn Talabucon
-------------------------------------------------------------------------*/
uint32_t return_to_main(void)
{
g_keypad_read_count = keypad_get_key();
/* If the number of received characters by usart is non-zero */
if(g_keypad_read_count == 0)
{
/* If the received character is the ESC character */
if(g_keypad_read_buffer[g_keypad_read_count - 1] == '*')
{
/* Clear the read buffer */
memset(g_keypad_read_buffer, 0, 256);
/* Reset the read count to 0 */
g_keypad_read_count = 0;
/* Return to main menu */
return 1;
}
else return 0;
}
}
/*-------------------------------------------------------------------------
FUNCTION: void USART2_Init(void)
DESCRIPTION: Configures and Initializes USART2 using a configurable interrupt
PARAMETERS: None
RETURNS: None
AUTHOR: Habib Islam
Modified by Kathryn Talabucon
-------------------------------------------------------------------------*/
void USART2_Init(void)
{
/* Enable GPIOA clock by setting GPIOAEN (Bit[1]) in RCC_AHB1ENR register */
RCC->AHB1ENR |= 1 << 0;                 // RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
/* Enable USART2 peripheral clock by setting USART2EN (Bit[17]) in RCC_APB1ENR register */
RCC->APB1ENR |= 1 << 17;                // RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
/* Clear MODER2 (Bits[5:4]) and MODER3 (Bits[7:6]) in GPIOA MODER register */
GPIOA->MODER &= ~(0xF << 4);            // GPIOA->MODER &= ~(GPIO_MODER_MODE2|GPIO_MODER_MODE3);
/* Enable alternate function for PA2 and PA3 by writing 0b10 to MODER2 (Bits[5:4]) and MODER3 (Bits[7:6]) in GPIOA
MODER register */
GPIOA->MODER |= (0xA << 4);             // GPIOA->MODER |= (GPIO_MODER_MODE2_1|GPIO_MODER_MODE3_1);
/* Clear AFRL[2] (Bits[11:8]) and AFRL[3] (Bits[15:12]) in GPIO_AFRL register */
GPIOA->AFR[0] &= ~(0xFF<<8);            // GPIOA->AFR[0] &= ~(GPIO_AFRL_AFSEL2|GPIO_AFRL_AFSEL3);
```

```
/* Select AF7 for PA2 and PA3 by writing 0b0111 to AFRL[2] (Bits[11:8]) and AFRL[3] (Bits[15:12]) in GPIO_AFRL register
*/
GPIOA->AFR[0] |= 0x77 << 8;
/* Disable USART2 by clearing UE (Bit[13]) of USART2_CR1 register */
USART2->CR1 &= ~(1 << 13);              // USART2->CR1 &= ~USART_CR1_UE;
/* Configuer for word length of 8 data bits and 1 start bit by clearing M (Bit[12]) of USART2_CR1 register */
USART2->CR1 &= ~(1 << 12);              // USAR2->CR1 &= ~USART_CR1_M
/* Configure for 16x oversampling by clearing OVER8 (Bit[15]) of USART2_CR1 register */
USART2->CR1 &= ~(1 << 15);              // USAR2->CR1 &= ~USART_CR1_OVER8
/* Disable parity control by clearing PCE (Bit[10]) of USART2_CR1 register */
USART2->CR1 &= ~(1 << 10);              // USAR2->CR1 &= ~USART_CR1_PCE
/* Configure for 1 stop bit by clearing STOP (Bits[13:12]) of USART2_CR2 register */
USART2->CR2 &= ~(3 << 12);              // USAR2->CR2 &= ~USART_CR2_STOP
/* Configure for no flow control by clearing CTSE (Bit[9]) and RTSE (Bit[8]) of USART2_CR3 register
*/
USART2->CR3 &= ~(3 << 8);               //USAR2->CR3 &= ~(USART_CR3_RTSE_Msk|USART_CR3_CTSE_Msk)
/* Configure for 9600 baud rate by writing 273 to Bits[15:4] and 7 to Bits[3:0] in USART2_BRR */
USART2->BRR = (273 << 4) | 7;
/* Enable USART2 transmitter and receiver by setting TE (Bit[3])
and RE (Bit[2]) of USART2_CR1 register, respectively */
USART2->CR1 |= 3 << 2;                  // USART2->CR1 |= (USART_CR1_TE|USART_CR1_RE);
/* Enable USART2 by setting UE (Bit[13]) in USART2_CR1 */
USART2->CR1 |= 1 << 13;                 // USART2->CR1 |= USART_CR1_UE;
/* Enable RXNE interrupt by setting RXNE (Bit[5] in USART2_CR1 register */
USART2->CR1 |= 1 << 5;                  // USART2->CR1 |= USART_CR1_RXNE;
/* Set the interrupt priority to highest */
NVIC_SetPriority(38, 0);                // NVIC_SetPriority(USART2_IRQn, 0);
/* Enable NVIC level interrupt for USART2 */
NVIC_EnableIRQ(38);                     // NVIC_EnableIRQ(USART2_IRQn);
}

/*-------------------------------------------------------------------------------------------------
-----------------------------
FILENAME: mcro_main.c
DESCRIPTION: The file mcro_main.c is the entry point to the application.
It uses the peripheral USART2 to transmit data by polling and receive data by interrupt. The function USARTx_Write_Str()
is used to transmit the menu item messages which are displayed on teraterm. The menu items are defined in the char
variable g_main_menu[]. The menu items contain Main Menu and all the demo names of this project. Each menu item is
associated with a character keyboard-pressed by the user. Users are prompted to press a character on the keyboard to
run a particular demo/application. The character inputs from the users are received by the USART receiver using receiver
not empty (RXNE) interrupt. The received characters are stored in the array g_usart2_receive_buffer. When a character
is available at g_usart2_receive_buffer, it is read and assigned to the variable "keyboard_input". If the received character
is not the Esc character, the program calls the demo function related to the received character. The demo function
selection is done by the switch - case construct. Before reading the character, the global interrupt is enabled by CMSIS
function __enable_irq(). All demo functions display the prompt and wait for the ESC character from the TeraTerm.
HARDWARE: Olimex prototyping board with STM32F405RG microcontroller
AUTHOR: Iouri Kourilov
Modified and commented by Habib Islam and Kathryn Talabucon
-------------------------------------------------------------------------------------------------
---------------------------------*/
/* Include the header file mcro350_main.h that includes all the required header files */
#include "mcro_include.h"
/* Define the char array g_main_menu[] */
uint8_t g_main_menu[] = "\n\r\n\r\t\t\tMain Menu\n\r\n\r\
\t1 - Record\r\n\
\t2 - Octave_control_using_footpedal\r\n\
```

```
\t3 - Octave_and_LED_control_using_footpedal\r\n\
\t4 - Settings\r\n";
/* Declare the message that would prompt the user to press * on keyboard to disable a function and g
o back to the main menu */
uint8_t g_ESC_msg[] = "\n\r\t\t------Press * on the keypad to return to the main menu----\n\r\n";
int32_t main(void)
{
unsigned char key;
char str[50];
unsigned char len= 0;
/* Configure GPIOA registers to initialize PA2 as USART2 TxD pin */
GPIOx_Init(GPIOA,                       /* Define a pointer that points to the base address of GPIOA */
GPIOA_CLK_EN_BIT_MASK,                  /* Enable the clock of GPIOA by setting Bit[1] in RCC_AHB1ENR */
2,                                      /* GPIOA pin number 2 */
2,                                      /* Set pin 10 in AF mode by writing 2 to Bits[21:20] in GPIOB_MODER */
7,                                      /* AF number is 7 */
0,                                      /* Set output type to push-pull by clearing Bit[0] in GPIOA_OTYPER*/
2,                                      /* Set GPIO speed as HIGH speed by writing 0b10 to Bits[1:0] in GPIOA_OSPEEDR */
0                                       /* Select "no pull-up pull-down by clearing Bits[1:0] in GPIOA_PUPDR */
);
/* Configure GPIOA registers to initialize PA3 as USART2 RxD pin */
GPIOx_Init(GPIOA,                       /* Define a pointer that points to the base address of GPIOA */
GPIOA_CLK_EN_BIT_MASK,                  /* Enable the clock of GPIOA by setting Bit[1] in RCC_AHB1ENR */
3,                                      /* GPIOA pin number 3 */
2,                                      /* Set pin 10 in AF mode by writing 2 to Bits[21:20] in GPIOB_MODER */
7,                                      /* AF number is 7 */
0,                                      /* Set output type to push-pull by clearing Bit[0] in GPIOA_OTYPER*/
2,                                      /* Set GPIO speed as HIGH speed by writing 0b10 to Bits[1:0] in GPIOA_OSPEEDR */
1                                       /* Select "no pull-up pull-down by clearing Bits[1:0] in GPIOA_PUPDR */
);
/* Configure and initialize USART2 by calling USARTx_Init() function with the desired parameters*/
USARTx_Init(2,                          /* Select USART2 */
USART2,                                 /* Pointer to the base address of USART2 */
USART2_CLK_EN_BIT_MASK,                 /* Set Bit[17] in RCC_APB1ENR to turn on USART2 peripheral clock */
0,                                      /* Configure for 8-bits of character length */
0,                                      /* Configure for 1 stop bit */
0,                                      /* Disable parity control */
0,                                      /* Doesn't matter since parity control is disabled */
0,                                      /* Configure for 16x oversampling */
(273 << 4) | (7 << 0),          /* USARTDIV = f_PCLK/(8*(2-OVER8)*baud_rate) = 42 MHz/(8*(2-0)*9600 bps) =273.4375
                                   Write Mantissa 273 to Bits[15:4] and fraction 0.4375*16 = 7 to Bits[3:0] in USART2_BRR */
0,                                      /* Disalbe DMA transmit */
0,                                      /* Disable DMA receive */
USART_CR1_RXNEIE,                       /* Configure for RXNE interrupt */
USART2_IRQn,                            /* IRQ number for USART2: doesn't matter since no interrupt is configured */
0                                       /* Highest priority */
);
/* Display g_main_menu items */
USARTx_Write_Str(USART2, g_main_menu);
/* Display g_ESC_message */
USARTx_Write_Str(USART2, g_ESC_msg);
/* Enable global interrupt (Note: there are two underscores before the word enable) */
__enable_irq();                         // Enable global interrupt
/* Begin infinite loop */
uint8_t pressed_key[] = "k\r\n";
```

```
/* Initialize keypad row and column pins */
Keypad_Init();
while(1)
{
key = keypad_get_key();
/* If no key is pressed, ignore successive commands */
if(key == 0)continue;
switch(key)
{
case '1': LED_Blink(); break;
case '2': LED_control_using_footpedal(); break;
case '3': four_LED_control_using_footpedal(); break;
case '4': keypad_testing(); break;
default:
str[len] = key;
str[len +1] = 0;
len++;
if (len >= 48) len = 0;
}
/* Display g_main_menu items */
USARTx_Write_Str(USART2, g_main_menu);
} /* End of while(1) */
} /* End of main(void) */
/* Define the delay function. This function asks the processor to wait for a specified amount of time before executing the
next instruction. In the while loop, the processor wait until the selected number 'num_tick' counts down to 0. */
void delay(uint32_t num_tick)
{
/* Wait until 'num_tick' counts down to 0 */
while(num_tick--);
}
```

LCD Code:

gpio_functions.c:

```
/*-----------------------------------------------------------------------------
FILENAME: gpio_functions.c
HEADER: __cross_studio_io.h, stm32f4xx.h
DESCRIPTION: This file contains the definitions of all the functions that are used to control
input and output pins by writing to or reading from GPIOx registers.
REFERENCES: STM32F405 Datasheet
AUTHOR: Habib Islam
-----------------------------------------------------------------------------*/
#include "mcro_include.h"
/*-------------------------------------------------------------------------------------------
FUNCTION: void GPIOx_Init(GPIO_TypeDef* GPIOx, uint32_t GPIOx_CLK_EN_bit_mask, uint32_t GPIOx_pin_number, uint32_t GPIOx_mode,
uint32_t periph_AF_number, uint32_t GPIOx_output_type, uint32_t GPIOx_output_speed, uint32_t GPIOx_pupd)
DESCRIPTION: This function configures GPIOx (x = A, B, C, D, etc.) pins in an arbitrary GPIO mode so that digital I/O pins can be used by any
arbitrary STM32F405 peripheral.
PARAMETERS: GPIOx - Pointer to the base address of GPIOC
            GPIOx_pin_number - GPIOx pin number
            GPIOx_mode - Selected Mode (e.g. Input, Output, Analog, Alternate Function)
            periph_AF_number - AF number associated with the selected peripheral
            GPIOx_output_type - GPIOx output type (e.g. push-pull, open-drain)
            GPIOx_output_speed - GPIOx output speed (e.g. low, medium, high)
            GPIOx_pupd - GPIOx pull-up/pull-down mode
RETURNS: None
AUTHOR: Habib Islam
-------------------------------------------------------------------------------------------*/
void GPIOx_Init(GPIO_TypeDef* GPIOx,
uint32_t GPIOx_CLK_EN_bit_mask,
uint32_t GPIOx_pin_number,
uint32_t GPIOx_mode,
uint32_t periph_AF_number,
uint32_t GPIOx_output_type,
uint32_t GPIOx_output_speed,
uint32_t GPIOx_pupd)
{
/* Enable GPIOx clock by setting corresponding EN bit in RCC_AHB1ENR register */
RCC->AHB1ENR |= GPIOx_CLK_EN_bit_mask;
/* Clear mode of the selected pin by clearing corresponding bit in MODER register */
GPIOx->MODER &= ~(3 << 2*GPIOx_pin_number);
/* Set the desired mode for the selected GPIOx pin */
GPIOx->MODER |= GPIOx_mode << 2*GPIOx_pin_number;
/* If GPIOx mode is Alternate Function, configure GPIOx_AFRL (AFR[0]) and GPIOx_AFRH (AFR[1]) registers for the GPIOx pin */
if (GPIOx_mode == 2)
{
/* If the GPIOx pin number is 0 to 7 */
if ((GPIOx_pin_number >= 0) && (GPIOx_pin_number <= 7))
{
/* Clear the 4 bits in AFRL register associated with the GPIOx pin */
GPIOx->AFR[0] &= ~(0xF << 4*GPIOx_pin_number);
/* Write the AF number in corresponding 4 bits in GPIOx_AFRL register */
GPIOx->AFR[0] |= periph_AF_number << 4*GPIOx_pin_number;
}
/* If the GPIOx pin number is 8 to 16*/
else if ((GPIOx_pin_number >= 8) && (GPIOx_pin_number <= 16))
{
/* Clear the 4 bits in AFRL register associated with the GPIOx pin */
GPIOx->AFR[1] &= ~(0xF << (4*GPIOx_pin_number - 32));
/* Write the AF number in corresponding 4 bits in GPIOx_AFRL register */
GPIOx->AFR[1] |= periph_AF_number << (4*GPIOx_pin_number - 32);
}
}
/* Clear output type by clearing the bit associated with GPIOx pin in GPIOx_OTYPER register */
GPIOx->OTYPER &= ~(1 << GPIOx_pin_number);
```

```
/* Select the desired output type by setting the corresponding bit in GPIOx_OTYPER register */
GPIOx->OTYPER |= GPIOx_output_type << GPIOx_pin_number;
/* Clear output speed by clearing the two bits associated with GPIOx pin in GPIOx_OSPEEDR register */
GPIOx->OSPEEDR &= ~(3 << 2*GPIOx_pin_number);
/* Select the desired output speed by writing the desired speed to the corresponding bits in GPIOx_OTYPER register */
GPIOx->OSPEEDR |= GPIOx_output_speed << 2*GPIOx_pin_number;
/* Clear pull-up/pull-down by clearing the two bits associated with GPIOx pin in GPIOx_PUPDR register */
GPIOx->PUPDR &= ~(3 << 2*GPIOx_pin_number);
/* Select the desired pull-up/pull-down type by writing the desired type to the corresponding bits in GPIOx_OTYPER register GPIOx->PUPDR |=
GPIOx_pupd << 2*GPIOx_pin_number;
} /* End of GPIOx_Init */
/*-----------------------------------------------------------------------------------------------
FUNCTION: void GPIOx_EXTI_Init(uint32_t GPIOx_pin_number, uint32_t EXTICR_EXTIx_mask, uint32_t EXTICR_EXTIx_PORTx_mask, uint32_t
EXTI_RTSR_TRx_mask, uint32_t EXTI_FTSR_TRx_mask, uint32_t EXTI_IMR_MRx_mask, uint32_t NVIC_EXTIx_IRQn, uint32_t
NVIC_EXTIx_priority)
DESCRIPTION: This function configures an arbitrary GPIOx (x = A, B, C, D, etc.) pin in
input interrupt mode.
PARAMETERS: GPIOx_pin_number - GPIOx pin/line number (0, 1, 2,....,15)
            EXTICR_EXTIx_mask - EXTIx bit mask in SYSCFG_EXTICR register (EXTI0_mask 15 << 0, EXTI1_mask 15 << 4,
            EXTICR_EXTIx_PORTx_mask - EXTIx port mask in SYSCFG_EXTICR register (PAx = 0, PBx = 1, PCx = 3, ......)
            EXTI_RTSR_TRx_mask - TRx bit mask in EXTI_RTSR register (TR0: 1 << 0, TR1: 1 << 1, TR2: 1 << 3, ......)
            EXTI_FTSR_TRx_mask - TRx bit mask in EXTI_FTSR register (TR0: 1 << 0, TR1: 1 << 1, TR2: 1 << 3, ......)
            EXTI_IMR_MRx_mask - MRx bit mask in EXTI_IMR register (MR0: 1 << 0, MR1: 1 << 1, MR2: 1 << 3, ......)
            NVIC_EXTIx_IRQn - IRQ number for EXTIx line
            NVIC_EXTIx_priority - Interrupt priority for EXTIx line
RETURNS: None
AUTHOR: Habib Islam
-----------------------------------------------------------------------------------------------*/
void GPIOx_EXTI_Init(uint32_t GPIOx_pin_number,
uint32_t EXTICR_EXTIx_mask,
uint32_t EXTICR_EXTIx_PORTx_mask,
uint32_t EXTI_RTSR_TRx_mask,
uint32_t EXTI_FTSR_TRx_mask,
uint32_t EXTI_IMR_MRx_mask,
uint32_t NVIC_EXTIx_IRQn,
uint32_t NVIC_EXTIx_priority)
{
/* Enable system configuration clock by setting Bit[14] of RCC_APB2ENR */
RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN_Msk;
/* For GPIO pins 0 t0 3, use SYSCFG_EXTICR1 register */
if((GPIOx_pin_number >= 0) && (GPIOx_pin_number <= 3))
{
/* Clear EXTIx bits of EXTICR1 register */
SYSCFG->EXTICR[0] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[0] |= EXTICR_EXTIx_PORTx_mask;
}
/* For GPIO pins 4 t0 7, use SYSCFG_EXTICR2 register */
else if((GPIOx_pin_number >= 4) && (GPIOx_pin_number <= 7))
{
/* Clear EXTIx bits of EXTICR2 register */
SYSCFG->EXTICR[1] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[1] |= EXTICR_EXTIx_PORTx_mask;
}
/* For GPIO pins 8 t0 11, use SYSCFG_EXTICR3 register */
else if((GPIOx_pin_number >= 8) && (GPIOx_pin_number <= 11))
{
/* Clear EXTIx bits of EXTICR3 register */
SYSCFG->EXTICR[2] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[2] |= EXTICR_EXTIx_PORTx_mask;
}
/* For GPIO pins 8 t0 11, use SYSCFG_EXTICR4 register */
else if((GPIOx_pin_number >= 12) && (GPIOx_pin_number <= 15))
{
```

```
/* Clear EXTIx bits of EXTICR4 register */
SYSCFG->EXTICR[3] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[3] |= EXTICR_EXTIx_PORTx_mask;
}
/* Enable/disable interrupt trigger for rising edge by setting/clearing the corresponding TRx bit in EXTI_RTSR register*/
EXTI->RTSR |= EXTI_RTSR_TRx_mask;
/* Enable/disable interrupt trigger for falling edge by setting/clearing the corresponding TRx bit in EXTI_FTSR register*/
EXTI->FTSR |= EXTI_FTSR_TRx_mask;
/* Unmask the interrupt request from line x by setting the MRx bit in EXTI_IMR register */
EXTI->IMR |= EXTI_IMR_MRx_mask;
/* Set the priority for the external interrupt */
NVIC_SetPriority(NVIC_EXTIx_IRQn, NVIC_EXTIx_priority);
/* Enable NVIC level interrupt for EXTIx */
NVIC_EnableIRQ(NVIC_EXTIx_IRQn);
} /* End of GPIOx_Read_interrupt() */
/*-------------------------------------------------------------------------------------------------------
FUNCTION: void EXTI0_IRQHandler(void)
DESCRIPTION: Define external interrupt handler for GPIO line 0 (EXTI0). Each time the selected edge arrives on the external interrupt line 0, a
global variable count g_exti0_count is incremented. This value of g_exti0_count can be used to conditionally run any application.
PARAMETERS: None
RETURNS: None
AUTHOR: Habib Islam
-------------------------------------------------------------------------------------------------------*/
uint32_t g_exti0_count = 0;
void EXTI0_IRQHandler(void)
{
/* If the pending bit PR0 in EXTI_PR register is set */
if ((EXTI->PR & (1 << 0)) == (1 << 0)) // if ((EXTI->PR & EXTI_PR_PR0) == EXTI_PR_PR0)
{
/* Increment the interrupt event count */
g_exti0_count++;
/* Clear the pending bit by setting the PR0 bit */
EXTI->PR |= 1 << 0; // EXTI->PR |= EXTI_PR_PR0;
} /* End of if ((EXTI->PR & (1 << 0)) == (1 << 0)) */
} /* End of EXTI0_IRQHandler() */
/* -------------------------------------------------------------------------------------------------------*/
```

lcd_demos.c:

```
/*-------------------------------------------------------------------------------------------------------
FILE NAME : lcd_demos.c
HEADER: mcro_include.h
DESCRIPTION: This file contains the application of the simulated I2C functions defined in the file lcd_functions.c
Specific applications include reading and writing, and displaying the data to the LCD.
REFERENCES: STM32F405 datasheet, NHD-C0220BiZ-FSW-FBW-3V3M datasheet
HOOKUP:   Connect 3.3V pin of Olimex to breadboard positive voltage rail (PIN 6 on the LCD).
          Connect GND pin of Olimex to breadboard negative voltage rail (PIN 5 on the LCD).
          Connect LCD pin 1 (RST), and pin 5 (Vcc) to breadboard postive voltage rail.
          Connect LCD pin 2 (SCL) to PB13, and pin 3 (SDA) to PB14.
          Connect LCD pins 4 (Vss), and 6 (Vout) to negative voltage rail.
          HARWARE: Olimex development board with STM32F405RG
-------------------------------------------------------------------------------------------------------*/
#include "mcro_include.h"
uint32_t slave = 0x78;
uint32_t com_send = 0x00;
uint32_t data_send = 0x40;
uint32_t clear = 0x01;
/* Declare and initialize an 8-bit unsigned integer variable keyboard_input. This variable will be assigned with the character read from the
g_usart2_receive_buffer */
uint8_t keyboard_input = 0;
uint8_t key_input = 0;
/*-------------------------------------------------------------------------------------------------------
FUNCTION: void LCD_show(unsigned char *text)
DESCRIPTION: Writes a 20-char string to the RAM of the LCD
PARAMETERS: *text - message data to write
```

```
AUTHOR: Jahziel Ortega
-------------------------------------------------------------------------------------------------------*/
void display(unsigned char *text)
{
uint32_t n;
I2C_Start(GPIOB, 14, 13);
I2C_send_data(GPIOB, 14, 13, slave); //Slave address=0x78
I2C_send_data(GPIOB, 14, 13, data_send); //Control byte: ram data bytes follow; data_send address = 0x00
for(n=0; n< 20; n++)
{
I2C_send_data(GPIOB, 14, 13, *text);
text++;
}
I2C_Stop(GPIOB, 14, 13);
}
/*-------------------------------------------------------------------------------------------------------
FUNCTION: clear_display(void)
DESCRIPTION: Writes a 20-char string to the RAM of the LCD
AUTHOR: Jahziel Ortega
-------------------------------------------------------------------------------------------------------*/
/* Sends the "clear display" command to the LCD. */
void clear_display(void)
{
I2C_Start(GPIOB, 14, 13);
/* Slave address of panel. */
I2C_send_data(GPIOB, 14, 13, slave);
/* Control byte: all following bytes are commands. */
I2C_send_data(GPIOB, 14, 13, com_send);
/* Clear display command */
I2C_send_data(GPIOB, 14, 13, clear);
I2C_Stop(GPIOB, 14, 13);
} // end of cleaar_display
/*-------------------------------------------------------------------------------------------------------
FUNCTION: LCD_init(void)
DESCRIPTION: This function initializes the LCD so that it is ready to display characters onto
the screen.
AUTHOR: Jahziel Ortega
-------------------------------------------------------------------------------------------------------*/
void LCD_init(void)
{
I2C_Start(GPIOB, 14, 13);
/* Slave address of the LCD */
I2C_send_data(GPIOB, 14, 13, slave);
/* Control byte: all of the following are commands */
I2C_send_data(GPIOB, 14, 13, com_send);
/* Set as 8-bit bus, 2 lines, normal instruction mode. */
I2C_send_data(GPIOB, 14, 13, 0x38);
delay(10);
/* Set as 8-bit bus, 2 lines, extension instruction mode */
I2C_send_data(GPIOB, 14, 13, 0x39);
delay(10);
/* display on, cursor on, cursor position off */
I2C_send_data(GPIOB, 14, 13, 0x14);
/* Slave address of LCD */
I2C_send_data(GPIOB, 14, 13, slave);
/* ICON on, booster circuit on, contrast set to 2 */
I2C_send_data(GPIOB, 14, 13, 0x5E);
/* Follower cicuit on, follower amplified ratio to 5 */
I2C_send_data(GPIOB, 14, 13, 0x6D);
/* display on */
I2C_send_data(GPIOB, 14, 13, 0x0C);
/* clear display */
I2C_send_data(GPIOB, 14, 13, 0x01);
/* entry mode set to 0110 */
I2C_send_data(GPIOB, 14, 13, 0x06);
delay(10);
```

```
I2C_Stop(GPIOB, 14, 13);
} // end of LCD_init
void LCD_demo(void)
{
/* Configure GPIOB pin PB14 as an I2C SDA pin */
GPIOx_Init(GPIOB, /* Base address of GPIOC is a pointer to GPIO_TypeDef structure */
GPIOB_CLK_EN_BIT_MASK, /* Enable GPIOC clock by setting Bit[0] in RCC_AHB1ENR register */
14, /* Select GPIOC pin to be PC0 */
1, /* Configure PB14 in Output mode by writing 0b01 (1) to Bits[1:0] in GPIOB_MODER register */
0, /* Since GPIO mode is OUTPUT, AF number doesn't matter */
1, /* Configure PB14 output type to be OPEN_DRAIN by setting Bit[0] in GPIOB_OTYPER register */
2, /* Configure PB14 output speed to be HIGH by writing 0b10 to Bits[1:0] in GPIOB_OSPEEDR register */
1 /* Configure PB14 to be PULL-UP by writing 0b01 (1) to Bits[1:0] in GPIOB_PUPDR register */
);
/* Configure GPIOB pin PB13 as an I2C SCL pin */
GPIOx_Init(GPIOC, /* Base address of GPIOB is a pointer to GPIO_TypeDef structure */
GPIOB_CLK_EN_BIT_MASK, /* Enable GPIOB clock by setting Bit[2] in RCC_AHB1ENR register */
1, /* Select GPIOB pin to be PC1 */
13, /* Configure PB13 in OUTPUT mode by writing 0b01 (1) to Bits[3:2] in GPIOB_MODER register */
0, /* Since GPIO mode is OUTPUT, AF number doesn't matter */
1, /* Configure PB13 output type to be OPEN_DRAIN by setting Bit[1] in GPIOB_OTYPER register */
2, /* Configure PB13 output speed to be HIGH by writing 0b10 to Bits[3:2] in GPIOB_OSPEEDR register */
1 /* Configure PB13 to be PULL-UP by writing 0b01 (1) to Bits[3:2] in GPIOB_PUPDR register */
);
LCD_mainmenu();
while(1)
{
/* If no key is pressed, ignore successive commands */
if(g_usart2_read_count == 0) continue;
/* If a key is pressed, assign the received character to the variable keyboard_input */
keyboard_input = g_usart2_read_buffer[g_usart2_read_count-1];
if((g_usart2_read_buffer[g_usart2_read_count-1]) == '*') LCD_mainmenu();
switch(keyboard_input)
{
case '1': {
while(1)
{
/* start recording function will show on the lcd */
start_recording();
/* if the user presses '3' the while loop will break */
if((g_usart2_read_buffer[g_usart2_read_count-1]) == '3') break;
} // end of while (1)
/* Clear read count */
g_usart2_read_count = 0;
/*recording will be saved to the EEPROM */
save_recording();
/*delay by 1 second */
delay_ms_timer(1000);
/* return back to LCD main menu */
LCD_mainmenu();
break;
}//end of case '1'
case '2':
{
recorded_songs();
/*if recorded song is chosen, song will play from EEPROM */
delay_ms_timer(3000);
LCD_mainmenu();
break;
} // end of case '2'
} // end of switch(keyboard_input)
/* Clear read count */
g_usart2_read_count = 0;
}
}
```

lcd_display.c:

```
#include "mcro_include.h"
#include "mcro_include.h"
/* LCD main menu shows option to record song or recorded songs */
void LCD_mainmenu()
{
/* LCD is initialized */
LCD_init();
/*LCD display cleared */
clear_display();
/* First 9 characters are hidden to the left of line 1, last 11 characters shown on line 1*/
display(" 1. Record a");
/* First 9 characters are shown on line 1, last 11 characters hidden to the left of line 1*/
display(" song ");
/* First 9 characters are hidden to the left of line, last 11 characters shown on line 2*/
display(" 2. View Son");
/* First 9 characters are shown on line 2, last 11 characters hidden to the left of line 2*/
display("gs ");
} // end of LCD_mainmenu
void start_recording()
{
/* LCD is initialized */
LCD_init();
/*LCD display cleared */
clear_display();
/* First 9 characters are hidden to the left of line 1, last 11 characters shown on line 2*/
display(" Recording..");
/* First 9 characters are shown on line 1, last 11 characters hidden to the left of line 1*/
display(". ");
/* First 9 characters are hidden to the left of line 2, last 11 characters shown on line 2*/
display(" 3. Stop ");
/* First 9 characters are shown on line 2, last 11 characters hidden to the left of line 2*/
display("Recording ");
} //end of start_recording
void save_recording()
{
/* LCD is initialized */
LCD_init();
/*LCD display cleared */
clear_display();
/* First 9 characters are hidden to the left of line 1
last 11 characters shown on line 1*/
display(" Recording");
/* First 9 characters are shown on line 1
last 11 characters hidden to the left of line 1*/
display(" ");
/* First 9 characters are hidden to the left of line 2
last 11 characters shown on line 2*/
display(" ");
/* First 9 characters are shown on line 2
last 11 characters hidden to the left of line 2*/
display("Saved ");
} // end of save_recording
void recorded_songs()
{
/* LCD is initialized */
LCD_init();
/*LCD display cleared */
clear_display();
/* First 9 characters are hidden to the left of line 2
last 11 characters shown on line 2*/
display(" Recorded s");
/* First 9 characters are shown on line 1
last 11 characters hidden to the left of line 1*/
```

```
display("ong [1] " );
/* First 9 characters are hidden to the left of line 2
last 11 characters shown on line 2*/
display(" Recorded s");
/* First 9 characters are shown on line 1
last 11 characters hidden to the left of line 1*/
display("ong [2] ");
} // end of recorded_songs
```

lcd_functions.c:

```
#include "mcro_include.h"
/*-----------------------------------------------------------------------------------------------
FUNCTION: I2C_Start(GPIO_TypeDef * GPIOx, uint32_t GPIOx_SDA, uint32_t GPIOx_SCL)
DESCRIPTION: Generates START condition for I2C communications. It configures two GPIOx pins as SDA (serial data) and SCL (serial clock). These
two pins are configured in output mode and when the SCL pin is HIGH, the SDA pin goes from HIGH to LOW.
PARAMETERS: GPIOx - Base Address of GPIOx is a pointer to GPIO_TypeDef structure
                    GPIOx_SDA - I2C data pin
                    GPIOx_SCL - I2C clock pin
RETURNS: None
AUTHOR: Habib Islam
Modified by Jahziel Ortega
-----------------------------------------------------------------------------------------------*/
void I2C_Start(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL)
{
/* Set SDA and SCL as output pins by writing 0b01 (1) to corresponding bits in GPIOx_MODER register */
GPIOx->MODER |= (1 << 2*GPIOx_SDA) | (1 << 2*GPIOx_SCL);
/* Drive SDA output level HIGH by setting the corresponding bit in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << GPIOx_SDA;
/* Introduce a small amount of delay */
delay(1);
/* Drive SCL output HIGH by setting the corresponding bit in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << GPIOx_SCL;
/* Introduce a small amount of delay */
delay(1);
/* Drive SDA output level LOW by setting the corresponding bit in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << (16 + GPIOx_SDA);
/* Introduce a small amount of delay */
delay(1);
/* Drive SCL output LOW by setting the corresponding bit in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << (16 + GPIOx_SCL); // set SCL low
/* Introduce a small amount of delay */
delay(1);
} /* End of I2C_Start() */
/*-----------------------------------------------------------------------------------------------
FUNCTION: I2C_Stop(GPIO_TypeDef * GPIOx, uint32_t GPIOx_SDA, uint32_t GPIOx_SCL)
DESCRIPTION: Generates STOP condition for I2C communications. It configures two GPIOx pins as SDA (serial data) and SCL (serial clock). These
two pins are configured in output mode and when the SCL pin is HIGH, the SDA pin goes from LOW to HIGH.
PARAMETERS: GPIOx - Base Address of GPIOx is a pointer to GPIO_TypeDef structure
                    GPIOx_SDA - I2C data pin
                    GPIOx_SCL - I2C clock pin
RETURNS: None
AUTHOR: Habib Islam
Modified by Jahziel Ortega
-----------------------------------------------------------------------------------------------*/
void I2C_Stop(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL)
{
/* Set SDA and SCL as output pins by writing 0b01 (1) to corresponding bits in GPIOx_MODER register */
GPIOx->MODER |= (1 << 2*GPIOx_SDA) | (1 << 2*GPIOx_SCL);
/* Drive SDA LOW by setting the corresponding bit in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << (16 + GPIOx_SDA);
/* Introduce a small amount of delay */
```

```
delay(1);
/* Drive SCL HIGH by setting the corresponding bit in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << GPIOx_SCL;
/* Introduce a small amount of delay */
delay(1);
/* Drive SDA HIGH by setting the corresponding bit in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << GPIOx_SCL;
/* Introduce a small amount of delay */
delay(1);
} /* End of I2C_Stop() */
/*---------------------------------------------------------------------------------------------------
FUNCTION: I2C_send_data(GPIO_TypeDef * GPIOx, uint32_t GPIOx_SDA, uint32_t GPIOx_SCL, uint8_t data)
DESCRIPTION: Sends a byte of data (address or message) using simulated I2C bus protocol. When the data bit is 1, the SDA output levle is HIGH
and when the data bit is 0, the SDA output level is LOW.
PARAMETERS: GPIOx - Base Address of GPIOx is a pointer to GPIO_TypeDef structure
            GPIOx_SDA - I2C data pin
            GPIOx_SCL - I2C clock pin
            data - address or message data to send
RETURNS: None
AUTHOR: Habib Islam
Modified by Jahziel Ortega
---------------------------------------------------------------------------------------------------*/
void I2C_send_data(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL,
uint8_t data)
{
/* Declare and initialize the index for the 'for' loop */
uint32_t i = 0;
/* Set both SDA and SCL as output pin by writing 0b01 (1) to corresponding bits in GPIOx_MODER register */
GPIOx->MODER |= (1 << 2*GPIOx_SDA) | (1 << 2*GPIOx_SCL);
for(i = 0; i < 8; i++)
{
/* If the data value is 1 */
if(1 & (data >> (7-i)))
{
/* Drive SDA HIGH by setting the bit corresponding to SDA pin in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << GPIOx_SDA;
}
/* Else if the data value is 0, drive SDA LOW by setting the bit corresponding to the SDA pin in GPIOx_BSRR register */
else GPIOx->BSRR |= 1 << (16 + GPIOx_SDA);
/* Introduce a small amount of delay */
delay(1);
/* Drive SCL LOW by setting the bit corresponding to SCL pin in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << (16 + GPIOx_SCL);
/* Introduce a small amount of delay */
delay(1);
/* Drive SCL HIGH by setting the bit corresponding to SCL pin in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << GPIOx_SCL;
/* Introduce a small amount of delay */
delay(1);
/* Drive SCL LOW by setting the bit corresponding to SCL pin in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << (16 + GPIOx_SCL);
/* Introduce a small amount of delay */
delay(1);
} /* End of for(i = 0; i < 8; i++) */
/* Drive SCL HIGH by setting the bit corresponding to SCL pin in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << GPIOx_SCL;
while (GPIOx_SDA == 1 << GPIOx_SDA)/* Drive SDA HIGH by setting the bit corresponding to SDA pin in
GPIOx_BSRR register */
{
/* Drive SCL LOW by setting the bit corresponding to SCL pin in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << (16 + GPIOx_SCL);
/* Introduce a small amount of delay */
delay(1);
/* Drive SCL HIGH by setting the bit corresponding to SCL pin in GPIOx_BSRR register */
```

```
GPIOx->BSRR |= 1 << GPIOx_SCL;
/* Introduce a small amount of delay */
delay(1);
}
/* Drive SCL LOW by setting the bit corresponding to SCL pin in GPIOx_BSRR register */
GPIOx->BSRR |= 1 << (16 + GPIOx_SCL);
} /* End of I2C_send_data() */
```

main.c:

```
/*-----------------------------------------------------------------------------------------------
FILENAME: main.c
DESCRIPTION: The file mcro_main.c is the entry point to the application.
It uses the peripheral USART2 to transmit data by polling and receive data by interrupt.
The function USARTx_Write_Str() is used to transmit the menu item messages which are displayed on teraterm.
The character inputs from the users are received by the USART receiver usingreceiver not empty (RXNE) interrupt.
The received characters are stored in the array g_usart2_receive_buffer.
When a character is available at g_usart2_receive_buffer, it is read and assigned to the variable "keyboard_input".
Before reading the character, the global interrupt is enabled by CMSIS function __enable_irq().
HARDWARE: Olimex prototyping board with STM32F405RG microcontroller
AUTHOR: Iouri Kourilov
Modified and commented by Habib Islam, & Jahziel Ortega
-----------------------------------------------------------------------------------------------*/
/* Include the header file mcro350_main.h that includes all the required header files */
#include "mcro_include.h"
/* Define the char array g_main_menu[] */
uint8_t g_main_menu[] = "\n\r\n\r\t\t\tLCD Demo\n\r\n\r";
int32_t main(void)
{
/* Declare and initialize an 8-bit unsigned integer variable keyboard_input. This variable will be
assigned
with the character read from the g_usart2_receive_buffer */
uint8_t keyboard_input = 0;
/* Configure GPIOA registers to initialize PA2 as USART2 TxD pin */
GPIOx_Init(GPIOA, /* Define a pointer that points to the base address of GPIOA */
GPIOA_CLK_EN_BIT_MASK, /* Enable the clock of GPIOA by setting Bit[1] in RCC_AHB1ENR */
2, /* GPIOA pin number 2 */
2, /* Set pin 10 in AF mode by writing 2 to Bits[21:20] in GPIOB_MODER */
7, /* AF number is 7 */
0, /* Set output type to push-pull by clearing Bit[0] in GPIOA_OTYPER */
2, /* Set GPIO speed as HIGH speed by writing 0b10 to Bits[1:0] in
GPIOA_OSPEEDR */
0 /* Select "no pull-up pull-down by clearing Bits[1:0] in GPIOA_PUPDR */
);
/* Configure GPIOA registers to initialize PA3 as USART2 RxD pin */
GPIOx_Init(GPIOA, /* Define a pointer that points to the base address of GPIOA */
GPIOA_CLK_EN_BIT_MASK, /* Enable the clock of GPIOA by setting Bit[1] in RCC_AHB1ENR */
3, /* GPIOA pin number 3 */
2, /* Set pin 10 in AF mode by writing 2 to Bits[21:20] in GPIOB_MODER */
7, /* AF number is 7 */
0, /* Set output type to push-pull by clearing Bit[0] in GPIOA_OTYPER */
2, /* Set GPIO speed as HIGH speed by writing 0b10 to Bits[1:0] in
GPIOA_OSPEEDR */
0 /* Select "no pull-up pull-down by clearing Bits[1:0] in GPIOA_PUPDR */
);
/* Configure and initialize USART2 by calling USARTx_Init() function with the desired parameters*/
USARTx_Init(2, /* Select USART2 */
USART2, /* Pointer to the base address of USART2 */
USART2_CLK_EN_BIT_MASK, /* Set Bit[17] in RCC_APB1ENR to turn on USART2 peripheral clock */
0, /* Configure for 8-bits of character length */
0, /* Configure for 1 stop bit */
0, /* Disable parity control */
0, /* Doesn't matter since parity control is disabled */
0, /* Configure for 16x oversampling */
(273 << 4) | (7 << 0), /* USARTDIV = f_PCLK/(8*(2-OVER8)*baud_rate) = 42 MHz/(8*(2-0)*9600 bps) = 273.4375
Write Mantissa 273 to Bits[15:4] and fraction 0.4375*16 = 7 to Bits[3:0] in USART2_BRR */
```

```
0, /* Disalbe DMA transmit */
0, /* Disable DMA receive */
USART_CR1_RXNEIE, /* Configure for RXNE interrupt */
USART2_IRQn, /* IRQ number for USART2: doesn't matter since no interrupt */
0 /* Highest priority */
);
/* Display g_main_menu items */
USARTx_Write_Str(USART2, g_main_menu);
/* Enable global interrupt (Note: there are two underscores before the word enable) */
__enable_irq(); // Enable global interrupt
/* Begin infinite loop */
while(1)
{
/* LCD_demo function will run */
LCD_demo();
} /* End of while(1) */
} /* End of main(void) */
/* Define the delay function. This function asks the processor to wait for a specified amount of time before executing the next instruction. In
the while loop, the processor wait until the selected number 'num_tick' counts down to 0. */
void delay(uint32_t num_tick)
{
/* Wait until 'num_tick' counts down to 0 */
while(num_tick--);
} // end of delay(uint32_t num_tick)
```

mcro_declare.h:

```
#ifndef H_MCRO_DECLARE
#define H_MCRO_DECLARE
/* Declare the delay function before calling it */
void delay(uint32_t num_tick);
/* Declare GPIOx_Init function before calling it */
void GPIOx_Init(GPIO_TypeDef* GPIOx,
uint32_t GPIOx_CLK_EN_BIT_MASK,
uint32_t GPIOx_pin_number,
uint32_t GPIOx_mode,
uint32_t periph_AF_number,
uint32_t GPIOx_output_type,
uint32_t GPIOx_output_speed,
uint32_t GPIOx_pupd);
/* Declare GPIOx_EXTI_Init function before calling it */
void GPIOx_EXTI_Init(uint32_t GPIOx_pin_number,
uint32_t EXTICR_EXTIx_mask,
uint32_t EXTICR_EXTIx_PORTx_mask,
uint32_t EXTI_RTSR_TRx_mask,
uint32_t EXTI_FTSR_TRx_mask,
uint32_t EXTI_IMR_MRx_mask,
uint32_t NVIC_EXTIx_IRQn,
uint32_t NVIC_EXTIx_priority);
/* Declare USARTx_Init() function before calling it */
void USARTx_Init(uint32_t USART_num,
USART_TypeDef * USARTx,
uint32_t USARTx_CLK_EN_bit_mask,
uint32_t word_length,
uint32_t num_stop_bits,
uint32_t parity,
uint32_t parity_select,
uint32_t oversampling,
uint32_t baud_rate,
uint32_t dma_tx,
uint32_t dma_rx,
uint32_t interrupt_mask,
uint32_t USARTx_IRQn,
uint32_t priority);
/* Declare USARTx_Write_Str() function before calling it */
void USARTx_Write_Str(USART_TypeDef * USARTx, uint8_t *str);
```

```
uint32_t return_to_main(void);
//void sysclk_testing(void);
void I2C_Start(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL);
void I2C_Stop(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL);
uint32_t I2C_read_ACK(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL);
void I2C_send_data(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL,
uint8_t data);
void I2C_Write(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL,
uint8_t dev_addr,
uint32_t mem_addr,
uint32_t mem_addr_size,
uint8_t data);
uint8_t I2C_Read(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL,
uint8_t dev_addr,
uint32_t mem_addr,
uint32_t mem_addr_size);
void delay_ms_timer(uint32_t num_ms);
void display(unsigned char *text);
void clear_display(void);
void LCD_init(void);
void LCD_mainmenu();
void LCD_menu2();
void LCD_demo(void);
void start_recording();
void save_recording();
void recorded_songs(void);
#endif
```

mcro_define.h:

```
#ifndef H_MCRO_DEFINE
#define H_MCRO_DEFINE
/* Declare the mask for the bit associated with turning on/off GPIOB peripheral clock */
/* Bit[1] of RCC_AHB1ENR register is associated with GPIOC */
#define GPIOB_CLK_EN_BIT_MASK (1 << 1)
/* Define port pins */
#define PB13 (13)
#define PB14 (14)
#endif
```

mcro_include.h:

```
#ifndef H_MCRO_INCLUDE
#define H_MCRO_INCLUDE
#include <__cross_studio_io.h>
#include "stm32f4xx.h"
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include "mcro_define.h"
#include "mcro_declare.h"
extern uint32_t g_usart2_read_count;
```

```
extern uint8_t g_usart2_read_buffer[256];
#endif
```

timer_functions.c:

```
/*-----------------------------------------------------------------------------------------------
FILENAME: timer_functions.c
HEADER: mcro_include.h
DESCRIPTION: This file contains the definitions of all the functions that are used to configure and initialize the Timer in different modes: timer mode, input capture mode, and output compare mode, both with interrupts and without interrupts.
This file also contains Timer interrupt service routine named TIM2_IRQHandler() and TIM3_IRQHandler()
HARDWARE: Olimex prototyping board with STM32F405RG microcontroller.
AUTHOR: Habib Islam
-----------------------------------------------------------------------------------------------*/
/* FUNCTION: void TIMx_Init_Timer(TIM_TypeDef * TIMx, uint32_t APB_number, uint32_t TIMx_CLK_EN_bit_mask, uint32_t TIMx_PSC_val, uint32_t TIMx_ARR_val, uint32_t TIMx_DIER_mask, uint32_t TIMx_IRQn, uint32_t TIMx_interrupt_priority)
DESCRIPTION: Configures and initializes any Timer channel to operate in timer mode only. These timers include TIM2, TIM3, TIM4, TIM5, TIM6, TIM7, TIM12, TIM13, and TIM14. This function can be used to generate a specified amount of delay or create a timer. It does not require to configure any GPIOx pin as a Timer pin.
PARAMETERS: TIMx - Base Address of selected timer is a pointer to TC_TypeDef structure
            APB_number - APB bus number (APB1 or APB2) the selected timer is connected with
            TIMx_CLK_EN_bit_mask - RCC_APB1 bit mask to enable the selected timer
            TIMx_PSC_val - Prescaler value
            TIMx_ARR_val - Auto Reload Register (ARR) value
            TIMx_DIER_mask - Desired interrupt mask for the selected timer
            TIMx_IRQn - IRQ number of the selected timer
            TIMx_interrupt_priority - Desired interrupt priority level for the selected timer
RETURNS: None
-----------------------------------------------------------------------------------------------*/
#include "mcro_include.h"
/* Function prototype */
void TIMx_Init_Timer(TIM_TypeDef * TIMx,
uint32_t APB_number,
uint32_t TIMx_CLK_EN_bit_mask,
uint32_t TIMx_PSC_val,
uint32_t TIMx_ARR_val,
uint32_t TIMx_DIER_mask,
uint32_t TIMx_IRQn,
uint32_t TIMx_interrupt_priority)
{
/* Function definition starts */
/* If the selected timer is connected to APB1 bus */
if(APB_number == APB1)
{
/* Enable TIMx clock by setting the corresponding bit in RCC_APB1ENR register */
RCC->APB1ENR |= TIMx_CLK_EN_bit_mask;
}
/* Else if the selected timer is connected to APB2 bus */
if(APB_number == APB2)
{
/* Enable TIMx clock by setting the corresponding bit in RCC_APB1ENR register */
RCC->APB2ENR |= TIMx_CLK_EN_bit_mask;
}
/* Disable TIMx before configuring its register by clearing CEN (Bit[0]) in TIMx_CR1 register */
TIMx->CR1 &= ~TIM_CR1_CEN;
/* Write the desired prescaler value in TIMx_PSC register */
TIMx->PSC = TIMx_PSC_val;
/* Write the desired ARR value to TIMx_ARR register */
TIMx->ARR = TIMx_ARR_val;
/* Enable the desired interrupts of the selected timer by setting the bits associated with the desired interrupts in TIMx_DIER register */
TIMx->DIER |= TIMx_DIER_mask;
/* Clear the current value of the counter */
TIMx->CNT = 0;
/* Enable the timer channel by setting the CEN (Bit[0]) in TIMx_CR1 register */
TIMx->CR1 |= TIM_CR1_CEN;
/* Set the interrupt priority of the selected timer */
```

```
NVIC_SetPriority(TIMx_IRQn, TIMx_interrupt_priority);
/* Enable the interrupt of the selected timer */
NVIC_EnableIRQ(TIMx_IRQn);
} /* End of TIMx_Init_Timer() */
/*-----------------------------------------------------------------------------------------------------
FUNCTION: void delay_ms_timer(uint32_t num_ms)
DESCRIPTION: This function generates a specified amount of millisecond delay using the timer TIM2. TIM2 is initialized with a prescaler value of
41 and ARR value of 999 resulting in an overflow every 1 ms. The processor checks the the UIF flag and every time this flag is set in 1 ms, it
decrements the variable 'num_ms' until it becomes zero. As a result 'num_ms' millisecond amount of delay is generated.
PARAMETERS: num_ms - amount of millisecond delay to be generated
RETURNS: None
-----------------------------------------------------------------------------------------------------*/
void delay_ms_timer(uint32_t num_ms)
{
/* Initialize and configure TIM2 */
TIMx_Init_Timer(TIM2, /* TIM_TypeDef* TIMx = TIM2. Base address of TIM2 is a pointer to TIM_TypeDef structure */
APB1, /* APB_number = APB1 */
RCC_APB1ENR_TIM2EN, /* TIMx_CLK_EN_bit_mask = 1 << 0. Enable TIM2 clock by setting TIM2 EN (Bit[0]) in RCC_APB1ENR register */
41, /* TIMx_PSC_val = 41. Configure TIM2 counter clock frequency to be 1 MHz by writing 41 to TIM2_PSC register
        TIM2 clock frequency = 42 MHz/(41 + 1) = 1 MHz.
        TIM2 clock duration = 1/1 MHz = 1 micro second */
999, /* TIMx_ARR_val = 999. Configure TIM2 overflow duration to be 1 ms by writing 999 to TIM2_ARR register.
        Overflow duration = 1 micro second * (999 + 1) = 1 ms. */
0, /* TIMx_DIER_mask = 0 (no interrupt enabled) */
0, /* TIMx_IRQn = 0 (doesn't matter since no interrupt is enabled) */
0 /* TIMx_priority = 0 (doesn't matter since no interrupt is enabled) */
);
/* While num_ms > 0 */
while(num_ms)
{
/* If Update Interrupt Flag (UIF) is set */
if(TIM2->SR & TIM_SR_UIF) // Same as if ((TIM2->SR & TIM_SR_UIF_Msk) != 0)
{
/* Clear UIF flag */
TIM2->SR &= ~(TIM_SR_UIF);
/* Decrement num_ms */
num_ms--;
} /* End of if(TIM2->SR & TIM_SR_UIF) */
} /* End of while(num_ms) */
} /* End of delay_ms_timer() */
```

usart_functions.c:

```
/*-----------------------------------------------------------------------------
FILENAME: usart_functions.c
HEADER: mcro_main.h
DESCRIPTION: This file contains the definitions of all the functions that are used for transmitting receiving data using USART.
This file also contains the USART2 interrupt service routine named USART2_IRQHandler()
REFERENCES: STM32F405 Datasheet
AUTHOR: Habib Islam
Modified by Jahziel Ortega
-----------------------------------------------------------------------------*/
#include "mcro_include.h"
/*-----------------------------------------------------------------------------
FUNCTION: void USARTx_Init(uint32_t USART_num, USART_TypeDef* USARTx, uint32_t USARTx_CLK_EN_bit_mask, uint32_t word_length,
uint32_t num_stop_bits, uint32_t parity, uint32_t parity_select, uint32_t oversampling, uint32_t baud_rate, uint32_t dma_tx, uint32_t dma_rx,
uint32_t interrupt_mask, uint32_t USARTx_IRQn, uint32_t priority)
DESCRIPTION: Configures and Initializes any USART using a configurable interrupt

PARAMETERS: USART_num - number index for USART in STM32F405RG, number = 1, 2, 3, .....etc
            USARTx - pointer to the base address of USARTx, x = 1, 2, 3, etc.
            USARTx_CLK_EN_bit_mask - Enables USARTx clock
            word_length - character length
            number_stop_bits - number of stop bits
            parity - parity enable/disable
            parity_select - even/odd parity
```

```
                oversampling - 8x/16x oversampling
                baud_rate - baud rate
                dma_tx - enables/disables DMA transmit
                dma_rx - enables/disables DMA receive
                interrupt_mask - USARTx bit mask for enabling desired interrupt
                USARTx_IRQn - USARTx IRQ number
                priority - Desired interrupt priority for USARTx
RETURNS: None
------------------------------------------------------------------------------*/
/* Function prototype declaration */
void USARTx_Init(uint32_t USART_num,
USART_TypeDef* USARTx,
uint32_t USARTx_CLK_EN_bit_mask,
uint32_t word_length, /* 00 = 8 data bits, 01 = 9 data bits */
uint32_t num_stop_bits, /* 00 = 1 stop bit, 01 = 0.5 stop bit, 10 = 2 stop bits, 11 = 1.5 stop bits */
uint32_t parity, /* 0 = parity disabled, 1 = parity enabled */
uint32_t parity_select, /* 0 = even parity, 1 = odd parity */
uint32_t oversampling, /* 0 = 16x oversampling, 1 = 8x oversampling */
uint32_t baud_rate, /* Baud Rate = f_PCLK/(8*(2-OVER8)*USARTDIV) */
uint32_t dma_tx, /* 0 = TX DMA disabled, 1 = TX DMA enabled */
uint32_t dma_rx, /* 0 = RX DMA disabled, 1 = RX DMA enabled */
uint32_t interrupt_mask, /* TXE = Bit[7], TC = Bit[6], RXNE = Bit[5] */
uint32_t USARTx_IRQn,
uint32_t priority
)
{
/* If USART2 or USART3 is selected, enable its clock by setting the corresponding bit in RCC_APB1ENR register */
if ((USART_num == 2)|(USART_num == 3))
{
RCC->APB1ENR |= USARTx_CLK_EN_bit_mask;
}
/* If USART1 or USART6 is selected, enable its clock by setting the corresponding bit in RCC_APB2ENR register */
else if ((USART_num == 1)|(USART_num == 6))
{
RCC->APB2ENR |= USARTx_CLK_EN_bit_mask;
}
USARTx->CR1 &= ~(1 << 13); // Disable USART
USARTx->CR1 &= ~(3 << 12); // clear M filed
USARTx->CR1 |= word_length << 12; // set the word length
USARTx->CR2 &= ~(3 << 12); // clear STOP field (Bits[13:12])
USARTx->CR2 |= num_stop_bits << 12; // set the number of stop bits
USARTx->CR1 &= ~(1 << 10); // clear PCE field (Bit[10])
USARTx->CR1 |= parity << 10; // enable/disable parity control
USARTx->CR1 &= ~(1 << 9); // clear PS field (Bit[9])
USARTx->CR1 |= parity_select << 9; // select the desired parity
USARTx->CR1 &= ~(1 << 15); // clear OVER8 field (Bit[15])
USARTx->CR1 |= oversampling << 15; // set the desired oversampling
USARTx->BRR |= baud_rate; // set the desired baud rate
USARTx->CR3 &= ~(1 << 7); // cleart DMAT field (Bit[7])
USARTx->CR3 |= dma_tx << 7; // enable/disable DMA for transmit
USARTx->CR3 &= ~(1 << 6); // cleart DMAR field (Bit[6])
USARTx->CR3 |= dma_rx << 6; // enable/disable DMA for receive
USARTx->CR1 |= 3 << 2; // Enable TX and RX
USARTx->CR1 |= 1 << 13; // Enable USART
USARTx->CR1 &= ~interrupt_mask; // clear desired interrupt bit fields
USARTx->CR1 |= interrupt_mask; // enable desired interrupt
NVIC_SetPriority(USARTx_IRQn, priority); // set the priority
NVIC_EnableIRQ(USARTx_IRQn); // Enable IRQ for USARTx
}
/*------------------------------------------------------------------------------
FUNCTION: void USARTx_Write_Str(USART_TypeDef * USARTx, uint8_t * str)
DESCRIPTION: Transmits a string of characters using USART
PARAMETERS: USARTx - Pointer to the base address of USARx
                str - Pointer to the string to be sent
RETURNS: None
AUTHOR: Habib Islam
```

```
-------------------------------------------------------------------------------*/
/* Function prototype declaration */
void USARTx_Write_Str(USART_TypeDef *USARTx, uint8_t *str)
/* Beginning of function definition */
{
uint32_t i = 0; /* Declare and initialize an index for the string elements */
for(i = 0; i < strlen(str); i++)
{
/* Wait until TC (Transmission Complete) bit (Bit[6]) in USART_SR register is set */
while(!(USARTx->SR & USART_SR_TC)); /* while((USARTx->SR & (1 << 6) == 0) */
/* Write a character to be transmitted to the Data Register (USART_DR) */
USARTx->DR = (str[i] & 0xFF);
} /* End of for(i = 0; i < strlen(str); i++) loop */
} /* End of USARTx_Write_Str */
/*-------------------------------------------------------------------------------
FUNCTION: void USART2_IRQHandler(void)
DESCRIPTION: Define ISR for USART2 RXNE interrupt
PARAMETERS: None
RETURNS: None
-------------------------------------------------------------------------------*/
/* Initialize the number of characters received by USARTx_DR */
uint32_t g_usart2_read_count = 0;
/* Initialize the array that would store the characters read from USARTx_DR */
uint8_t g_usart2_read_buffer[256] = {};
void USART2_IRQHandler(void)
{
/* If RXNE (Bit[5]) flag is set in USART2_SR */
if(USART2->SR & (1 << 5))
{
/* Increment the read count */
g_usart2_read_count ++;
/* Read data into read buffer array */
g_usart2_read_buffer[g_usart2_read_count - 1] = USART2->DR;
/* If the read buffer is full */
if(g_usart2_read_count >= 256)
{
/* Clear read buffer array */
memset(g_usart2_read_buffer, 0, 256);
/* Reset the read count */
g_usart2_read_count = 0;
}
}
}
```

EEPROM code:

This code allows the user to record songs and store them in the EEPROM. Stored songs can be read from the EEPROM, and played back or erased. This code was written by utilizing functions written and provided by Habib Islam and Iouri Kourilov during the Micro Fundamentals (MCRO-310) and Micro Design & Application (MCRO-350) courses in the ENT program.

adc_functions.c:

```
/*-----------------------------------------------------------------------------------------------------------------FILENAME: adc_functions.c
DESCRIPTION: This file contains the definitions of all the functions that are used to configure and initialize
an arbitrary ADC with an arbitrary channel of STM32F405xx microcnontrollers.
This file also contains the ADC1 interrupt service routine named ADC1_IRQHandler()
HARDWARE: Olimex prototyping board with STM32F405RG microcontroller.
AUTHOR: Habib Islam
-----------------------------------------------------------------------------------------------------------------/* FUNCTION: void ADCx_Init(ADC_TypeDef*
ADCx,
uint32_t ADCx_CLK_EN_bit_mask,
uint32_t ADC_CCR_mask,
uint32_t ADCx_SQR1_mask,
uint32_t ADCx_SQR2_mask,
uint32_t ADCx_SQR3_mask,
uint32_t ADCx_SMPR1_mask,
uint32_t ADCx_SMPR2_mask,
uint32_t ADCx_CR1_mask,
uint32_t ADCx_CR2_mask,
uint32_t ADCx_interrupt_priority)
DESCRIPTION: Configures and initializes an arbitrary ADC channel.
PARAMETERS: ADCx - Base address of ADCx is a pointer to ADC_Typedef structure
ADCx_CLK_EN_bit_mask - Bit mask in RCC_APB2ENR register to enable the clock of ADCx
ADC_CCR_mask - Bit masks in ADC_CCR register to configure ADC channel prescaler (ADCPRE),
temperature sensor, battery, etc.
ADCx_SQR1_mask - Bit masks in ADCx_SQR1 register to select the regular channel
sequence length and the conversion sequence number (13th to 16th)
of regular channels
ADCx_SQR2_mask - Bit masks in ADCx_SQR2 register to select the conversion
sequence number (7th to 12th) of regular channels
ADCx_SQR3_mask - Bit masks in ADCx_SQR3 register to select the conversion
sequence number (1st to 6th) of regular channels
ADCx_SMPR1_mask - Bit masks in ADCx_SMPR1 register to select the sampling time
for Channel 10 to Channel 18
ADCx_SMPR1_mask - Bit masks in ADCx_SMPR2 register to select the sampling time
for Channel 0 to Channel 9
ADCx_CR1_mask - Bit masks in ADCx_CR1 register to to configure ADC resolution,
End of Conversion interrupt, etc
ADCx_CR2_mask - Bit masks in ADCx_CR2 register to to configure single/continuous conversion,
mode, DMA mode, data alignment, external trigger enable/disable,
external trigger selection, etc
ADCx_interrupt_priority - Desired interrupt priority for ADCx
RETURNS: None
HARDWARE: Olimex prototyping board with STM32F405RG microcontroller.
-----------------------------------------------------------------------------------------------------------------#include "mcro_include.h"
void ADCx_Init(ADC_TypeDef * ADCx,
uint32_t ADCx_CLK_EN_bit_mask,
uint32_t ADC_CCR_mask,
uint32_t ADCx_SQR1_mask,
uint32_t ADCx_SQR2_mask,
uint32_t ADCx_SQR3_mask,
uint32_t ADCx_SMPR1_mask,
uint32_t ADCx_SMPR2_mask,
uint32_t ADCx_CR1_mask,
uint32_t ADCx_CR2_mask,
uint32_t ADCx_interrupt_priority)
```

```
{
/* Enable ADCx clock by setting the corresponding bit in RCC_APB2ENR register */
RCC->APB2ENR |= ADCx_CLK_EN_bit_mask;
/* Disable ADCx before configuring its registers by clearing ADON (Bit[0]) in ADCx_CR2 register */
ADCx->CR2 &= ~ADC_CR2_ADON;
/* Clear ADCPRE (Bits[17:16]), VBATE (Bit[22]) and TSVREFE (Bit[23]) in ADC_CCR register */
ADC->CCR &= ~(ADC_CCR_ADCPRE|ADC_CCR_TSVREFE|ADC_CCR_VBATE);
/* Set ADCPRE, VBATE, and TSVREF to desired configuration in ADC_CCR */
ADC->CCR |= ADC_CCR_mask;
/* Clear L[3:0] (Bits[23:20]) to set the regular channel sequence length to 1 (default) */
ADCx->SQR1 &= ~ADC_SQR1_L;
/* Set the bits related to desired channel sequence length and sequence numbers */
ADCx->SQR1 |= ADCx_SQR1_mask;
/* Clear all bits in ADCx_SQR2 register */
ADCx->SQR2 = 0;
/* Set the bits related to the desired sequence number of the selected channel */
ADCx->SQR2 |= ADCx_SQR2_mask;
/* Clear all bits of ADCx_SQR3 register */
ADCx->SQR3 = 0;
/* Set the bits related to the desired sequence number of the selected channel */
ADCx->SQR3 |= ADCx_SQR3_mask;
/* Clear all bits of ADCx_SMPR1 register */
ADCx->SMPR1 = 0;
/* Set the bits related to the desired sampling time for the selected channel
if the selected channel is 10 to 18 */
ADCx->SMPR1 |= ADCx_SMPR1_mask;
/* Clear all bits of ADCx_SMPR2 */
ADCx->SMPR2 = 0;
/* Set the bits related to the desired sampling time for the selected channel
if the selected channel is 0 to 9 */
ADCx->SMPR2 |= ADCx_SMPR2_mask;
/* Clear SCAN, AWDEN, and RES bits in ADCx_CR1 register */
ADCx->CR1 &= ~(ADC_CR1_SCAN /* Disable scan mode by clearing SCAN (Bit[8]) in ADCx_CR1 register */
|ADC_CR1_AWDEN /* Disable analog watchdog on regular channels by clearing AWDEN (Bit[23]) in ADCx_CR1 |ADC_CR1_RES /* Configure for
12 bit resolution by clearing RES (Bits[25:24]) in ADCx_CR1 register */
);
/* Set the desired values for SCAN, AWDEN, and RES in ADCx_CR1 register */
ADCx->CR1 |= ADCx_CR1_mask;
/* Clear CONT, EOCS, and EXTEN bits in ADCx_CR2 register */
ADCx->CR2 &= ~(ADC_CR2_CONT /* Configure for single conversion by clearing CONT (Bit[1]) in ADCx_CR2 register */
|ADC_CR2_EOCS /* Configure for the EOC bit to be set at the end of each sequence of regular
conversions by clearing EOCS (Bit[10]) in ADCx_CR2 register*/
|ADC_CR2_EXTEN /* Disable external trigger detection by clearing EXTEN (Bits[29:28]) in ADCx_CR2 register );
/* Set the desired values for CONT, EOCS, and EXTEN bits */
ADCx->CR2 |= ADCx_CR2_mask;
/* Enable ADCx by setting ADON (Bit[0]) in ADCx_CR2 register */
ADCx->CR2 |= ADC_CR2_ADON;
/* Configure ADCx for the desired interrupt priority */
NVIC_SetPriority(ADC_IRQn, ADCx_interrupt_priority);
/* Enable interrupt for ADCx */
NVIC_EnableIRQ(ADC_IRQn);
}
/*-----------------------------------------------------------------------------------------------
FUNCTION: void ADC_IRQHandler(void)
FILE NAME: adc_functions.c
DESCRIPTION: Checks the status of End of Conversion (EOC) flag in ADC1_SR register.
If the EOC flag is set, it reads the converted data from ADC1_DR
and assign it to the external global variable g_adc_read. Every time
a new converted data is read by the IRQ Handler, the flag
g_new_adc_read is set.
PARAMETERS: None
RETURNS: None
----------------------------------------------------------------------------------------------- */
/* Declare and initialize the global variable g_adc_read that would hold the ADC converted data. */
uint16_t g_adc_read = 0;
/* Declare and initialize the global array g_adc_results[2] that would hold the ADC converted data from two channels. */
```

```
uint16_t g_adc_results[12] = {0};
/* Declare and initialize ADC modes of operation. g_adc_mode = 0 (single channel conversion),
g_adc_mode = 1 (multi-channel conversion) */
uint32_t g_adc_mode = 0;
/* Declare and initialize the global variable g_new_adc_read */
uint32_t g_new_adc_read = 0;
/* Declare and initialize index for multiple ADC channels */
uint32_t n = 0;
void ADC_IRQHandler(void)
{
/* If ADC is in single channel conversion mode */
if(g_adc_mode == 0)
{
/* If EOC flag is set */
if(ADC1->SR & ADC_SR_EOC)
{
/* Read the converted the data into g_adc_read */
g_adc_read = ADC1->DR;
/* Set the flag for new read data */
g_new_adc_read = 1;
}
}
/* If ADC is in multi-channel conversion mode */
else if(g_adc_mode == 1)
{
/* If EOC flag is set */
if(ADC1->SR & ADC_SR_EOC)
{
/* Read the data converted from channel n into g_adc_results[n] */
g_adc_results[n] = ADC1->DR;
/* Increment the channel index */
n++;
/* Set channel index to 0 if n is an integer multiple of 12 */
n %= 12;
/* Set the flag for new ADC date */
g_new_adc_read = 1;
} /* End of else if(g_adc_mode == 1) */
} /* End of while(1) */
} /* End of ADC_IRQHandler() */
```

dac_functions.c:

```
/*-----------------------------------------------------------------------------------------------------------------FILENAME: dac_functions.c
HEADER FILE: mcro_include.h
DESCRIPTION: This file contains the definitions of all the functions that are used to configure and initialize
the DAC controller (DAC)
HARDWARE: Olimex prototyping board with STM32F405RG microcontroller.
AUTHOR: Habib Islam
-----------------------------------------------------------------------------------------------------------------#include "mcro_include.h"
/* -----------------------------------------------------------------------------------------------------------------/* FUNCTION: void
DACC_Init(DAC_CH_number, DAC_CR_mask, DAC_SWTRIGR_mask)
DESCRIPTION: Configures and initializes an arbitrary DAC channel.
PARAMETERS: DAC_CH_number - Selected DAC channel number
DAC_CR_mask - desired values in DAC_CR register
DAC_SWTRIGR_mask - desired value in DAC_SWTRIGR register
RETURNS: None
HARDWARE: Olimex prototyping board with STM32F405RG microcontroller.
-----------------------------------------------------------------------------------------------------------------void DAC_Init(uint32_t DAC_CH_number,
uint32_t DAC_CR_mask,
uint32_t DAC_SWTRIGR_mask)
{
/* Enable DAC clock */
RCC->APB1ENR |= RCC_APB1ENR_DACEN;
/* If DAC channel number is 1 */
if (DAC_CH_number == 1)
{
```

```
/* Clear EN1, TEN1, DMAEN1, TSEL1 and WAVE 1 bits in DAC_CR */
DAC->CR &= ~(DAC_CR_EN1 | DAC_CR_TEN1 | DAC_CR_DMAEN1 | DAC_CR_TSEL1 | DAC_CR_WAVE1 | DAC_CR_MAMP1);
/* Set the desired values for TEN1, DMAEN1, TSEL1, and WAVE1 */
DAC->CR |= DAC_CR_mask;
/* Clear SWTRIGR1 (Bit[0]) in DAC_SWTRIGR register */
DAC->SWTRIGR &= ~DAC_SWTRIGR_SWTRIG1;
/* Set the desired value for SWTRIGR1 */
DAC->SWTRIGR |= DAC_SWTRIGR_mask;
/* Clear data register for channel 1 */
DAC->DHR12R1 = 0;
/* Enable DAC Channel 1 */
DAC->CR |= DAC_CR_EN1;
}
/* If DAC channel number is 2 */
else if (DAC_CH_number == 2)
{
/* Clear EN2, TEN2, DMAEN2, TSEL2 and WAVE2 bits in DAC_CR */
DAC->CR &= ~(DAC_CR_EN2 | DAC_CR_TEN2 | DAC_CR_DMAEN2 | DAC_CR_TSEL2 | DAC_CR_WAVE2 | DAC_CR_MAMP2);
/* Set the desired values for TEN2, DMAEN2, TSEL2, and WAVE2 */
DAC->CR |= DAC_CR_mask;
/* Clear SWTRIGR2 (Bit[1]) in DAC_SWTRIGR register */
DAC->SWTRIGR &= ~DAC_SWTRIGR_SWTRIG2;
/* Set the desired value for SWTRIGR2 */
DAC->SWTRIGR |= DAC_SWTRIGR_mask;
/* Clear data register for channel 2 */
DAC->DHR12R2 = 0;
/* Enable DAC Channel 2 */
DAC->CR |= DAC_CR_EN2;
}
}
```

gpio_functions.c:

```
/*-----------------------------------------------------------------------------
FILENAME: gpio_functions.c
HEADER: __cross_studio_io.h, stm32f4xx.h
DESCRIPTION: This file contains the definitions of all the functions that are used to control
input and output pins by writing to or reading from GPIOx registers.
REFERENCES: STM32F405 Datasheet
AUTHOR: Habib Islam
-----------------------------------------------------------------------------*/
#include "mcro_include.h"
/*-------------------------------------------------------------------------------------------
FUNCTION: void GPIOx_Init(GPIO_TypeDef* GPIOx,
uint32_t GPIOx_CLK_EN_bit_mask,
uint32_t GPIOx_pin_number,
uint32_t GPIOx_mode,
uint32_t periph_AF_number,
uint32_t GPIOx_output_type,
uint32_t GPIOx_output_speed,
uint32_t GPIOx_pupd)
DESCRIPTION: This function configures GPIOx (x = A, B, C, D, etc.) pins in an arbitrary GPIO mode
so that digital I/O pins can be used by any arbitrary STM32F405 peripheral.
PARAMETERS: GPIOx - Pointer to the base address of GPIOC
GPIOx_pin_number - GPIOx pin number
GPIOx_mode - Selected Mode (e.g. Input, Output, Analog, Alternate Function)
periph_AF_number - AF number associated with the selected peripheral
GPIOx_output_type - GPIOx output type (e.g. push-pull, open-drain)
GPIOx_output_speed - GPIOx output speed (e.g. low, medium, high)
GPIOx_pupd - GPIOx pull-up/pull-down mode
RETURNS: None
AUTHOR: Habib Islam
-------------------------------------------------------------------------------------------*/
void GPIOx_Init(GPIO_TypeDef* GPIOx,
uint32_t GPIOx_CLK_EN_bit_mask,
uint32_t GPIOx_pin_number,
```

```
uint32_t GPIOx_mode,
uint32_t periph_AF_number,
uint32_t GPIOx_output_type,
uint32_t GPIOx_output_speed,
uint32_t GPIOx_pupd)
{
/* Enable GPIOx clock by setting corresponding EN bit in RCC_AHB1ENR register */
RCC->AHB1ENR |= GPIOx_CLK_EN_bit_mask;
/* Clear mode of the selected pin by clearing corresponding bit in MODER register */
GPIOx->MODER &= ~(3 << 2*GPIOx_pin_number);
/* Set the desired mode for the selected GPIOx pin */
GPIOx->MODER |= GPIOx_mode << 2*GPIOx_pin_number;
/* If GPIOx mode is Alternate Function, configure GPIOx_AFRL (AFR[0])
and GPIOx_AFRH (AFR[1]) registers for the GPIOx pin */
if (GPIOx_mode == 2)
{
/* If the GPIOx pin number is 0 to 7 */
if ((GPIOx_pin_number >= 0) && (GPIOx_pin_number <= 7))
{
/* Clear the 4 bits in AFRL register associated with the GPIOx pin */
GPIOx->AFR[0] &= ~(0xF << 4*GPIOx_pin_number);
/* Write the AF number in corresponding 4 bits in GPIOx_AFRL register */
GPIOx->AFR[0] |= periph_AF_number << 4*GPIOx_pin_number;
}
/* If the GPIOx pin number is 8 to 16*/
else if ((GPIOx_pin_number >= 8) && (GPIOx_pin_number <= 16))
{
/* Clear the 4 bits in AFRL register associated with the GPIOx pin */
GPIOx->AFR[1] &= ~(0xF << (4*GPIOx_pin_number - 32));
/* Write the AF number in corresponding 4 bits in GPIOx_AFRL register */
GPIOx->AFR[1] |= periph_AF_number << (4*GPIOx_pin_number - 32);
}
}
/* Clear output type by clearing the bit associated with GPIOx pin in GPIOx_OTYPER register */
GPIOx->OTYPER &= ~(1 << GPIOx_pin_number);
/* Select the desired output type by setting the corresponding bit in GPIOx_OTYPER register */
GPIOx->OTYPER |= GPIOx_output_type << GPIOx_pin_number;
/* Clear output speed by clearing the two bits associated with GPIOx pin in GPIOx_OSPEEDR register */
GPIOx->OSPEEDR &= ~(3 << 2*GPIOx_pin_number);
/* Select the desired output speed by writing the desired speed to the corresponding bits in GPIOx_OTYPER register */
GPIOx->OSPEEDR |= GPIOx_output_speed << 2*GPIOx_pin_number;
/* Clear pull-up/pull-down by clearing the two bits associated with GPIOx pin in GPIOx_PUPDR register */
GPIOx->PUPDR &= ~(3 << 2*GPIOx_pin_number);
/* Select the desired pull-up/pull-down type by writing the desired type to the corresponding bits in GPIOx_OTYPER register GPIOx->PUPDR |=
GPIOx_pupd << 2*GPIOx_pin_number;
} /* End of GPIOx_Init */
/*-----------------------------------------------------------------------------------------------
FUNCTION: void GPIOx_EXTI_Init(uint32_t GPIOx_pin_number,
uint32_t EXTICR_EXTIx_mask,
uint32_t EXTICR_EXTIx_PORTx_mask,
uint32_t EXTI_RTSR_TRx_mask,
uint32_t EXTI_FTSR_TRx_mask,
uint32_t EXTI_IMR_MRx_mask,
uint32_t NVIC_EXTIx_IRQn,
uint32_t NVIC_EXTIx_priority)
DESCRIPTION: This function configures an arbitrary GPIOx (x = A, B, C, D, etc.) pin in
input interrupt mode.
PARAMETERS: GPIOx_pin_number - GPIOx pin/line number (0, 1, 2,....,15)
EXTICR_EXTIx_mask - EXTIx bit mask in SYSCFG_EXTICR register (EXTI0_mask 15 << 0, EXTI1_mask 15 << EXTICR_EXTIx_PORTx_mask - EXTIx port
mask in SYSCFG_EXTICR register (PAx = 0, PBx = 1, PCx = 3, ......)
EXTI_RTSR_TRx_mask - TRx bit mask in EXTI_RTSR register (TR0: 1 << 0, TR1: 1 << 1, TR2: 1 << 3, ......)
EXTI_FTSR_TRx_mask - TRx bit mask in EXTI_FTSR register (TR0: 1 << 0, TR1: 1 << 1, TR2: 1 << 3, ......)
EXTI_IMR_MRx_mask - MRx bit mask in EXTI_IMR register (MR0: 1 << 0, MR1: 1 << 1, MR2: 1 << 3, ......)
NVIC_EXTIx_IRQn - IRQ number for EXTIx line
NVIC_EXTIx_priority - Interrupt priority for EXTIx line
RETURNS: None
```

```
AUTHOR: Habib Islam
-----------------------------------------------------------------------------------------------*/
void GPIOx_EXTI_Init(uint32_t GPIOx_pin_number,
uint32_t EXTICR_EXTIx_mask,
uint32_t EXTICR_EXTIx_PORTx_mask,
uint32_t EXTI_RTSR_TRx_mask,
uint32_t EXTI_FTSR_TRx_mask,
uint32_t EXTI_IMR_MRx_mask,
uint32_t NVIC_EXTIx_IRQn,
uint32_t NVIC_EXTIx_priority)
{
/* Enable system configuration clock by setting Bit[14] of RCC_APB2ENR */
RCC->APB2ENR |= RCC_APB2ENR_SYSCFGEN_Msk;
/* For GPIO pins 0 t0 3, use SYSCFG_EXTICR1 register */
if((GPIOx_pin_number >= 0) && (GPIOx_pin_number <= 3))
{
/* Clear EXTIx bits of EXTICR1 register */
SYSCFG->EXTICR[0] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[0] |= EXTICR_EXTIx_PORTx_mask;
}
/* For GPIO pins 4 t0 7, use SYSCFG_EXTICR2 register */
else if((GPIOx_pin_number >= 4) && (GPIOx_pin_number <= 7))
{
/* Clear EXTIx bits of EXTICR2 register */
SYSCFG->EXTICR[1] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[1] |= EXTICR_EXTIx_PORTx_mask;
}
/* For GPIO pins 8 t0 11, use SYSCFG_EXTICR3 register */
else if((GPIOx_pin_number >= 8) && (GPIOx_pin_number <= 11))
{
/* Clear EXTIx bits of EXTICR3 register */
SYSCFG->EXTICR[2] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[2] |= EXTICR_EXTIx_PORTx_mask;
}
/* For GPIO pins 8 t0 11, use SYSCFG_EXTICR4 register */
else if((GPIOx_pin_number >= 12) && (GPIOx_pin_number <= 15))
{
/* Clear EXTIx bits of EXTICR4 register */
SYSCFG->EXTICR[3] &= ~EXTICR_EXTIx_mask;
/* Set EXTIx bits for corresponding port */
SYSCFG->EXTICR[3] |= EXTICR_EXTIx_PORTx_mask;
}
/* Enable/disable interrupt trigger for rising edge by setting/clearing the
corresponding TRx bit in EXTI_RTSR register*/
EXTI->RTSR |= EXTI_RTSR_TRx_mask;
/* Enable/disable interrupt trigger for falling edge by setting/clearing the
corresponding TRx bit in EXTI_FTSR register*/
EXTI->FTSR |= EXTI_FTSR_TRx_mask;
/* Unmask the interrupt request from line x by setting
the MRx bit in EXTI_IMR register */
EXTI->IMR |= EXTI_IMR_MRx_mask;
/* Set the priority for the external interrupt */
NVIC_SetPriority(NVIC_EXTIx_IRQn, NVIC_EXTIx_priority);
/* Enable NVIC level interrupt for EXTIx */
NVIC_EnableIRQ(NVIC_EXTIx_IRQn);
} /* End of GPIOx_Read_interrupt() */
/*-----------------------------------------------------------------------------------------------
FUNCTION: void EXTI0_IRQHandler(void)
DESCRIPTION: Define external interrupt handler for GPIO line 0 (EXTI0). Each time the selected edge
arrives on the external interrupt line 0, a global variable count
g_exti0_count is incremented. This value of g_exti0_count can be used
to conditionally run any application.
PARAMETERS: None
```

```
RETURNS: None
AUTHOR: Habib Islam
-----------------------------------------------------------------------------------------------------------*/
uint32_t g_exti0_count = 0;
void EXTI0_IRQHandler(void)
{
/* If the pending bit PR0 in EXTI_PR register is set */
if ((EXTI->PR & (1 << 0)) == (1 << 0)) // if ((EXTI->PR & EXTI_PR_PR0) == EXTI_PR_PR0)
{
/* Increment the interrupt event count */
g_exti0_count++;
/* Clear the pending bit by setting the PR0 bit */
EXTI->PR |= 1 << 0; // EXTI->PR |= EXTI_PR_PR0;
} /* End of if ((EXTI->PR & (1 << 0)) == (1 << 0)) */
} /* End of EXTI0_IRQHandler() */
/* -----------------------------------------------------------------------------------------------------------*/
```

harp_debug_functions.c:

```
#include "mcro_include.h"
/*----------Functions used to test features of the Laser Harp:----------*/
void DAC_Test(void) // Test the Signal Generator by manipulating the DAC Outputs manually using USART2
{
/* Message for user prompt */
uint8_t msg[] = "\n\n\tConnect UEXT pin 10 to oscilloscope and observe the waveform!!!\n\r";
/* Write the message to USART2_DR to be displayed on TeraTerm */
USARTx_Write_Str(USART2, msg);
/* Declare and initialize a table of Sawtooth Wave samples.
Note: Since the maximum resolution of DAC is 12-bit, each sample is defined as a 16-bit integer */
uint32_t frequency = 0;
uint32_t volume = 0;
/* Initialize and configure DAC channel 1 */
DAC_Init(1, 0, 0);
DAC_Init(2, 0, 0);
uint8_t display[MAX_STR_LEN] = {0};
uint8_t kb_input;
uint32_t previous_usart2_read_count;
g_usart2_read_count = 0;
sprintf(display, "\r\tFrequency = %d, Volume = %d\r\t", frequency, volume);
USARTx_Write_Str(USART2, display);
while(1)
{
/* If the user hits Esc on keyboard */
if(return_to_main())
{
/* Disable DAC channel 1 by clearing Bit[0] in DAC_CR register */
DAC->CR &= ~DAC_CR_EN1;
/* Disable DAC clock by clearing DAC EN (Bit[29]) in RCC_APB1ENR register */
RCC->APB1ENR &= ~RCC_APB1ENR_DACEN;
/* Return to main menu */
return;
}
if(g_usart2_read_count == 0) continue;
if(previous_usart2_read_count == g_usart2_read_count) continue;
previous_usart2_read_count = g_usart2_read_count;
kb_input = g_usart2_read_buffer[g_usart2_read_count-1];
if((kb_input == 'w') && (frequency >= 0)) frequency -= 1;
else if((kb_input == 'e') && (frequency < 0xFFF)) frequency += 1;
else if((kb_input == 'q') && (frequency >= 10)) frequency -= 10;
else if((kb_input == 'r') && (frequency < 0xFF5)) frequency += 10;
else if((kb_input == 'i') && (volume >= 0)) volume -= 1;
else if((kb_input == 'o') && (volume < 0xFFF)) volume += 1;
else if((kb_input == 'u') && (volume >= 10)) volume -= 10;
else if((kb_input == 'p') && (volume < 0xFF5)) volume += 10;
sprintf(display, "\r\tFrequency = %d, Volume = %d\r\t", frequency, volume);
USARTx_Write_Str(USART2, display);
```

```
DAC->DHR12R1 = frequency;
DAC->DHR12R2 = volume;
} /* End of while(1) */
}
void DAC_Song(void) // Play a song
{
/* Message for user prompt */
uint8_t msg[] = "\n\n\tConnect UEXT pin 10 to oscilloscope and observe the waveform!!!\n\r";
/* Write the message to USART2_DR to be displayed on TeraTerm */
USARTx_Write_Str(USART2, msg);
uint32_t n_len = 250;
uint32_t w_len = 100;
/* Initialize and configure DAC channels*/
DAC_Init(1, 0, 0);
DAC_Init(2, 0, 0);
uint8_t display[MAX_STR_LEN] = {0};
uint8_t kb_input;
uint32_t previous_usart2_read_count;
g_usart2_read_count = 0;
sprintf(display, "\r\tPlaying Music.\r\t");
USARTx_Write_Str(USART2, display);
while(1)
{
/* If the user hits Esc on keyboard */
if(return_to_main())
{
/* Disable DAC channel 1 by clearing Bit[0] in DAC_CR register */
DAC->CR &= ~DAC_CR_EN1;
/* Disable DAC clock by clearing DAC EN (Bit[29]) in RCC_APB1ENR register */
RCC->APB1ENR &= ~RCC_APB1ENR_DACEN;
/* Return to main menu */
return;
}
/*
PlayNote('C',n_len,w_len);
PlayNote('C',n_len,w_len);
PlayNote('G',n_len,w_len);
PlayNote('G',n_len,w_len);
PlayNote('A',n_len,w_len);
PlayNote('A',n_len,w_len);
PlayNote('G',2*n_len,2*w_len);
PlayNote('F',n_len,w_len);
PlayNote('F',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('C',2*n_len,2*w_len);
PlayNote('G',n_len,w_len);
PlayNote('G',n_len,w_len);
PlayNote('F',n_len,w_len);
PlayNote('F',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('D',2*n_len,2*w_len);
PlayNote('G',n_len,w_len);
PlayNote('G',n_len,w_len);
PlayNote('F',n_len,w_len);
PlayNote('F',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('D',2*n_len,2*w_len);
PlayNote('C',n_len,w_len);
PlayNote('C',n_len,w_len);
PlayNote('G',n_len,w_len);
PlayNote('G',n_len,w_len);
```

```
PlayNote('A',n_len,w_len);
PlayNote('A',n_len,w_len);
PlayNote('G',2*n_len,2*w_len);
PlayNote('F',n_len,w_len);
PlayNote('F',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('C',2*n_len,10*w_len);
*/
/*Mary had a little lamb */
/*
PlayNote('E',n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('C',n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('E',2*n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('D',2*n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('E',2*n_len,2*w_len);
PlayNote('E',n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('C',n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('E',2*n_len,2*w_len);
PlayNote('D',n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('E',n_len,w_len);
PlayNote('D',n_len,w_len);
PlayNote('C',3*n_len,w_len);
PlayNote('C',n_len,4*w_len);
*/
}
}
void Single_Laser_Tone(void) // Test a single LED and Photoresistor setup
{
/* Message for user prompt */
uint8_t msg[] = "\r\n\n\tApply a 1-KHz sine wave with amplitude 3.3 V\
\n\r\tand DC offset of 1.65 V to PA1 and observe the TeraTerm\n\r\n";
/* Display message on teraterm */
USARTx_Write_Str(USART2, msg);
/* Declare and initialize the char array to display data */
uint8_t display[MAX_STR_LEN] = {0};
/* Define the display format */
uint8_t display_format[] = "\r\tADC Input = %5.3f V, ADC Output = %4d (dec), ADC Output = 0x%04X (hex)\r\t";
/* Declare and initialize a float variable to store ADC input voltage */
float adc_in = 0;
/* Declare and initialize an integer variable to hold ADC output */
uint32_t adc_out = 0;
/* Define an integer to update the current output as past output */
uint32_t prev_adc_out;
DAC_Init(1, 0, 0);
DAC_Init(2, 0, 0);
/* Configure PA1 as ADC1 pin */
GPIOx_Init(GPIOA, /* Base address of GPIOA is a pointer to GPIO_TypeDef structure */
RCC_AHB1ENR_GPIOAEN, /* Enable GPIOA clock by setting Bit[0] in RCC_AHB1ENR register */
PA1, /* Select GPIOA pin to be PA1 */
```

```
GPIO_MODE_ANALOG, /* Configure PA1 in Analog mode by writing 0b11 to Bits[3:2] in GPIOA_MODER register 0, /* Ignore AF number by
writing 0 since PA1 is configured as Analog */
GPIO_OTYPE_PUSH_PULL, /* Configure PA1 output type to be PUSH-PULL by clearing Bit[1] in GPIOA_OTYPER register GPIO_OSPEED_HIGH, /*
Configure PA1 output speed to be HIGH by writing 0b10 to Bits[3:2] in GPIOA_OSPEEDR GPIO_PUPD_NO_PUPD /* Configure PA1 to be NO
PULL-UP PULL-DOWN by clearing Bits[3:2] in GPIOA_PUPDR register );
/* Configure and initialize ADC1 */
ADCx_Init(ADC1, /* Based address of ADC1 is a pointer to ADC_TypeDef structure */
RCC_APB2ENR_ADC1EN, /* Enable ADC1 clock by setting Bit[8] in RCC_APB2ENR register */
ADC_CCR_ADCPRE, /* Configure ADC1 prescaler to be 8 by writing 0b11 to Bits[17:16] in ADC_CCR register 0, /* Configure for total number of
conversions to be 1 by clearing all bits in ADC_SQR1 0, /* Ignore configuring ADC_SQR2 by writing 0 to ADC_SQR2 register
since the number of conversions is 1 */
ADC_SQR3_SQ1_0, /* Configure for 1st conversion in regular sequence on channel 1 by
writing 0b00001 to SQ1 (Bits[4:0]) in ADC_SQR3 register */
0, /* Ignore configuring ADC_SMPR1 register by writing 0 since this register is used for
configuring sampling time for ADC channel 10 to ADC channel 18 */
0, /* Configure ADC1 sampling time to be 3 ADC1 clock cycles by writing 0b000 to
Bits[5:3] in ADC_SMPR2 register */
0, /* Keep SCAN, AWDEN, and RES bits in ADCx_CR1 register in default mode */
0, /* Keep CONT, EOCS, and EXTEN bits in ADCx_CR2 register in default mode */
0 /* Priority doesn't matter as interrupt is not enabled */
);
int32_t volume = 0;
while(1)
{
/* If the user hits Esc key */
if(return_to_main())
{
/* Disable ADC1 by clearing ADON (Bit[0]) in ADC1_CR2 register */
ADC1->CR2 &= ~ADC_CR2_ADON;
/* Disable ADC clock by clearing ADC1EN (Bit[8]) in RCC_APB2ENR register */
RCC->APB2ENR &= ~RCC_APB2ENR_ADC1EN;
/* Return to main menu */
return;
}
/* Make ADC1 starting conversion of analog data by setting SWSTART (Bit[30]) in ADC1_CR2 register */
ADC1->CR2 |= ADC_CR2_SWSTART;
/* Wait until EOC flag is set */
while(!(ADC1->SR & ADC_SR_EOC));
/* Read the converted data into adc_out */
adc_out = ADC1->DR;
/* Calculate the input voltage to ADC1 */
adc_in = (float)(ADC_VREF/ADC_RES_12 * adc_out);
/* If the current converted data is equal to the previous converted data, ignore the instructions followed by */
//if(prev_adc_out == adc_out) continue;
/* If the current converted data is not equal to the previous converted data,
make the current converted data as previous converted data */
prev_adc_out = adc_out;
/* If the input voltage is more than 2.5 V, turn on board LED */
if(adc_in < 0.5)
{
volume = 0xEFF;
PlayNote(1);
DAC->DHR12R2 = volume;
}
/* If the input voltage is not more than 2.5 V, turn off board LED */
else
{
if(volume>0xCF0)volume -= 50;
else volume = 0;
DAC->DHR12R2 = volume;
}
/* Convert the data and texts to be displayed into string array */
sprintf(display, display_format, adc_in, adc_out, adc_out);
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
}
```

```
}
```

harp_functions.c:

```
#include "mcro_include.h"
uint32_t g_volume = 0;
uint32_t g_IsRecording = 0;
/* Play notes using the keyboard */
void Keyboard_Piano(void)
{
/* Message for user prompt */
uint8_t msg[] = "\n\n\tPress q,w,e,r,t,y,u,2,3,5,6,7 to play notes.\n\r\
\tPress x to toggle recording.\n\r\
\tPress c to playback recorded songs.\n\r\
\tPress v to write songs to the EEPROM.\n\r\
\tPress b to read songs from the EEPROM.\n\r\
\tPress n to play a song by number.\n\r\
\tPress m to erase a song by number.\n\r";
/* Write the message to USART2_DR to be displayed on TeraTerm */
USARTx_Write_Str(USART2, msg);
/* Initialize Timer */
TIMx_Init_Timer(TIM2,
1,
RCC_APB1ENR_TIM2EN,
41,
999,
TIM_DIER_UIE,
TIM2_IRQn,
0
);
/* Declare and initialize a table of Sawtooth Wave samples.
Note: Since the maximum resolution of DAC is 12-bit, each sample is defined as a 16-bit integer */
uint32_t n_len = 250;
/* Initialize and configure DAC channel 1 */
DAC_Init(1, 0, 0);
DAC_Init(2, 0, 0);
uint8_t display[MAX_STR_LEN] = {0};
uint8_t kb_input;
uint32_t previous_usart2_read_count;
//uint32_t num_seconds = 0;
//num_seconds = (uint32_t)g_tim2_overflow_count/1000;
g_usart2_read_count = 0;
g_volume = 0;
g_address = 0;
uint32_t note = 0;
uint32_t prev_note = 0;
uint32_t wait = 0;
memset(g_songs, 0, sizeof(g_songs));
while(1)
{
/* If the user hits Esc on keyboard */
if(return_to_main())
{
/* Disable TIM2 */
TIM2->CR1 &= ~TIM_CR1_CEN;
/* Disable TIM2 clock */
RCC->APB1ENR &= ~RCC_APB1ENR_TIM12EN;
/* Clear overflow count */
g_tim2_overflow_count = 0;
/* Disable DAC channel 1 by clearing Bit[0] in DAC_CR register */
DAC->CR &= ~DAC_CR_EN1;
/* Disable DAC channel 2 by clearing Bit[16] in DAC_CR register */
DAC->CR &= ~DAC_CR_EN2;
/* Disable DAC clock by clearing DAC EN (Bit[29]) in RCC_APB1ENR register */
RCC->APB1ENR &= ~RCC_APB1ENR_DACEN;
/* Return to main menu */
```

```
return;
}
if(g_usart2_read_count == 0) continue;
if(previous_usart2_read_count == g_usart2_read_count) continue;
previous_usart2_read_count = g_usart2_read_count;
kb_input = g_usart2_read_buffer[g_usart2_read_count-1];
switch(kb_input)
{
case 'q': note = 1; break;
case 'w': note = 2; break;
case 'e': note = 3; break;
case 'r': note = 4; break;
case 't': note = 5; break;
case 'y': note = 6; break;
case 'u': note = 7; break;
case '2': note = 8; break;
case '3': note = 9; break;
case '5': note = 10; break;
case '6': note = 11; break;
case '7': note = 12; break;
case 'x':
if(g_IsRecording)
{
g_IsRecording = 0;
EndSong();
USARTx_Write_Str(USART2, "\n\rFinished Recording.\n\r");
USARTx_Write_Str(USART2, msg);
}
else
{
g_IsRecording = 1;
g_tim2_overflow_count = 0;
BeginSong();
USARTx_Write_Str(USART2, "\n\rRecording.\n\r");
}
break;
case 'c':
USARTx_Write_Str(USART2, "\n\rPlaying recorded songs.\n\r");
Playback_Songs();
USARTx_Write_Str(USART2, msg);
break;
case 'v':
USARTx_Write_Str(USART2, "\n\rWriting songs to EEPROM.\n\r");
Write_Notes();
USARTx_Write_Str(USART2, msg);
break;
case 'b':
USARTx_Write_Str(USART2, "\n\rReading songs from EEPROM.\n\r");
Read_Notes();
USARTx_Write_Str(USART2, msg);
break;
case 'n':
USARTx_Write_Str(USART2, "\n\rSelect a song to play:\n\r");
while(1)
{
if(g_usart2_read_count == 0) continue;
if(previous_usart2_read_count == g_usart2_read_count) continue;
previous_usart2_read_count = g_usart2_read_count;
kb_input = g_usart2_read_buffer[g_usart2_read_count-1];
uint32_t song = kb_input - 0x30;
if((song <= 0) || (song > g_number_of_songs))
{
USARTx_Write_Str(USART2, "\n\rInvalid choice\n\r");
USARTx_Write_Str(USART2, msg);
break;
}
```

```
else
{
uint8_t selected_song_display[] = "\n\rPlaying song %d.\n\r\t";
sprintf(display, selected_song_display, song);
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
Playback_Song(song-1);
USARTx_Write_Str(USART2, "\n\rSong Finished.\n\r");
USARTx_Write_Str(USART2, msg);
break;
}
}
break;
case 'm':
USARTx_Write_Str(USART2, "\n\rSelect a song to erase:\n\r");
while(1)
{
if(g_usart2_read_count == 0) continue;
if(previous_usart2_read_count == g_usart2_read_count) continue;
previous_usart2_read_count = g_usart2_read_count;
kb_input = g_usart2_read_buffer[g_usart2_read_count-1];
uint32_t song = kb_input - 0x30;
if((song <= 0) || (song > g_number_of_songs))
{
USARTx_Write_Str(USART2, "\n\rInvalid choice\n\r");
USARTx_Write_Str(USART2, msg);
break;
}
else
{
Erase_Song(song-1);
uint8_t erased_song_display[] = "\n\rSong %d Erased.\n\r\t";
sprintf(display, erased_song_display, song);
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
USARTx_Write_Str(USART2, msg);
break;
}
}
break;
}
if(note)
{
PlayNote(note);
wait = g_tim2_overflow_count;
g_tim2_overflow_count = 0;
if(g_IsRecording)
{
Save_Note(note, g_tim2_overflow_count, wait);
g_tim2_overflow_count = 0;
}
}
else
{
if(g_volume>0xCF0)g_volume -= 50;
else g_volume = 0;
DAC->DHR12R2 = g_volume;
}
note = 0;
}
}
/* Play notes using the MCU's ADCs (USART2 cannot be used to send/receive data) */
void Harp_using_ADC(void)
{
/* Initialize Timer */
TIMx_Init_Timer(TIM2,
```

```
1,
RCC_APB1ENR_TIM2EN,
41,
999,
TIM_DIER_UIE,
TIM2_IRQn,
0
);
ADCx_Init(ADC1, /* Based address of ADC1 is a pointer to ADC_TypeDef structure */
RCC_APB2ENR_ADC1EN, /* Enable ADC1 clock by setting Bit[8] in RCC_APB2ENR register */
ADC_CCR_ADCPRE, /* Configure ADC1 prescaler to be 8 by writing 0b11 to Bits[17:16] in ADC_CCR register 11 << 20, /* Configure for total
number of conversions to be 12 by writing 0b1011 to Bits[24:20] in 10 << 0 | 11 << 5| 12 << 10 | 13 << 15 | 14 << 20 | 15 << 25, /* Configure
channel conversion sequence */
1 << 0 | 2 << 5 | 6 << 10 | 7 << 15 | 8 << 20 | 9 << 25, /* Configure channel conversion sequence */
ADC_SMPR1_SMP10|ADC_SMPR1_SMP11|ADC_SMPR1_SMP12|ADC_SMPR1_SMP13|ADC_SMPR1_SMP14|ADC_SMPR1_SMP15, /*
Configure ADC_SMPR2_SMP1|ADC_SMPR2_SMP2|ADC_SMPR2_SMP6|ADC_SMPR2_SMP7|ADC_SMPR2_SMP8|ADC_SMPR2_SMP9, /*
Configure ADC_CR1_EOCIE | /* Enable End of Conversion (EOC) interrupt by setting EOCIE (Bit[5]) in ADC_CR1 register ADC_CR1_SCAN, /*
Enable SCAN mode by setting SCAN (Bit[8]) in ADC_CR1 register */
ADC_CR2_EOCS | /* Enable EOC (Bit[5] in ADC_CR1) to be set at the end of each regular conversion
by setting EOCS (Bit[10]) in ADC_CR2 register */
ADC_CR2_CONT, /* Enable continuous conversion mode by setting CONT (Bit[1]) in ADC_CR2 register */
0 /* Configure for highest interrupt priority */
);
/* Declare and initialize a table of Sawtooth Wave samples.
Note: Since the maximum resolution of DAC is 12-bit, each sample is defined as a 16-bit integer */
uint32_t n_len = 250;
/* Initialize and configure DAC channel 1 */
DAC_Init(1, 0, 0);
DAC_Init(2, 0, 0);
g_volume = 0;
g_address = 0;
memset(g_songs, 0, sizeof(g_songs));
uint32_t adc_in[12] = {0};
uint32_t prev_note = 0;
uint32_t note = 0;
uint32_t wait = 0;
uint32_t note_map[] = {1,8,2,9,3,4,10,5,11,6,12,7}; // Array used to specify the note played for each string. */
while(1)
{
/* If the user hits Esc on keyboard */
if(return_to_main())
{
/* Disable TIM2 */
TIM2->CR1 &= ~TIM_CR1_CEN;
/* Disable TIM2 clock */
RCC->APB1ENR &= ~RCC_APB1ENR_TIM12EN;
/* Clear overflow count */
g_tim2_overflow_count = 0;
/* Disable DAC channel 1 by clearing Bit[0] in DAC_CR register */
DAC->CR &= ~DAC_CR_EN1;
/* Disable DAC channel 2 by clearing Bit[16] in DAC_CR register */
DAC->CR &= ~DAC_CR_EN2;
/* Disable DAC clock by clearing DAC EN (Bit[29]) in RCC_APB1ENR register */
RCC->APB1ENR &= ~RCC_APB1ENR_DACEN;
/* Disable ADC1 by clearing ADON (Bit[0]) in ADC1_CR2 register */
ADC1->CR2 &= ~ADC_CR2_ADON;
/* Disable ADC clock by clearing ADC1EN (Bit[8]) in RCC_APB2ENR register */
RCC->APB2ENR &= ~RCC_APB2ENR_ADC1EN;
/* Clear g_adc_results */
memset(g_adc_results, 0, sizeof(g_adc_results));
/* Return to main menu */
return;
}
/* If g_new_adc_read flag is set */
if(g_new_adc_read)
{
```

```
/* Clear g_new_adc_read flag */
g_new_adc_read = 0;
uint32_t i = 0;
for(i = 0; i < 12; i++)
{
/* Calculate the adc input voltage corresponding to the ADC1_IN1 output */
adc_in[i] = ((double)(ADC_VREF / ADC_RES_12 * g_adc_results[i])) > 1.0;
if(adc_in[i])
{
note = i;
}
}
if(note != prev_note)
{
if(g_IsRecording && prev_note)
{
Save_Note(prev_note, g_tim2_overflow_count, wait);
g_tim2_overflow_count = 0;
}
if(note)
{
PlayNote(1);
wait = g_tim2_overflow_count;
g_tim2_overflow_count = 0;
}
else
{
if(g_volume>0xCF0)g_volume -= 50;
else g_volume = 0;
DAC->DHR12R2 = g_volume;
}
prev_note = note;
}
}
}
}
```

mcro_declare.h:

```
#ifndef H_MCRO_DECLARE
#define H_MCRO_DECLARE
/* Declare the delay function before calling it */
void delay(uint32_t num_tick);
/* Declare GPIOx_Init function before calling it */
void GPIOx_Init(GPIO_TypeDef* GPIOx,
uint32_t GPIOx_CLK_EN_BIT_MASK,
uint32_t GPIOx_pin_number,
uint32_t GPIOx_mode,
uint32_t periph_AF_number,
uint32_t GPIOx_output_type,
uint32_t GPIOx_output_speed,
uint32_t GPIOx_pupd);
/* Declare GPIOx_EXTI_Init function before calling it */
void GPIOx_EXTI_Init(uint32_t GPIOx_pin_number,
uint32_t EXTICR_EXTIx_mask,
uint32_t EXTICR_EXTIx_PORTx_mask,
uint32_t EXTI_RTSR_TRx_mask,
uint32_t EXTI_FTSR_TRx_mask,
uint32_t EXTI_IMR_MRx_mask,
uint32_t NVIC_EXTIx_IRQn,
uint32_t NVIC_EXTIx_priority);
/* Declare USARTx_Init() function before calling it */
void USARTx_Init(uint32_t USART_num,
USART_TypeDef * USARTx,
uint32_t USARTx_CLK_EN_bit_mask,
uint32_t word_length,
```

```
uint32_t num_stop_bits,
uint32_t parity,
uint32_t parity_select,
uint32_t oversampling,
uint32_t baud_rate,
uint32_t dma_tx,
uint32_t dma_rx,
uint32_t interrupt_mask,
uint32_t USARTx_IRQn,
uint32_t priority);
/* Declare USARTx_Write_Str() function before calling it */
void USARTx_Write_Str(USART_TypeDef * USARTx, uint8_t *str);
void LED_ON(void);
void LED_OFF(void);
void LED_Blink(void);
void LED_control_using_pushbutton(void);
void LED_control_using_EXTI(void);
void USART2_transmits_message(void);
uint8_t USARTx_Read_Poll(USART_TypeDef * USARTx);
void USART2_controls_LED_poll(void);
uint32_t return_to_main(void);
void LED_control_using_keyboard(void);
void delay_asm(uint32_t num_tick);
void led_on_off_asm(void);
void control_led_using_pushbutton_asm(void);
void Keypad_Init(void);
uint8_t keypad_get_key(void);
void keypad_testing(void);
//void SPI2_Init(void);
void SPI2_Init(uint32_t SPI_CR1_CPHA_mask,
uint32_t SPI_CR1_CPOL_mask,
uint32_t SPI_CR1_MSTR_mask,
uint32_t SPI_CR1_BR_mask,
uint32_t SPI_CR1_LSBFIRST_mask,
uint32_t SPI_CR1_SSI_mask,
uint32_t SPI_CR1_SSM_mask,
uint32_t SPI_CR1_DFF_mask,
uint32_t SPI_CR2_Interrupt_mask,
uint32_t SPI_Interrupt_priority
);
void SPI2_Write(uint8_t data);
void SPI2_Master_Transmits(void);
uint8_t SPI2_Read(void);
void SPI2_Slave_Receives(void);
void SPI2_Slave_Receives_Interrupt(void);
void SPI_Init_Custom(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_CLK_EN_bit_mask,
uint32_t GPIOx_CS_pin,
uint32_t GPIOx_SCK_pin,
uint32_t GPIOx_MISO_pin,
uint32_t GPIOx_MOSI_pin,
uint32_t SPI_mode);
void SPI_Write_Custom(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_CS_pin,
uint32_t GPIOx_SCK_pin,
uint32_t GPIOx_MOSI_pin,
uint8_t data);
void SPI_Master_Transmits_Custom(void);
void USART2_Input_Validation(void);
void sysclk_testing(void);
void I2C_Start(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL);
void I2C_Stop(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL);
```

```
uint32_t I2C_read_ACK(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL);
void I2C_send_data(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL,
uint8_t data);
void I2C_Write(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL,
uint8_t dev_addr,
uint32_t mem_addr,
uint32_t mem_addr_size,
uint8_t data);
uint8_t I2C_Read(GPIO_TypeDef * GPIOx,
uint32_t GPIOx_SDA,
uint32_t GPIOx_SCL,
uint8_t dev_addr,
uint32_t mem_addr,
uint32_t mem_addr_size);
void I2C_EEPROM_Read_Write(void);
void ADCx_Init(ADC_TypeDef * ADCx,
uint32_t ADCx_CLK_EN_bit_mask,
uint32_t ADC_CCR_mask,
uint32_t ADCx_SQR1_mask,
uint32_t ADCx_SQR2_mask,
uint32_t ADCx_SQR3_mask,
uint32_t ADCx_SMPR1_mask,
uint32_t ADCx_SMPR2_mask,
uint32_t ADCx_CR1_mask,
uint32_t ADCx_CR2_mask,
uint32_t ADCx_interrupt_priority);
void ADC1_converts_single_channel_poll(void);
void ADC1_converts_single_channel_interrupt(void);
void DAC_Init(uint32_t DAC_CH_number,
uint32_t DAC_CR_mask,
uint32_t DAC_SWTRIGR_mask);
void DAC1_generates_sine_wave(void);
void DAC1_generates_sawtooth_wave(void);
void DAC1_generates_triangle_wave(void);
void DAC1_generates_square_wave(void);
void ADC1_reads_temperature_sensor(void);
void ADC1_converts_multiple_channels_interrupt(void);
void TIMx_Init_Timer(TIM_TypeDef * TIMx,
uint32_t APB_number,
uint32_t TIMx_CLK_EN_bit_mask,
uint32_t TIMx_PSC_val,
uint32_t TIMx_ARR_val,
uint32_t TIMx_DIER_mask,
uint32_t TIMx_IRQn,
uint32_t TIMx_interrupt_priority);
void delay_ms_timer(uint32_t num_ms);
void TIMx_Init_Capture_Compare(TIM_TypeDef * TIMx,
uint32_t APB_number,
uint32_t TIMx_CLK_EN_bit_mask,
uint32_t TIMx_PSC_val,
uint32_t TIMx_ARR_val,
uint32_t TIMx_ch_number,
uint32_t TIMx_CCMRy_mask,
uint32_t TIMx_CCRy_val,
uint32_t TIMx_CCER_mask,
uint32_t TIMx_SMCR_mask,
uint32_t TIMx_EGR_mask,
uint32_t TIMx_DIER_mask,
uint32_t TIMx_IRQn,
uint32_t TIMx_interrupt_priority);
```

```
void TIM2_Timer(void);
void TIM3_output_compare_toggle_pin(void);
void PWM_TIM3_controls_LED_brightness(void);
void TIM3_controls_PWM_duty_cycle_using_keyboard(void);
void DAC_Test(void);
void DAC_Song(void);
void Keyboard_Piano(void);
void Single_Laser_Tone(void);
void PlayNote(uint8_t note);
void PlaybackNote(uint32_t note, uint32_t duration, uint32_t wait);
void BeginSong(void);
void EndSong(void);
void Save_Note(uint8_t note, uint32_t duration, uint32_t wait);
void Write_Notes(void);
void Read_Notes(void);
void Playback_Songs(void);
void Playback_Song(uint32_t song_number);
void Erase_Song(uint32_t song_number);
void Harp_using_ADC(void);
#endif
```

mcro_define.h:

```
#ifndef H_MCRO_DEFINE
#define H_MCRO_DEFINE
#define ESC (27)
/* Declare the mask for the bit corresponding to pin connected with board LED.
Board LED is connected with PC12 */
#define LED_MASK (1 << 12)
/* Declare the mask for the bit corresponding to the pin connected with board push button */
/* The only push button on the board is connected with PB0 */
#define PUSH_BUTTON_MASK (1 << 0)
/* Declare the mask for the bit associated with turning on/off GPIOC peripheral clock */
/* Bit[2] of RCC_AHB1ENR register is associated with GPIOC */
#define GPIOC_CLK_EN_BIT_MASK (1 << 2)
/* Declare the mask for the bit associated with turning on/off GPIOA peripheral clock */
/* Bit[0] of RCC_AHB1ENR register is associated with GPIOC */
#define GPIOA_CLK_EN_BIT_MASK (1 << 0)
#define GPIOB_CLK_EN_BIT_MASK (1 << 1)
#define USART2_CLK_EN_BIT_MASK (1 << 17)
#define SPI_MASTER_MODE (1 << 2)
#define SPI_SLAVE_MODE (0 << 2)
#define SPI_CPOL1 (1 << 1)
#define SPI_CPOL0 (0 << 1)
#define SPI_CPHASE1 (1 << 0)
#define SPI_CPHASE0 (0 << 0)
#define SPI_LSBFIRST (1 << 7)
#define SPI_MSBFIRST (0 << 7)
#define SPI_SSM (1 << 9)
#define SPI_SSI (1 << 8)
#define SPI_DFF8 (0 << 11)
#define SPI_DFF16 (1 << 11)
#define SPI_RXNE_MASK (1 << 0)
#define SPI_MASTER (0)
#define SPI_SLAVE (1)
#define VALID_STATE (4)
#define BUF_LEN (256)
#define MAX_STR_LEN (50)
#define APB1_CLK (42000000)
#define APB2_CLK (84000000)
#define AHB_CLK (168000000)
#define ADC_VREF (3.3)
#define ADC_RES_12 (0xFFF)
#define ADC_RES_10 (0x3FF)
#define ADC_RES_8 (0xFF)
#define ADC_RES_6 (0x3F)
```

```
#define SAMPLE_SIZE (256)
#define PI (3.14159265358979323)
#define GPIO_MODE_INPUT (0)
#define GPIO_MODE_OUTPUT (1)
#define GPIO_MODE_AF (2)
#define GPIO_MODE_ANALOG (3)
#define GPIO_OTYPE_PUSH_PULL (0)
#define GPIO_OTYPE_OPEN_DRAIN (1)
#define GPIO_OSPEED_LOW (0)
#define GPIO_OSPEED_MEDIUM (1)
#define GPIO_OSPEED_HIGH (2)
#define GPIO_OSPEED_VERY_HIGH (3)
#define GPIO_PUPD_NO_PUPD (0)
#define GPIO_PUPD_PULL_UP (1)
#define GPIO_PUPD_PULL_DOWN (2)
/* Define port pins */
#define PA0 (0)
#define PA1 (1)
#define PA2 (2)
#define PA3 (3)
#define PA8 (8)
#define PB0 (0)
#define PB1 (1)
#define PB2 (2)
#define PB5 (5)
#define PB8 (8)
#define PB9 (9)
#define PB10 (10)
#define PB11 (11)
#define PB12 (12)
#define PB13 (13)
#define PB14 (14)
#define PB15 (15)
#define PC0 (0)
#define PC1 (1)
#define PC2 (2)
#define PC3 (3)
#define PC4 (4)
#define PC5 (5)
#define PC6 (6)
#define PC7 (7)
#define PC8 (8)
#define PC9 (9)
#define PC10 (10)
#define PC12 (12)
#define PC13 (13)
#define PD2 (2)
#endif
```

mcro_global.h:

```
#ifndef H_MCRO_GLOBAL
#define H_MCRO_GLOBAL
extern uint32_t g_exti0_count;
extern uint32_t g_usart2_read_count;
extern uint8_t g_usart2_read_buffer[256];
extern uint32_t g_spi2_read_count;
extern uint8_t g_spi2_read_buffer[256];
extern uint16_t g_adc_read;
extern uint32_t g_new_adc_read;
extern uint16_t g_adc_results[12];
extern uint32_t g_adc_mode;
extern uint32_t g_tim2_overflow_count;
extern uint32_t g_n_len;
extern uint32_t g_volume;
extern uint32_t g_address;
```

```
extern uint32_t g_number_of_notes;
extern uint32_t g_number_of_songs;
extern uint8_t g_songs[6000];
extern uint32_t g_song_indexes[1000][2];
extern uint32_t g_IsRecording;
#endif
```

mcro_include.h:

```
#ifndef H_MCRO_INCLUDE
#define H_MCRO_INCLUDE
#include <__cross_studio_io.h>
#include "stm32f4xx.h"
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>
#include "mcro_define.h"
#include "mcro_declare.h"
#include "mcro_global.h"
#endif
```

mcro_main.c:

```
/*-----------------------------------------------------------------------------------------------------------------------FILENAME: mcro_main.c
DESCRIPTION: The file mcro_main.c is the entry point to the application.
It uses the peripheral USART2 to transmit data by polling and receive data by interrupt.
The function USARTx_Write_Str() is used to transmit the menu item messages which are displayed on teraterm.
The menu items are defined in the char variable g_main_menu[].
The menu items contain Main Menu and all the demo names of this project.
Each menu item is associated with a character keyboard-pressed by the user.
Users are prompted to press a character on the keyboard to run a particular demo/application.
The character inputs from the users are received by the USART receiver using
receiver not empty (RXNE) interrupt.
The received characters are stored in the array g_usart2_receive_buffer.
When a character is available at g_usart2_receive_buffer, it is read and assigned to the variable "keyboard_If the received character is not the
Esc character,
the program calls the demo function related to the received character.
The demo function selection is done by the switch - case construct.
Before reading the character, the global interrupt is
enabled by CMSIS function __enable_irq().
All demo functions display the prompt and wait for the ESC character from
the TeraTerm.
HARDWARE: Olimex prototyping board with STM32F405RG microcontroller
AUTHOR: Iouri Kourilov
Modified and commented by Habib Islam, Rohnin Menezes
-----------------------------------------------------------------------------------------------------------------------/*
CODE SUMMARY:
This program allows notes to be played using the keyboard. Songs can be recorded, played back, and erased. Songs can from an EEPROM. The
EEPROM to be used is the 25LC256. On the EEPROM, Pins 1,2,3,4, and 7 are connected to ground, Pin and Pin 8 is connected to the MCU's 3.3V
pin. The main functions of this program can be found in note_functions.c, harp_The file harp_debug_functions.c contains code used to run tests
on the hardware. The original design for the harp was Photoresistors (connected to standard resistors to form Voltage Dividers) using the ADC
inputs, in order to detect when The function Harp_using_ADC() allows songs to be played using the MCU's ADCs as inputs instead of the
keyboard, however the USART2_TX pin as an ADC input, and therefore cannot be tested using USART2.
*/
/* Include the header file mcro350_main.h that includes all the required header files */
#include "mcro_include.h"
/* Define the char array g_main_menu[] */
uint8_t g_main_menu[] = "\n\r\n\r\t\t\tMain Menu\n\r\n\r\
\t1 - dac_piano\r\n";
/* Declare the message that would prompt the user to press Esc on keyboard to disable a function and go back to the main uint8_t
g_ESC_msg[] = "\n\r\t\t------Press Esc on keyboard to return to the main menu----\n\r\n";
int32_t main(void)
{
/* Decalre and initialize an 8-bit unsigned integer variable keyboard_input. This variable will be assigned
```

```
with the character read from the g_usart2_receive_buffer */
uint8_t keyboard_input = 0;
/* Configure GPIOA registers to initialize PA2 as USART2 TxD pin */
GPIOx_Init(GPIOA, /* Define a pointer that points to the base address of GPIOA */
GPIOA_CLK_EN_BIT_MASK, /* Enable the clock of GPIOA by setting Bit[1] in RCC_AHB1ENR */
2, /* GPIOA pin number 2 */
2, /* Set pin 10 in AF mode by writing 2 to Bits[21:20] in GPIOB_MODER */
7, /* AF number is 7 */
0, /* Set output type to push-pull by clearing Bit[0] in GPIOA_OTYPER */
2, /* Set GPIO speed as HIGH speed by writing 0b10 to Bits[1:0] in GPIOA_OSPEEDR */
0 /* Select "no pull-up pull-down by clearing Bits[1:0] in GPIOA_PUPDR */
);
/* Configure GPIOA registers to initialize PA3 as USART2 RxD pin */
GPIOx_Init(GPIOA, /* Define a pointer that points to the base address of GPIOA */
GPIOA_CLK_EN_BIT_MASK, /* Enable the clock of GPIOA by setting Bit[1] in RCC_AHB1ENR */
3, /* GPIOA pin number 3 */
2, /* Set pin 10 in AF mode by writing 2 to Bits[21:20] in GPIOB_MODER */
7, /* AF number is 7 */
0, /* Set output type to push-pull by clearing Bit[0] in GPIOA_OTYPER */
2, /* Set GPIO speed as HIGH speed by writing 0b10 to Bits[1:0] in GPIOA_OSPEEDR */
0 /* Select "no pull-up pull-down by clearing Bits[1:0] in GPIOA_PUPDR */
);
/* Configure and initialize USART2 by calling USARTx_Init() function with the desired parameters*/
USARTx_Init(2, /* Select USART2 */
USART2, /* Pointer to the base address of USART2 */
USART2_CLK_EN_BIT_MASK, /* Set Bit[17] in RCC_APB1ENR to turn on USART2 peripheral clock */
0, /* Configure for 8-bits of character length */
0, /* Configure for 1 stop bit */
0, /* Disable parity control */
0, /* Doesn't matter since parity control is disabled */
0, /* Configure for 16x oversampling */
(273 << 4) | (7 << 0), /* USARTDIV = f_PCLK/(8*(2-OVER8)*baud_rate) = 42 MHz/(8*(2-0)*9600 bps) = 273.4375
Write Mantissa 273 to Bits[15:4] and fraction 0.4375*16 = 7 to Bits[3:0] in USART2_0, /* Disalbe DMA transmit */
0, /* Disable DMA receive */
USART_CR1_RXNEIE, /* Configure for RXNE interrupt */
USART2_IRQn, /* IRQ number for USART2: doesn't matter since no interrupt is configured */
0 /* Highest priority */
);
/* Display g_main_menu items */
USARTx_Write_Str(USART2, g_main_menu);
/* Display g_ESC_message */
USARTx_Write_Str(USART2, g_ESC_msg);
/* Enable global interrupt (Note: there are two underscores before the word enable) */
__enable_irq(); // Enable global interrupt
/* Begin infinite loop */
while(1)
{
/* If no key is pressed, ignore successive commands */
if(g_usart2_read_count == 0) continue;
/* If a key is pressed, assign the received character to the variable keyboard_input */
keyboard_input = g_usart2_read_buffer[g_usart2_read_count-1];
switch(keyboard_input)
{
case '1': Keyboard_Piano(); break;
/* Requires use of the USART2_TX pin as an ADC input. */
//case '2': Harp_using_ADC(); break;
default: g_usart2_read_count = 0;
}
/* Display g_main_menu items */
USARTx_Write_Str(USART2, g_main_menu);
/* Clear read count */
g_usart2_read_count = 0;
} /* End of while(1) */
} /* End of main(void) */
/* Define the delay function. This function asks the processor
to wait for a specified amount of time before executing the
```

```
next instruction. In the while loop, the processor wait
until the selected number 'num_tick' counts down to 0. */
void delay(uint32_t num_tick)
{
/* Wait until 'num_tick' counts down to 0 */
while(num_tick--);
}
```

note_functions.c:

```
#include "mcro_include.h"
uint32_t g_n_len = 250;
void PlayNote(uint8_t note)
{
g_volume = 0xEFF;
uint32_t note_id;
switch(note)
{
case 1:
note_id = 2842;
break;
case 2:
note_id = 2700;
break;
case 3:
note_id = 2564;
break;
case 4:
note_id = 2494;
break;
case 5:
note_id = 2351;
break;
case 6:
note_id = 2201;
break;
case 7:
note_id = 2057;
break;
case 8:
note_id = 2781;
break;
case 9:
note_id = 2630;
break;
case 10:
note_id = 2423;
break;
case 11:
note_id = 2273;
break;
case 12:
note_id = 2130;
break;
}
DAC->DHR12R1 = note_id;
DAC->DHR12R2 = g_volume;
uint8_t display[MAX_STR_LEN] = {0};
uint8_t display_format[] = "\n\r\tnote = %d.\r\t";
sprintf(display, display_format, note);
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
}
void PlaybackNote(uint32_t note, uint32_t duration, uint32_t wait)
{
delay_ms_timer(wait);
```

```
g_volume = 0xEFF;
uint32_t note_id;
switch(note)
{
case 1:
note_id = 2842;
break;
case 2:
note_id = 2700;
break;
case 3:
note_id = 2564;
break;
case 4:
note_id = 2494;
break;
case 5:
note_id = 2351;
break;
case 6:
note_id = 2201;
break;
case 7:
note_id = 2057;
break;
case 8:
note_id = 2781;
break;
case 9:
note_id = 2630;
break;
case 10:
note_id = 2423;
break;
case 11:
note_id = 2273;
break;
case 12:
note_id = 2130;
break;
}
DAC->DHR12R1 = note_id;
DAC->DHR12R2 = g_volume;
/* Declare and initialize the char array to display data */
uint8_t display[MAX_STR_LEN] = {0};
/* Define the display format */
uint8_t display_format[] = "\n\r\tnote = %d, duration = %d, wait = %d\r\t";
sprintf(display, display_format, note, duration, wait);
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
delay_ms_timer(duration);
}
```

recording_functions.c:

```
#include "mcro_include.h"
uint32_t g_address = 0;
uint32_t g_number_of_songs = 0;
uint32_t g_song_indexes[1000][2] = {{0}}; // array used to hold the beginning and ending indexes of each song
/* Define the device address of EEPROM with WRITE (Bits[7:1] are address bits and Bit[0] is the Write bit */
#define DEV_ADDR (0xA0)
/* Delay between read and write operations */
#define READ_WRITE_DELAY (0xFF)
uint8_t g_songs[6000] = {0};
void clear_song_indexes() // Clears the song index array
{
```

```
uint32_t i = 0;
for(i = 0; i < sizeof(g_song_indexes)/sizeof(g_song_indexes[0]); i++)
{
g_song_indexes[i][0] = 0;
g_song_indexes[i][1] = 0;
}
}
void BeginSong(void) // Begins a new song recording
{
g_songs[g_address] = 0x33;
g_song_indexes[g_number_of_songs][0] = g_address;
g_address++;
}
void EndSong(void) // Ends the current song recording
{
g_songs[g_address] = 0xCC;
g_song_indexes[g_number_of_songs][1] = g_address;
g_number_of_songs++;
g_address++;
}
void Save_Note(uint8_t note, uint32_t duration, uint32_t wait) // Saves a note in RAM
{
g_songs[g_address] = note;
g_address++;
uint32_t i;
for(i = 0; i < 32; i += 8)
{
g_songs[g_address] = (uint8_t)((duration & (0xFF << i)) >> i);
g_address++;
}
for(i = 0; i < 32; i += 8)
{
g_songs[g_address] = (uint8_t)((wait & (0xFF << i)) >> i);
g_address++;
}
}
void Write_Notes(void) // Saves all note data to the EEPROM
{
/* Declare an array of 50 characters to display data on TeraTerm */
uint8_t display[50];
/* Configure GPIOC pin PC0 as an I2C SDA pin */
GPIOx_Init(GPIOC, /* Base address of GPIOC is a pointer to GPIO_TypeDef structure */
GPIOC_CLK_EN_BIT_MASK, /* Enable GPIOC clock by setting Bit[0] in RCC_AHB1ENR register */
0, /* Select GPIOC pin to be PC0 */
1, /* Configure PC0 in Output mode by writing 0b01 (1) to Bits[1:0] in GPIOC_MODER register 0, /* Since GPIO mode is OUTPUT, AF number
doesn't matter */
1, /* Configure PC0 output type to be OPEN_DRAIN by setting Bit[0] in GPIOC_OTYPER register 2, /* Configure PC0 output speed to be HIGH by
writing 0b10 to Bits[1:0] in GPIOA_OSPEEDR 1 /* Configure PC0 to be PULL-UP by writing 0b01 (1) to Bits[1:0] in GPIOA_PUPDR register );
/* Configure GPIOC pin PC1 as an I2C SCL pin */
GPIOx_Init(GPIOC, /* Base address of GPIOC is a pointer to GPIO_TypeDef structure */
GPIOC_CLK_EN_BIT_MASK, /* Enable GPIOC clock by setting Bit[2] in RCC_AHB1ENR register */
1, /* Select GPIOC pin to be PC1 */
1, /* Configure PC1 in OUTPUT mode by writing 0b01 (1) to Bits[3:2] in GPIOC_MODER register 0, /* Since GPIO mode is OUTPUT, AF number
doesn't matter */
1, /* Configure PC1 output type to be OPEN_DRAIN by setting Bit[1] in GPIOC_OTYPER register 2, /* Configure PC1 output speed to be HIGH by
writing 0b10 to Bits[3:2] in GPIOC_OSPEEDR 1 /* Configure PC1 to be PULL-UP by writing 0b01 (1) to Bits[3:2] in GPIOA_PUPDR register );
/* Drive SDA (PC0) and SCL (PC1) HIGH */
GPIOC->BSRR = (1 << 0) | (1 << 1);
uint32_t i = 0;
uint8_t byte_number_display[] = "\r\tWriting byte %d of %d.\r";
for(i = 0; i < g_address; i++)
{
sprintf(display, byte_number_display, i, (sizeof(g_songs)/sizeof(g_songs[0])));
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
/* Write to the EEPROM. Note: for EEPROM 24LC256, use mem_addr_size = 2 */
```

```
I2C_Write(GPIOC, 0, 1, DEV_ADDR, i, 2, g_songs[i]);
delay(READ_WRITE_DELAY);
}
sprintf(display, byte_number_display, i, (sizeof(g_songs)/sizeof(g_songs[0])));
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
/* Write an empty byte after the last song to specifiy the end of data when reading from the EEPROM */
I2C_Write(GPIOC, 0, 1, DEV_ADDR, g_address, 2, i);
delay(READ_WRITE_DELAY);
USARTx_Write_Str(USART2, "\n\rWrite complete.\n");
}
void Read_Notes(void) // Read note data from the EEPROM
{
/* Declare an array of 50 characters to display data on TeraTerm */
uint8_t display[50];
/* Configure GPIOC pin PC0 as an I2C SDA pin */
GPIOx_Init(GPIOC, /* Base address of GPIOC is a pointer to GPIO_TypeDef structure */
GPIOC_CLK_EN_BIT_MASK, /* Enable GPIOC clock by setting Bit[0] in RCC_AHB1ENR register */
0, /* Select GPIOC pin to be PC0 */
1, /* Configure PC0 in Output mode by writing 0b01 (1) to Bits[1:0] in GPIOC_MODER register 0, /* Since GPIO mode is OUTPUT, AF number
doesn't matter */
1, /* Configure PC0 output type to be OPEN_DRAIN by setting Bit[0] in GPIOC_OTYPER register 2, /* Configure PC0 output speed to be HIGH by
writing 0b10 to Bits[1:0] in GPIOA_OSPEEDR 1 /* Configure PC0 to be PULL-UP by writing 0b01 (1) to Bits[1:0] in GPIOA_PUPDR register );
/* Configure GPIOC pin PC1 as an I2C SCL pin */
GPIOx_Init(GPIOC, /* Base address of GPIOC is a pointer to GPIO_TypeDef structure */
GPIOC_CLK_EN_BIT_MASK, /* Enable GPIOC clock by setting Bit[2] in RCC_AHB1ENR register */
1, /* Select GPIOC pin to be PC1 */
1, /* Configure PC1 in OUTPUT mode by writing 0b01 (1) to Bits[3:2] in GPIOC_MODER register 0, /* Since GPIO mode is OUTPUT, AF number
doesn't matter */
1, /* Configure PC1 output type to be OPEN_DRAIN by setting Bit[1] in GPIOC_OTYPER register 2, /* Configure PC1 output speed to be HIGH by
writing 0b10 to Bits[3:2] in GPIOC_OSPEEDR 1 /* Configure PC1 to be PULL-UP by writing 0b01 (1) to Bits[3:2] in GPIOA_PUPDR register );
g_number_of_songs = 0;
clear_song_indexes();
/* Drive SDA (PC0) and SCL (PC1) HIGH */
GPIOC->BSRR = (1 << 0) | (1 << 1);
uint32_t i = 0;
uint8_t byte_number_display[] = "\r\tReading byte %d of %d.";
while(i < (sizeof(g_songs)/sizeof(g_songs[0])))
{
sprintf(display, byte_number_display, i, (sizeof(g_songs)/sizeof(g_songs[0])));
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
/* Read from the EEPROM. Note: for EEPROM 24LC256, use mem_addr_size = 2 */
g_songs[i] = I2C_Read(GPIOC, 0, 1, DEV_ADDR, i, 2);
delay(READ_WRITE_DELAY);
if (g_songs[i] == 0x33)
{
g_song_indexes[g_number_of_songs][0] = i;
i++;
while (1)
{
sprintf(display, byte_number_display, i, (sizeof(g_songs) / sizeof(g_songs[0])));
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
g_songs[i] = I2C_Read(GPIOC, 0, 1, DEV_ADDR, i, 2);
delay(READ_WRITE_DELAY);
uint8_t tmp = g_songs[i];
if (g_songs[i] == 0xCC)
{
g_song_indexes[g_number_of_songs][1] = i;
g_number_of_songs++;
i++;
break;
}
i++;
uint32_t j = 0;
```

```
for (j = 0; j < 8; j++)
{
sprintf(display, byte_number_display, i, (sizeof(g_songs) / sizeof(g_songs[0])));
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
g_songs[i] = I2C_Read(GPIOC, 0, 1, DEV_ADDR, i, 2);
delay(READ_WRITE_DELAY);
i++;
}
}
}
else break;
}
g_address = i;
USARTx_Write_Str(USART2, "\n\rRead complete.\n");
}
void Playback_Songs() // Plays all recorded songs in order
{
uint32_t i = 0;
g_number_of_songs = 0;
clear_song_indexes();
/* Declare and initialize the char array to display data */
uint8_t display[MAX_STR_LEN] = {0};
while(i < (sizeof(g_songs)/sizeof(g_songs[0])) )
{
if(g_songs[i] == 0x33)
{
g_song_indexes[g_number_of_songs][0] = i;
i++;
/* Define the display format */
uint8_t song_number_display[] = "\n\rPlaying song number %d:\r\t";
sprintf(display, song_number_display, g_number_of_songs + 1);
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
while(1)
{
uint32_t j = 0;
uint8_t note = 0;
uint32_t duration = 0;
uint32_t wait = 0;
if(g_songs[i] == 0xCC)
{
g_song_indexes[g_number_of_songs][1] = i;
uint8_t song_indexes_display[] = "\n\rSong %d occupies indexes %d to %d. Total length = %d bytes\r\t";
uint32_t song_length = g_song_indexes[g_number_of_songs][1] - g_song_indexes[g_number_of_songs][0];
sprintf(display,
song_indexes_display,
g_number_of_songs + 1,
g_song_indexes[g_number_of_songs][0],
g_song_indexes[g_number_of_songs][1],
song_length
);
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
g_number_of_songs++;
i++;
break;
}
note = g_songs[i];
i++;
for(j = 0; j < 32; j += 8)
{
duration |= (uint32_t)g_songs[i] << j;
i++;
}
for(j = 0; j < 32; j += 8)
```

```
{
wait |= (uint32_t)g_songs[i] << j;
i++;
}
PlaybackNote(note, duration, wait);
}
}
else break;
}
/* Define the display format */
uint8_t total_songs_display[] = "\n\rtotal songs recorded = %d.\r\t";
sprintf(display, total_songs_display, g_number_of_songs);
/* Write the string array to USART_DR register to display on TeraTerm */
USARTx_Write_Str(USART2, display);
} /* End of I2C_EERPOM_Read_Write() */
void Playback_Song(uint32_t song_number) // Plays a recorded song by index
{
uint32_t i = g_song_indexes[song_number][0] + 1;
while (i < g_song_indexes[song_number][1])
{
uint32_t j = 0;
uint8_t note = 0;
uint32_t duration = 0;
uint32_t wait = 0;
note = g_songs[i];
i++;
for (j = 0; j < 32; j += 8)
{
duration |= (uint32_t)g_songs[i] << j;
i++;
}
for (j = 0; j < 32; j += 8)
{
wait |= (uint32_t)g_songs[i] << j;
i++;
}
PlaybackNote(note, duration, wait);
}
}
void Erase_Song(uint32_t song_number) // Erases a recorded song by index
{
uint32_t i = g_song_indexes[song_number][0];
uint32_t j = g_song_indexes[song_number][1] + 1;
while (j < (sizeof(g_songs)/sizeof(g_songs[0])))
{
g_songs[i] = g_songs[j];
i++;
j++;
}
/* Update Song indexes and number of songs */
i = 0;
g_number_of_songs = 0;
clear_song_indexes();
while(i < (sizeof(g_songs)/sizeof(g_songs[0])))
{
if (g_songs[i] == 0x33)
{
g_song_indexes[g_number_of_songs][0] = i;
i++;
while (1)
{
if (g_songs[i] == 0xCC)
{
g_song_indexes[g_number_of_songs][1] = i;
g_number_of_songs++;
i++;
```

```
break;
}
i += 9;
}
}
else break;
}
g_address = i;
}
```

timer_functions.c:

```
#include "mcro_include.h"
void TIMx_Init_Timer(TIM_TypeDef * TIMx,
uint32_t APB_number,
uint32_t TIMx_CLK_EN_bit_mask,
uint32_t TIMx_PSC_val,
uint32_t TIMx_ARR_val,
uint32_t TIMx_DIER_mask,
uint32_t TIMx_IRQn,
uint32_t TIMx_interrupt_priority)
{
/* If TIMx is connected to APB1 */
if(APB_number == 1)
{
/* Enable TIMx clock using RCC_APB1ENR register */
RCC->APB1ENR |= TIMx_CLK_EN_bit_mask;
}
/* If TIMx is connected to APB2 */
else if(APB_number == 2)
{
/* Enable TIMx clock using RCC_APB2ENR register */
RCC->APB2ENR |= TIMx_CLK_EN_bit_mask;
}
/* Disable TIMx before configuring its registers */
TIMx->CR1 &= ~TIM_CR1_CEN;
/* Write the desired PSC value to TIMx_PSC register */
TIMx->PSC = TIMx_PSC_val;
/* Write the desired ARR value to TIMx_PSC register */
TIMx->ARR = TIMx_ARR_val;
/* Configure TIMx_DIER to enable the desired interrupts */
TIMx->DIER |= TIMx_DIER_mask;
/* Clear TIMx_CNT register */
TIMx->CNT = 0;
/* Enable TIMx */
TIMx->CR1 |= TIM_CR1_CEN;
/* Set TIMx interrupt priority */
NVIC_SetPriority(TIMx_IRQn, TIMx_interrupt_priority);
/* Enable NVIC level interrupt */
NVIC_EnableIRQ(TIMx_IRQn);
}
uint32_t g_tim2_overflow_count = 0;
void TIM2_IRQHandler(void)
{
/* If UIF flag is set */
if(TIM2->SR & TIM_SR_UIF)
{
/* Clear UIF flag */
TIM2->SR &= ~TIM_SR_UIF;
/* Increment the overflow count */
g_tim2_overflow_count++;
}
}
void TIMx_Init_Capture_Compare(TIM_TypeDef * TIMx,
uint32_t APB_number,
uint32_t TIMx_CLK_EN_bit_mask,
```

```
uint32_t TIMx_PSC_val,
uint32_t TIMx_ARR_val,
uint32_t TIMx_ch_number,
uint32_t TIMx_CCMRy_mask,
uint32_t TIMx_CCRy_val,
uint32_t TIMx_CCER_mask,
uint32_t TIMx_SMCR_mask,
uint32_t TIMx_EGR_mask,
uint32_t TIMx_DIER_mask,
uint32_t TIMx_IRQn,
uint32_t TIMx_interrupt_priority)
{
/* If TIMx is connected to APB1 */
if(APB_number == 1)
{
/* Enable TIMx clock using RCC_APB1ENR register */
RCC->APB1ENR |= TIMx_CLK_EN_bit_mask;
}
/* If TIMx is connected to APB2 */
else if(APB_number == 2)
{
/* Enable TIMx clock using RCC_APB2ENR register */
RCC->APB2ENR |= TIMx_CLK_EN_bit_mask;
}
/* Disable TIMx before configuring its registers */
TIMx->CR1 &= ~TIM_CR1_CEN;
/* Write the desired PSC value to TIMx_PSC register */
TIMx->PSC = TIMx_PSC_val;
/* Write the desired ARR value to TIMx_PSC register */
TIMx->ARR = TIMx_ARR_val;
if (TIMx_ch_number == 1)
{
/* Clear OC1M, CC1S, IC1F, and IC1PSC bits in TIMx_CCMR1 register */
TIMx->CCMR1 &= ~(TIM_CCMR1_OC1M | TIM_CCMR1_CC1S | TIM_CCMR1_IC1F | TIM_CCMR1_IC1PSC);
/* Configure CCMR1 register with desired parameters */
TIMx->CCMR1 |= TIMx_CCMRy_mask;
/* Clear CC1NP (Bit[3]) in TIMx_CCER register */
TIMx->CCER &= ~TIM_CCER_CC1NP;
/* Write desired value for TIMx_CCR register */
TIMx->CCR1 = TIMx_CCRy_val;
}
else if (TIMx_ch_number == 2)
{
/* Clear OC1M, CC1S, IC1F, and IC1PSC bits in TIMx_CCMR1 register */
TIMx->CCMR1 &= ~(TIM_CCMR1_OC2M | TIM_CCMR1_CC2S | TIM_CCMR1_IC2F | TIM_CCMR1_IC2PSC);
/* Configure CCMR1 register with desired parameters */
TIMx->CCMR1 |= TIMx_CCMRy_mask;
/* Clear CC1NP (Bit[3]) in TIMx_CCER register */
TIMx->CCER &= ~TIM_CCER_CC2NP;
/* Write desired value for TIMx_CCR register */
TIMx->CCR2 = TIMx_CCRy_val;
}
else if (TIMx_ch_number == 3)
{
/* Clear OC1M, CC1S, IC1F, and IC1PSC bits in TIMx_CCMR1 register */
TIMx->CCMR2 &= ~(TIM_CCMR2_OC3M | TIM_CCMR2_CC3S | TIM_CCMR2_IC3F | TIM_CCMR2_IC3PSC);
/* Configure CCMR1 register with desired parameters */
TIMx->CCMR2 |= TIMx_CCMRy_mask;
/* Clear CC1NP (Bit[3]) in TIMx_CCER register */
TIMx->CCER &= ~TIM_CCER_CC3NP;
/* Write desired value for TIMx_CCR register */
TIMx->CCR3 = TIMx_CCRy_val;
}
else if (TIMx_ch_number == 4)
{
/* Clear OC1M, CC1S, IC1F, and IC1PSC bits in TIMx_CCMR1 register */
```

```
TIMx->CCMR2 &= ~(TIM_CCMR2_OC4M | TIM_CCMR2_CC4S | TIM_CCMR2_IC4F | TIM_CCMR2_IC4PSC);
/* Configure CCMR1 register with desired parameters */
TIMx->CCMR2 |= TIMx_CCMRy_mask;
/* Clear CC1NP (Bit[3]) in TIMx_CCER register */
TIMx->CCER &= ~TIM_CCER_CC4NP;
/* Write desired value for TIMx_CCR register */
TIMx->CCR4 = TIMx_CCRy_val;
}
/* Clear TIMx_CNT register */
TIMx->CNT = 0;
TIMx->SMCR &= ~TIMx_SMCR_mask;
TIMx->SMCR |= TIMx_SMCR_mask;
TIMx->EGR &= ~TIMx_EGR_mask;
TIMx->EGR |= TIMx_EGR_mask;
TIMx->CCER &= ~TIMx_CCER_mask;
TIMx->CCER |= TIMx_CCER_mask;
TIMx->DIER &= ~TIMx_DIER_mask;
/* Configure TIMx_DIER to enable the desired interrupts */
TIMx->DIER |= TIMx_DIER_mask;
TIMx->CR1 |= TIM_CR1_CEN;
NVIC_SetPriority(TIMx_IRQn, TIMx_interrupt_priority);
NVIC_EnableIRQ(TIMx_IRQn);
}
```

usart_functions.c:

```
/*-----------------------------------------------------------------------------
FILENAME: usart_functions.c
HEADER: mcro_main.h
DESCRIPTION: This file contains the definitions of all the functions that are used for transmitting
receiving data using USART.
This file also contains the USART2 interrupt service routine named USART2_IRQHandler()
REFERENCES: STM32F405 Datasheet
AUTHOR: Habib Islam
-----------------------------------------------------------------------------*/
#include "mcro_include.h"
/*-----------------------------------------------------------------------------
FUNCTION: void USARTx_Init(uint32_t USART_num,
USART_TypeDef* USARTx,
uint32_t USARTx_CLK_EN_bit_mask,
uint32_t word_length,
uint32_t num_stop_bits,
uint32_t parity,
uint32_t parity_select,
uint32_t oversampling,
uint32_t baud_rate,
uint32_t dma_tx,
uint32_t dma_rx,
uint32_t interrupt_mask,
uint32_t USARTx_IRQn,
uint32_t priority)
DESCRIPTION: Configures and Initializes any USART using a configurable interrupt
PARAMETERS: USART_num - number index for USART in STM32F405RG, number = 1, 2, 3, .....etc
USARTx - pointer to the base address of USARTx, x = 1, 2, 3, etc.
USARTx_CLK_EN_bit_mask - Enables USARTx clock
word_length - character length
number_stop_bits - number of stop bits
parity - parity enable/disable
parity_select - even/odd parity
oversampling - 8x/16x oversampling
baud_rate - baud rate
dma_tx - enables/disables DMA transmit
dma_rx - enables/disables DMA receive
interrupt_mask - USARTx bit mask for enabling desired interrupt
USARTx_IRQn - USARTx IRQ number
priority - Desired interrupt priority for USARTx
```

```
RETURNS: None
----------------------------------------------------------------------------*/
/* Function prototype declaration */
void USARTx_Init(uint32_t USART_num,
USART_TypeDef* USARTx,
uint32_t USARTx_CLK_EN_bit_mask,
uint32_t word_length, /* 00 = 8 data bits, 01 = 9 data bits */
uint32_t num_stop_bits, /* 00 = 1 stop bit, 01 = 0.5 stop bit, 10 = 2 stop bits, 11 = 1.5 stop uint32_t parity, /* 0 = parity disabled, 1 = parity
enabled */
uint32_t parity_select, /* 0 = even parity, 1 = odd parity */
uint32_t oversampling, /* 0 = 16x oversampling, 1 = 8x oversampling */
uint32_t baud_rate, /* Baud Rate = f_PCLK/(8*(2-OVER8)*USARTDIV) */
uint32_t dma_tx, /* 0 = TX DMA disabled, 1 = TX DMA enabled */
uint32_t dma_rx, /* 0 = RX DMA disabled, 1 = RX DMA enabled */
uint32_t interrupt_mask, /* TXE = Bit[7], TC = Bit[6], RXNE = Bit[5] */
uint32_t USARTx_IRQn,
uint32_t priority
)
{
/* If USART2 or USART3 is selected, enable its clock by setting
the corresponding bit in RCC_APB1ENR register */
if ((USART_num == 2)|(USART_num == 3))
{
RCC->APB1ENR |= USARTx_CLK_EN_bit_mask;
}
/* If USART1 or USART6 is selected, enable its clock by setting
the corresponding bit in RCC_APB2ENR register */
else if ((USART_num == 1)|(USART_num == 6))
{
RCC->APB2ENR |= USARTx_CLK_EN_bit_mask;
}
USARTx->CR1 &= ~(1 << 13); // Disable USART
USARTx->CR1 &= ~(3 << 12); // clear M filed
USARTx->CR1 |= word_length << 12; // set the word length
USARTx->CR2 &= ~(3 << 12); // clear STOP field (Bits[13:12])
USARTx->CR2 |= num_stop_bits << 12; // set the number of stop bits
USARTx->CR1 &= ~(1 << 10); // clear PCE field (Bit[10])
USARTx->CR1 |= parity << 10; // enable/disable parity control
USARTx->CR1 &= ~(1 << 9); // clear PS field (Bit[9])
USARTx->CR1 |= parity_select << 9; // select the desired parity
USARTx->CR1 &= ~(1 << 15); // clear OVER8 field (Bit[15])
USARTx->CR1 |= oversampling << 15; // set the desired oversampling
USARTx->BRR |= baud_rate; // set the desired baud rate
USARTx->CR3 &= ~(1 << 7); // cleart DMAT field (Bit[7])
USARTx->CR3 |= dma_tx << 7; // enable/disable DMA for transmit
USARTx->CR3 &= ~(1 << 6); // cleart DMAR field (Bit[6])
USARTx->CR3 |= dma_rx << 6; // enable/disable DMA for receive
USARTx->CR1 |= 3 << 2; // Enable TX and RX
USARTx->CR1 |= 1 << 13; // Enable USART
USARTx->CR1 &= ~interrupt_mask; // clear desired interrupt bit fields
USARTx->CR1 |= interrupt_mask; // enable desired interrupt
NVIC_SetPriority(USARTx_IRQn, priority); // set the priority
NVIC_EnableIRQ(USARTx_IRQn); // Enable IRQ for USARTx
}
/*----------------------------------------------------------------------------
FUNCTION: void USARTx_Write_Str(USART_TypeDef * USARTx, uint8_t * str)
DESCRIPTION: Transmits a string of characters using USART
PARAMETERS: USARTx - Pointer to the base address of USARx
str - Pointer to the string to be sent
RETURNS: None
AUTHOR: Habib Islam
----------------------------------------------------------------------------*/
/* Function prototype declaration */
void USARTx_Write_Str(USART_TypeDef *USARTx, uint8_t *str)
/* Beginning of function definition */
{
```

```
uint32_t i = 0; /* Declare and initialize an index for the string elements */
for(i = 0; i < strlen(str); i++)
{
/* Wait until TC (Transmission Complete) bit (Bit[6]) in USART_SR register is set */
while(!(USARTx->SR & USART_SR_TC)); /* while((USARTx->SR & (1 << 6) == 0) */
/* Write a character to be transmitted to the Data Register (USART_DR) */
USARTx->DR = (str[i] & 0xFF);
} /* End of for(i = 0; i < strlen(str); i++) loop */
} /* End of USARTx_Write_Str */
/*---------------------------------------------------------------------------
FUNCTION: uint8_t USARTx_Read_Poll(USART_TypeDef * USARTx)
DESCRIPTION: Reads a character received by USART_DR
PARAMETERS: USARTx - Pointer to the base address of USARx
RETURNS: Character
AUTHOR: Habib Islam
---------------------------------------------------------------------------*/
uint8_t USARTx_Read_Poll(USART_TypeDef * USARTx)
{
/* Wait until RXNE (Bit[5]) in USARTx_SR is set */
while(!(USARTx->SR & (1 << 5)));
/* Read and return the data in USARTx_DR */
return (uint8_t)USARTx->DR;
}
/*---------------------------------------------------------------------------
FUNCTION: void USART2_IRQHandler(void)
DESCRIPTION: Define ISR for USART2 RXNE interrupt
PARAMETERS: None
RETURNS: None
---------------------------------------------------------------------------*/
/* Initialize the number of characters received by USARTx_DR */
uint32_t g_usart2_read_count = 0;
/* Initialize the array that would store the characters read from USARTx_DR */
uint8_t g_usart2_read_buffer[256] = {};
void USART2_IRQHandler(void)
{
/* If RXNE (Bit[5]) flag is set in USART2_SR */
if(USART2->SR & (1 << 5))
{
/* Increment the read count */
g_usart2_read_count ++;
/* Read data into read buffer array */
g_usart2_read_buffer[g_usart2_read_count - 1] = USART2->DR;
/* If the read buffer is full */
if(g_usart2_read_count >= 256)
{
/* Clear read buffer array */
memset(g_usart2_read_buffer, 0, 256);
/* Reset the read count */
g_usart2_read_count = 0;
}
}
}
/*---------------------------------------------------------------------------
FUNCTION: uint32_t return_to_main(void)
DESCRIPTION: This function allows the user to exit from a demo function and
return to the main function. After the user hits ESC on keyboard,
the main menu items are displayed on TeraTerm terminal.
PARAMETERS: None
RETURNS: Integer
AUTHOR: Iouri Kourilov
Modified and commented by Habib Islam
---------------------------------------------------------------------------*/
uint32_t return_to_main(void)
{
/* If the number of received characters by usart is non-zero */
if(g_usart2_read_count)
```

```
{
/* If the received character is the ESC character */
if(g_usart2_read_buffer[g_usart2_read_count - 1] == ESC)
{
/* Clear the read buffer */
memset(g_usart2_read_buffer, 0, 256);
/* Reset the read count to 0 */
g_usart2_read_count = 0;
/* Return to main menu */
return 1;
}
else return 0;
}
}
/*-------------------------------------------------------------------------------
FUNCTION: void USART2_Init(void)
DESCRIPTION: Configures and Initializes USART2 using a configurable interrupt
PARAMETERS: None
RETURNS: None
AUTHOR: Habib Islam
-------------------------------------------------------------------------------*/
void USART2_Init(void)
{
/* Enable GPIOA clock by setting GPIOAEN (Bit[1]) in RCC_AHB1ENR register */
RCC->AHB1ENR |= 1 << 0; // RCC->AHB1ENR |= RCC_AHB1ENR_GPIOAEN;
/* Enable USART2 peripheral clock by setting USART2EN (Bit[17]) in RCC_APB1ENR register */
RCC->APB1ENR |= 1 << 17; // RCC->APB1ENR |= RCC_APB1ENR_USART2EN;
/* Clear MODER2 (Bits[5:4]) and MODER3 (Bits[7:6]) in GPIOA MODER register */
GPIOA->MODER &= ~(0xF << 4); // GPIOA->MODER &= ~(GPIO_MODER_MODE2|GPIO_MODER_MODE3);
/* Enable alternate function for PA2 and PA3 by writing 0b10 to MODER2 (Bits[5:4])
and MODER3 (Bits[7:6]) in GPIOA MODER register */
GPIOA->MODER |= (0xA << 4); // GPIOA->MODER |= (GPIO_MODER_MODE2_1|GPIO_MODER_MODE3_1);
/* Clear AFRL[2] (Bits[11:8]) and AFRL[3] (Bits[15:12]) in GPIO_AFRL register */
GPIOA->AFR[0] &= ~(0xFF<<8); // GPIOA->AFR[0] &= ~(GPIO_AFRL_AFSEL2|GPIO_AFRL_AFSEL3);
/* Select AF7 for PA2 and PA3 by writing 0b0111 to AFRL[2] (Bits[11:8])
and AFRL[3] (Bits[15:12]) in GPIO_AFRL register */
GPIOA->AFR[0] |= 0x77 << 8;
/* Disable USART2 by clearing UE (Bit[13]) of USART2_CR1 register */
USART2->CR1 &= ~(1 << 13); // USART2->CR1 &= ~USART_CR1_UE;
/* Configuer for word length of 8 data bits and 1 start bit
by clearing M (Bit[12]) of USART2_CR1 register */
USART2->CR1 &= ~(1 << 12); // USAR2->CR1 &= ~USART_CR1_M
/* Configure for 16x oversampling by clearing OVER8 (Bit[15]) of USART2_CR1 register */
USART2->CR1 &= ~(1 << 15); // USAR2->CR1 &= ~USART_CR1_OVER8
/* Disable parity control by clearing PCE (Bit[10]) of USART2_CR1 register */
USART2->CR1 &= ~(1 << 10); // USAR2->CR1 &= ~USART_CR1_PCE
/* Configure for 1 stop bit by clearing STOP (Bits[13:12]) of USART2_CR2 register */
USART2->CR2 &= ~(3 << 12); // USAR2->CR2 &= ~USART_CR2_STOP
/* Configure for no flow control by clearing CTSE (Bit[9]) and RTSE (Bit[8]) of USART2_CR3 register */
USART2->CR3 &= ~(3 << 8); //USAR2->CR3 &= ~(USART_CR3_RTSE_Msk|USART_CR3_CTSE_Msk)
/* Configure for 9600 baud rate by writing 273 to Bits[15:4] and 7 to Bits[3:0] in USART2_BRR */
USART2->BRR = (273 << 4) | 7;
/* Enable USART2 transmitter and receiver by setting TE (Bit[3])
and RE (Bit[2]) of USART2_CR1 register, respectively */
USART2->CR1 |= 3 << 2; // USART2->CR1 |= (USART_CR1_TE|USART_CR1_RE);
/* Enable USART2 by setting UE (Bit[13]) in USART2_CR1 */
USART2->CR1 |= 1 << 13; // USART2->CR1 |= USART_CR1_UE;
/* Enable RXNE interrupt by setting RXNE (Bit[5] in USART2_CR1 register */
USART2->CR1 |= 1 << 5; // USART2->CR1 |= USART_CR1_RXNE;
/* Set the interrupt priority to highest */
NVIC_SetPriority(38, 0); // NVIC_SetPriority(USART2_IRQn, 0);
/* Enable NVIC level interrupt for USART2 */
NVIC_EnableIRQ(38); // NVIC_EnableIRQ(USART2_IRQn);
}
```