GEBZE TECHNICAL UNIVERSITY

ELECTRONIC ENGINEERING

ELEC334

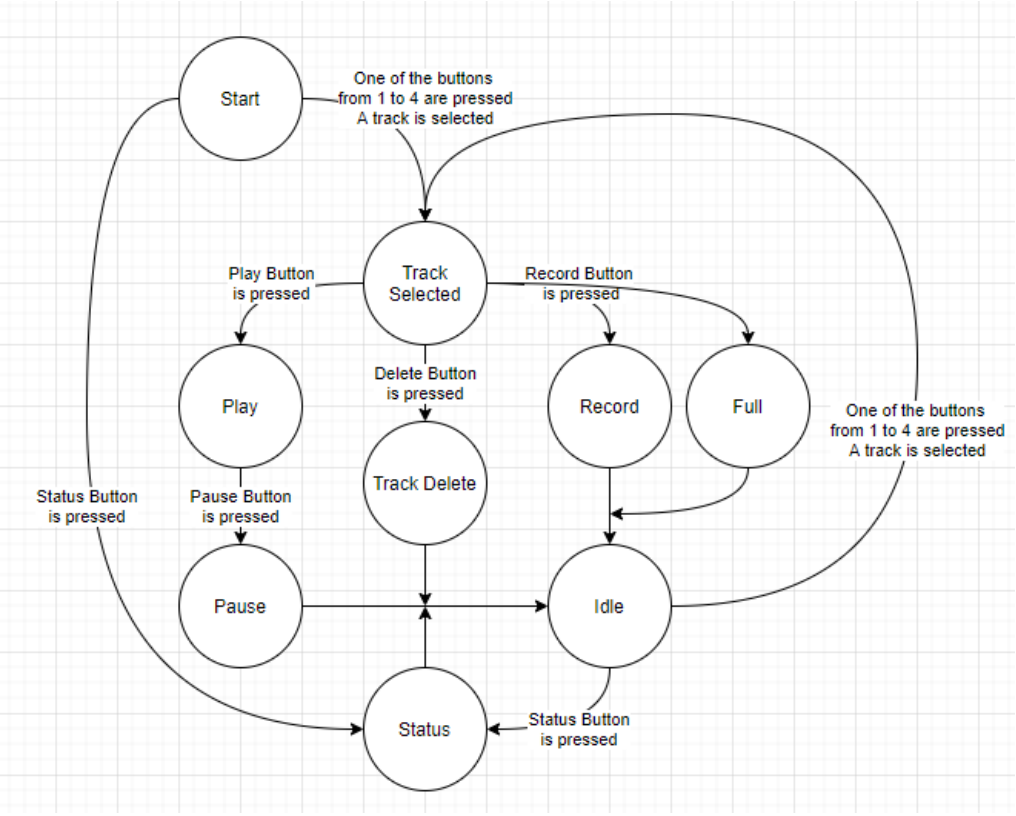
MICROPROCESSORS

Project 3 Report

Prepared by
171024086 Berat KIZILARMUT

1. Introduction

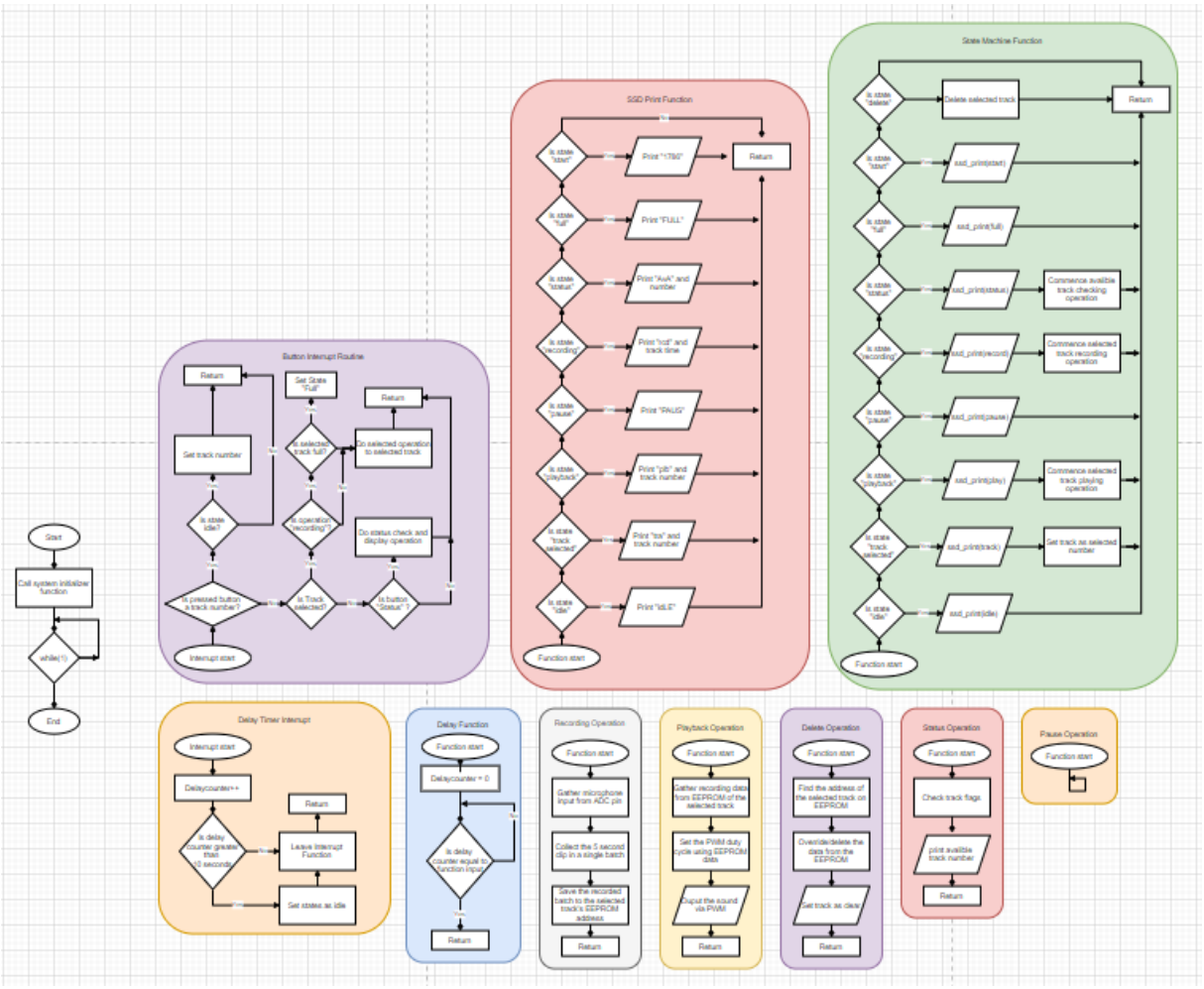
In this project I will design a digital voice recorder from the ground up without any additional pre-made libraries, HAL, and functions. In this report firstly I will be explaining how I will be approaching the project and details I have in mind, next I will create a task list of supposed functions I need to design, I may need to add other functions to supports these theorical function list. I will be providing flowcharts, hardware diagrams, state transition diagrams and photographs of the project.

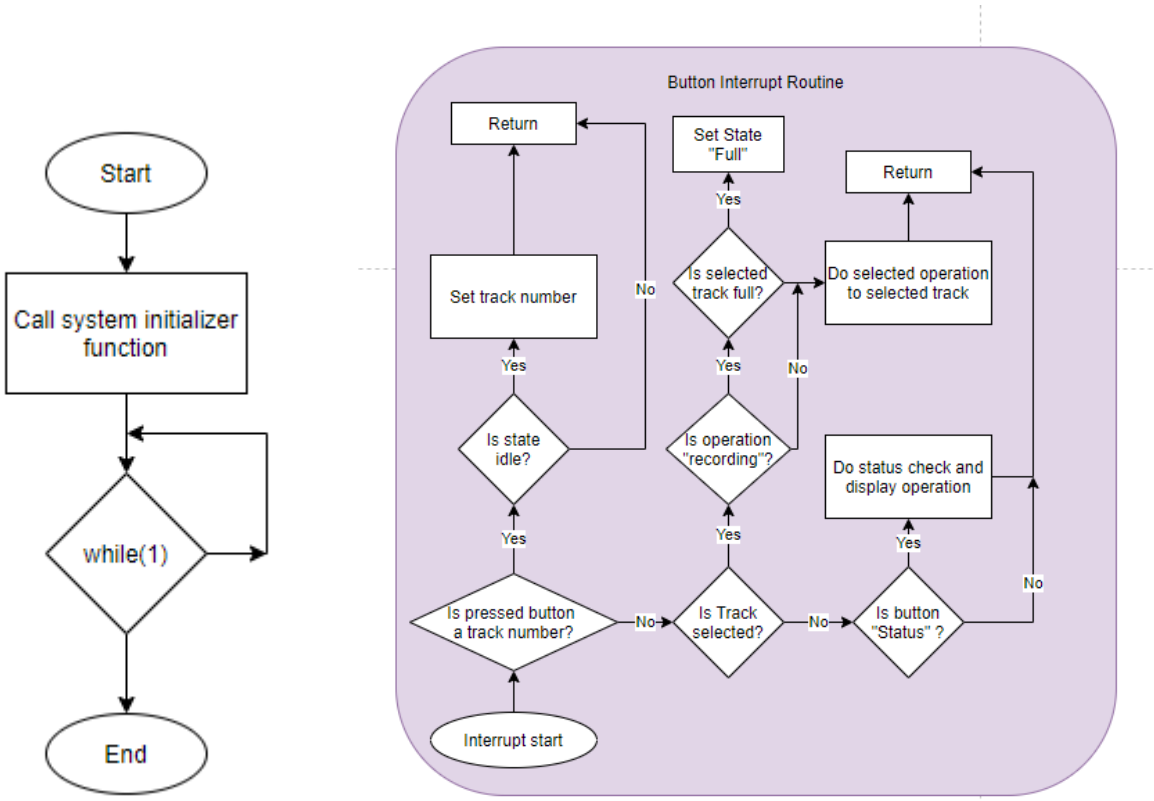


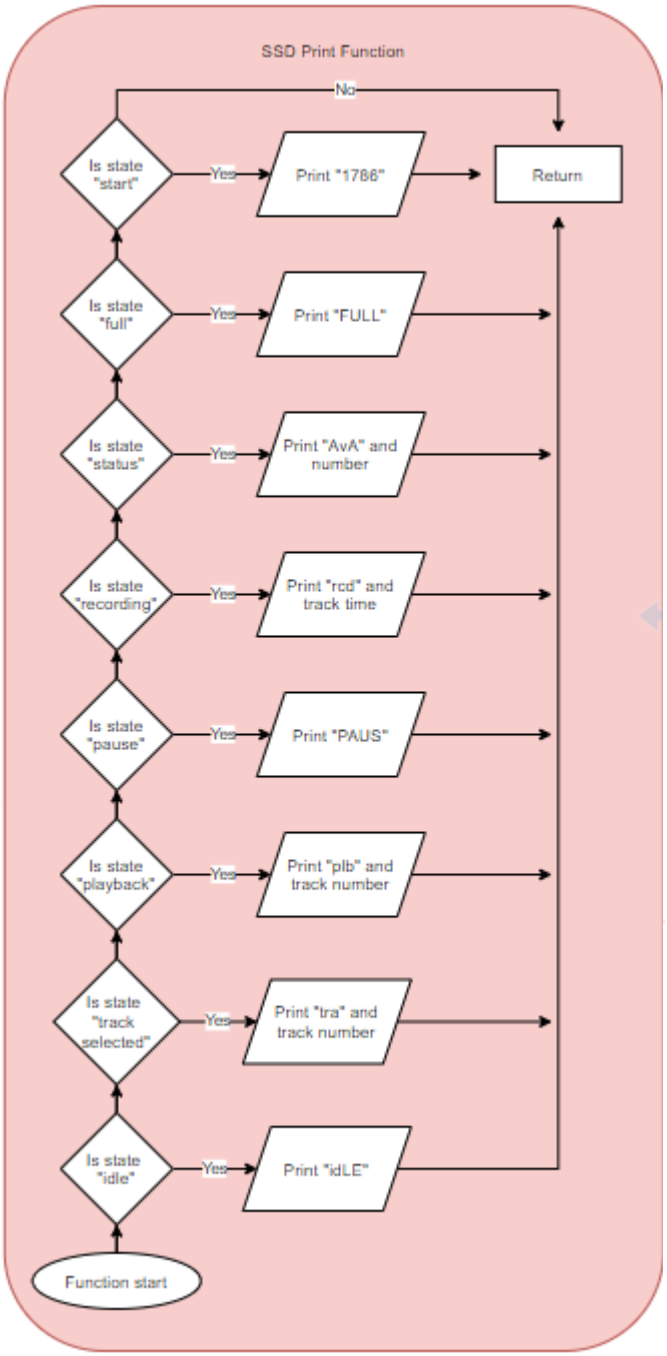
2. Approach and Details Explained

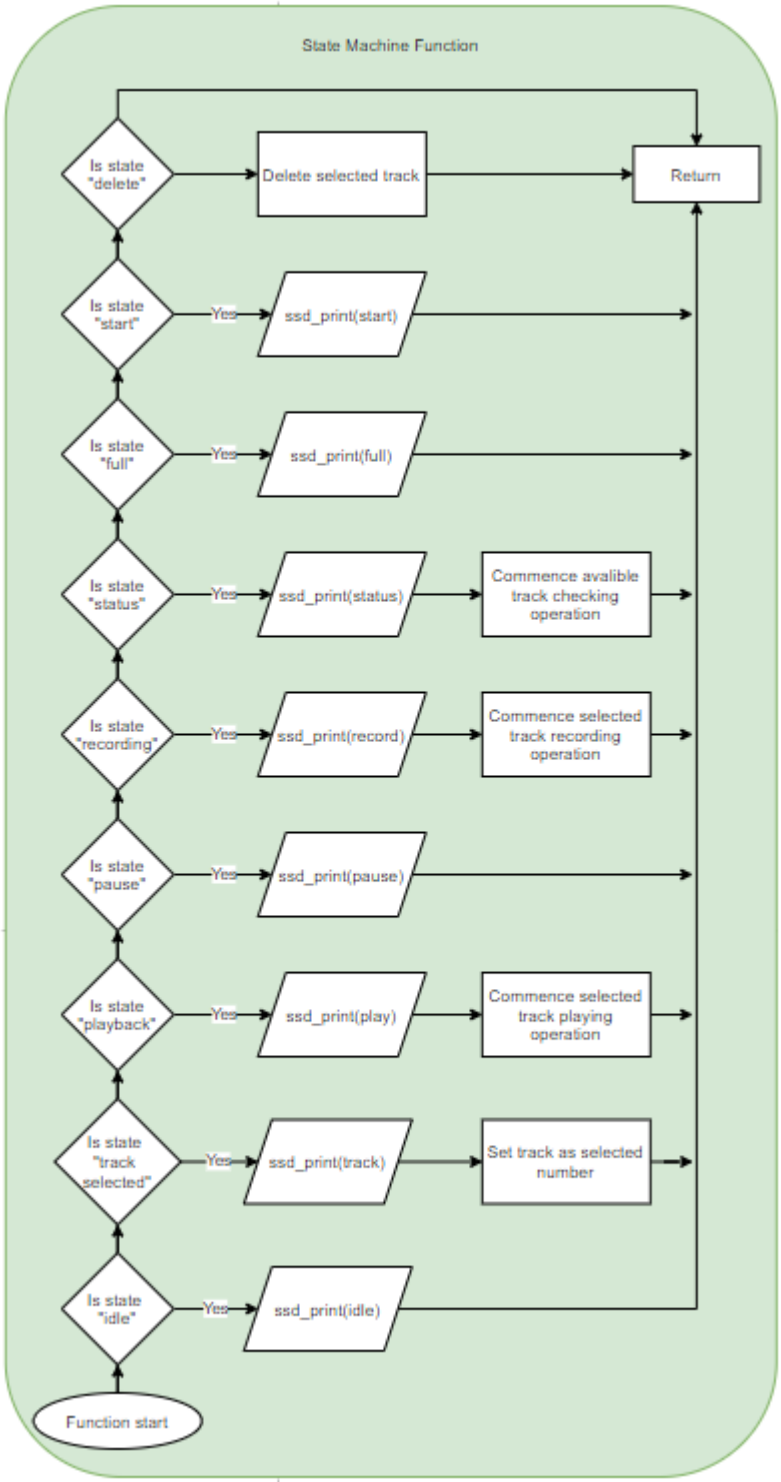
Primitive state diagram is given to the left. This is how I imagine the states of the digital voice recorder after reading the project task. I will be creating state enumerators that will be set at keypad input functions. I might need to implement a previous state variable. If the state is either start or idle, only track selection buttons will do something, otherwise button presses will simply change nothing in the variables.

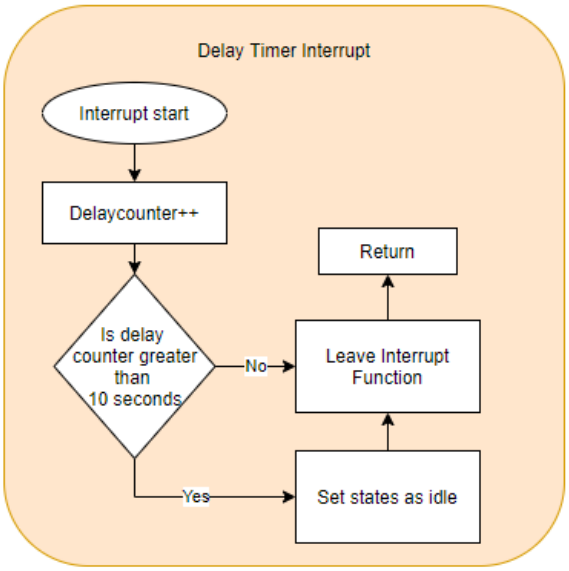
Only user interface available will be a 4x4 Keypad. There might be not enough pins to connect to the MCU so I might only use 2 rows and 4 columns of the keypad. Utilizing the first 2 rows, number 1 2 3 4 will choose track slots. A Button will be the record button, B Button will be the play button, 5 Button will be the status button and lastly 6 Button will be the track delete button. The way I've imagined the workings of the recorder, without choosing the track any of these function buttons will be doing nothing, only after a track select these become useful. Depending on the selected track, there will be separate "selected track slot is full" state that will display the word full on the SSD. When a track recording is ongoing, SSD will count up with the recording timing for example "RCD1" "RCD2" etc. When a track is played, SSD will display playback and track number. Status check will result in available remaining track slot amount being displayed.

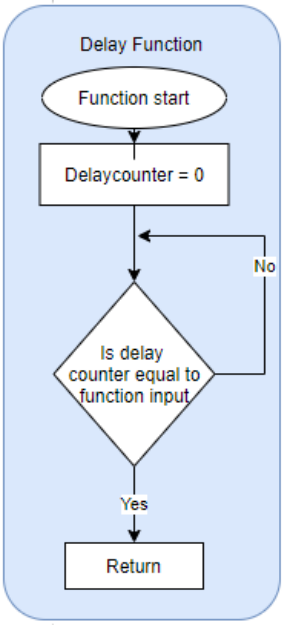

Start
Call system initializer function
while(1)
End
Button Interrupt Routine
Return
Set State "Full"
Return
Set track number
Is selected track "full"?
Do selected operation to selected track
Is state idle?
Is operation "recording"?
Do status check and display operation
Is pressed button a track number?
Is Track selected?
Is button "Status" ?
Interrupt start
SSD Print Function
Is state "start"
Print "-1760"
Return
Is state "full"
Print "FULL"
Is state "status"
Print "AvA" and number
Is state "recording"
Print "rcd" and track time
Is state "pause"
Print "PAUS"
Is state "playback"
Print "plb" and track number
Is state "track selected"
Print "tra" and track number
Is state "idle"
Print "idLE"
Function start
State Machine Function
Is state "delete"
Delete selected track
Return
Is state "start"
ssd_print(start)
Is state "full"
ssd_print(full)
Is state "status"
ssd_print(status)
Commence availble track checking operation
Is state "recording"
ssd_print(record)
Commence selected track recording operation
Is state "pause"
ssd_print(pause)
Is state "playback"
ssd_print(play)
Commence selected track playing operation
Is state "track selected"
ssd_print(track)
Set track as selected number
Is state "idle"
ssd_print(idle)
Function start
Delay Timer Interrupt
Interrupt start
Delaycounter++
Is delay counter greater than 10 seconds
Return
Leave Interrupt Function
Set states as idle
Delay Function
Function start
Delaycounter = 0
Is delay counter equal to function input
Return
Recording Operation
Function start
Gather microphone input from ADC pin
Collect the 5 second clip in a single batch
Save the recorded batch to the selected track's EEPROM address
Return
Playback Operation
Function start
Gather recording data from EEPROM of the selected track
Set the PWM duty cycle using EEPROM data
Ouput the sound via PWM
Return
Delete Operation
Function start
Find the address of the selected track on EEPROM
Override/delete the data from the EEPROM
Set track as clear
Return
Status Operation
Function start
Check track flags
print availble track number
Return
Pause Operation
Function start

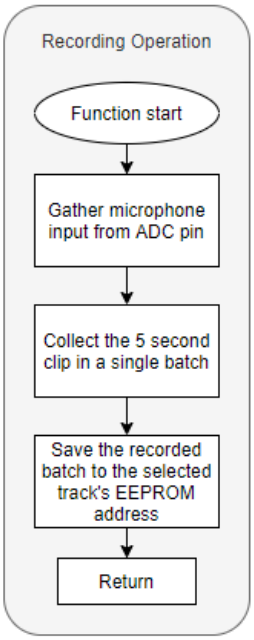

Start
Call system initializer function
while(1)
End
Button Interrupt Routine
Return
Set State "Full"
Return
Set track number
Is selected track full?
Do selected operation to selected track
Is state idle?
Is operation "recording"?
Do status check and display operation
Is pressed button a track number?
Is Track selected?
Is button "Status" ?
Interrupt start
Yes
No

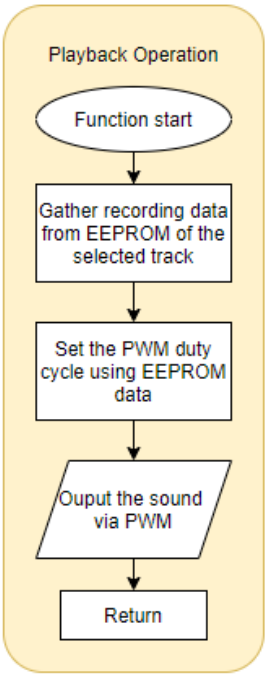

SSD Print Function
No
Is state "start"
Yes
Print "1786"
Return
Is state "full"
Yes
Print "FULL"
Is state "status"
Yes
Print "AvA" and number
Is state "recording"
Yes
Print "rcd" and track time
Is state "pause"
Yes
Print "PAUS"
Is state "playback"
Yes
Print "plb" and track number
Is state "track selected"
Yes
Print "tra" and track number
Is state "idle"
Yes
Print "idLE"
Function start

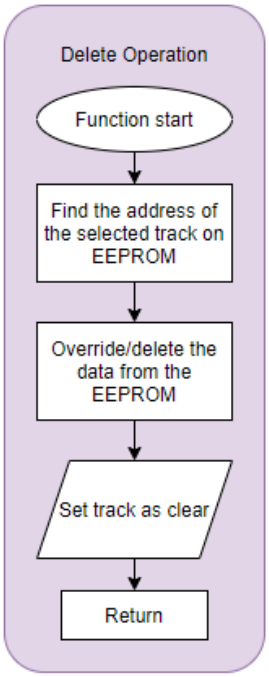

State Machine Function
Is state "delete"
Delete selected track
Return
Is state "start"
Yes
ssd_print(start)
Is state "full"
Yes
ssd_print(full)
Is state "status"
Yes
ssd_print(status)
Commence avalible track checking operation
Is state "recording"
Yes
ssd_print(record)
Commence selected track recording operation
Is state "pause"
Yes
ssd_print(pause)
Is state "playback"
Yes
ssd_print(play)
Commence selected track playing operation
Is state "track selected"
Yes
ssd_print(track)
Set track as selected number
Is state "idle"
Yes
ssd_print(idle)
Function start

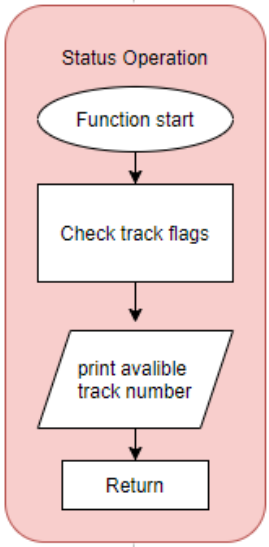

Delay Timer Interrupt
Interrupt start
Delaycounter++
Is delay counter greater than 10 seconds
No
Leave Interrupt Function
Return
Yes
Set states as idle

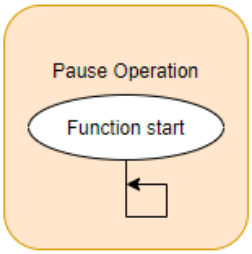

Delay Function
Function start
Delaycounter = 0
Is delay counter equal to function input
No
Yes
Return


Recording Operation
Function start
Gather microphone input from ADC pin
Collect the 5 second clip in a single batch
Save the recorded batch to the selected track's EEPROM address
Return


Playback Operation
Function start
Gather recording data from EEPROM of the selected track
Set the PWM duty cycle using EEPROM data
Ouput the sound via PWM
Return


Delete Operation
Function start
Find the address of the selected track on EEPROM
Override/delete the data from the EEPROM
Set track as clear
Return


Status Operation
Function start
Check track flags
print avalible track number
Return


Pause Operation
Function start

3. Task List

Created this task list to list what I think I have to create to complete this project.

- Initialize function ✓: A function that setup the board, GPIOs, interrupts, timers, I2C, PWM and ADC.
- Keypad input decipher function ✓: A function that will test and find what button is pressed, which row and which column.
- State machine ✓: A state machine that will control the menu functions and device functionality.
- SSD printing function ✓: A function that will display given numbers and display current state for example idle.
- Recording functions X: A function that will take the input from the microphone via ADC and record it to the EEPROM's with a chosen sample rate.
- Playback function X: A function that will read the data written in the EEPROM's and use PWM duty cycle to play it through the speakers.
- Delete function X: A function that will delete the data written in the EEPROM's depending on the sample choice and change the flags.

4. Explaining the Functions and Modules

In this section I will be explaining each of my functions, what I've tried to achieve and how I achieved that.

4.1 Initialization Function

This function initializes everything. Firstly, I clock the GPIOA and GPIOB peripherals. After the peripherals I start setting the 4x4 Keypad up. As a side note I like to use binary to decimal converters to complete MODER and ODR operations. Rather than setting the ODR or MODER's by pin by pin, I write the desired pin order in binary and convert it to decimal and set all the pins with a single line wherever I can. I've set the GPIOA MODER and PUPDR register using such technique however It was much easier to set the EXTICR's of the pins independent. For the keypad I used PA8 to PA11 as input columns and set an interrupt routine to these pins. I used PA12 and PA15 as my keypad rows. I didn't use all the rows because I only need 8 buttons and board was running out of external pins and the whole keypad was not necessary.

I've utilized the same SSD pins and layout I've used on prior labs which was PB0 to PB6 as A B C D E F and G pins of the SSD and PA4 to PA7 as D1 D2 D3 and D4 pins of the SSD. Disconnected the digit point pin because it was not needed on this project. Implemented a general-purpose timer, TIM1, to utilize in delay function. Timer interrupt priority is set as highest.

After that I've set up the ADC. Firstly, PA0 pin is set as analog mode on MODER. Enabled ADC Module clock, cleared ADC Control Register, cleared ADC Configurator Register and enabled voltage regulator. After enabling voltage regulator, we must wait to give time to voltage to stabilize. After voltage regulation, calibration is initialized. Conversation specifications are set and setup is complete.

PWM is required to get sound output from the recorder, utilized pin PB9 on PWM operations. Set pins PB9 as alternate function 2, enabled timer 17 to use on PWM. Set timer prescaler, auto reload, capture compare and break time registers.

I2C is required to write and read from the pair of EEPROM modules connected. I've utilized pin PB7 as SDA, pin PB8 as SCL on this project. Set both of these pins as alternate function, open-drain mode, and chose AF6 as their alternate function. Enabled the error interrupt. Set SCL high and low periods. Set the data hold and data setup timings. Enabled peripheral. Enabled the I2C Interrupt Handler.

4.2 Keypad Functions and Interrupts

Created several sub functions to use the keypad. Firstly, because I've used pins PA8 to PA11 as column pins for the keypad, I used external interrupt handler EXTI4_15_IRQHandler. Utilized only first two rows of the device because this project only required 8 buttons total.

When an input from the keypad is detected we arrive at the interrupt handler. First function called in the interrupt handler is keypad_check, which firstly sets every row voltage as zero using clear_rows function then sets a row high and checks for column input. Using this function exact button is determined and the states are set depending on the button. Rows voltages are set as high before leaving this function using activate_rows function.

4.3 State Machine

This state machine runs continuously through the whole time MCU is active. I print the appropriate text messages to the SSD in this function. Also recording, deleting and playing operations are initiated here. Just after entering the function, I turn off the PWM output by setting the duty cycle as zero unless the current state is playback.

4.4 Printing Functions

I did an overhaul of my Seven Segment Display functions and changed how they operate because on my previous tasks my GPIOB ODR operations over rid the whole register, now I've changed it so they do not modify the whole ODR register Prior to my changes, I failed to use PWM on PB9 because my SSD functions modified PB9 too, after changing my SSD functions my PWM started working. SSD_Print function now has 2 inputs, first input is operation type which depends on the state, second input is the number input. Depending on the state choice, SSD can print decimal digits, operation informations and menu functions etc.

4.5 Recording/Playback/Delete Functions

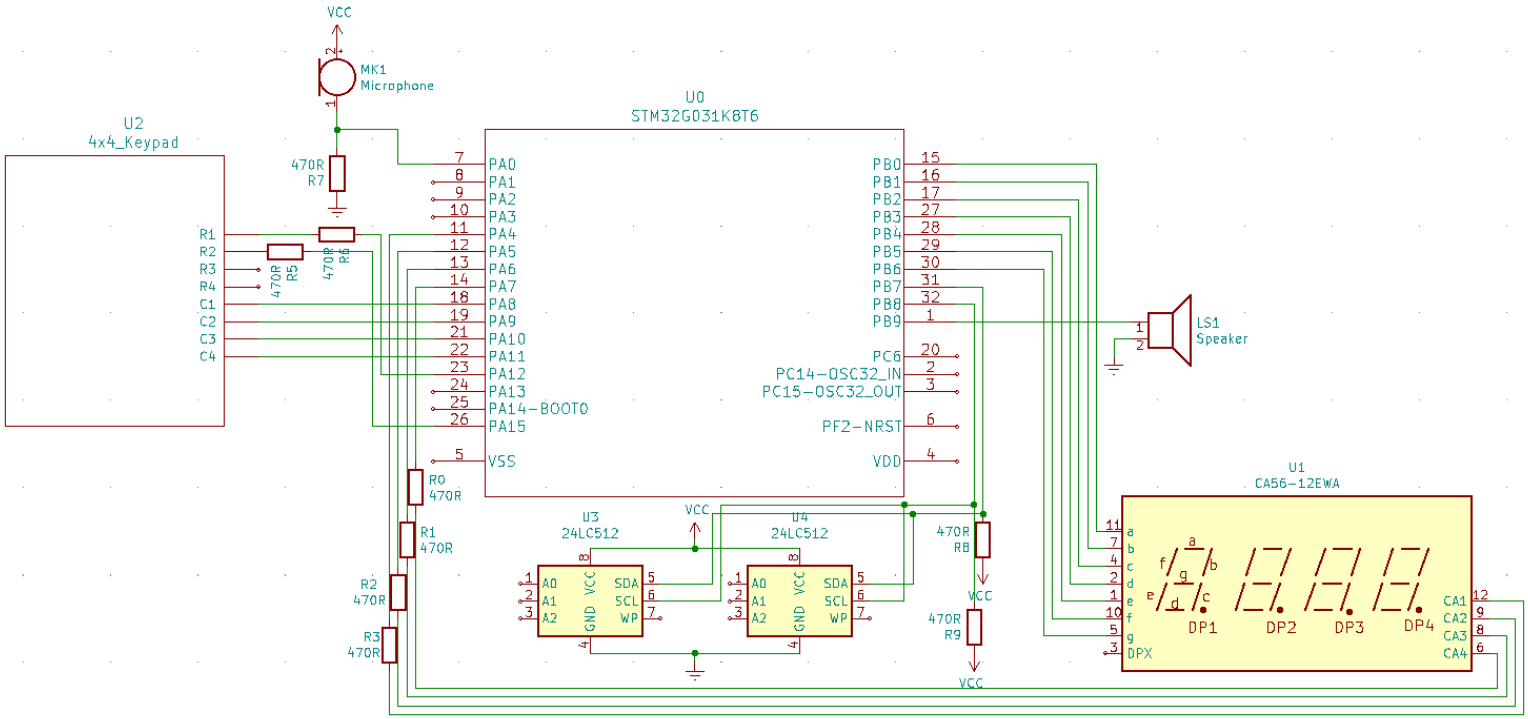
I could not set these up. I don't know how to, every research I made led me to HAL library used tutorials. I wanted to get information from the ADC in recording function, bunch these up in a buffer array and send it to the EEPROM's via I2C in a single operation, not continuous. I only could find I2C setup in lessons, I could not understand how to use in this situation.

4.6 Other Functions

Set-up a general-purpose timer, TIM1, on system initializer to use on the delay operations. Created a delay_ms function that delays for the given integer millisecond amount. This function utilizes the fact that the timer interrupt handler that I've set up continuously increases a global variable, delaycount, and since I've set the timer prescaler as 16000 this corresponds for a millisecond. When we call the function delay_ms, function resets the delaycount value to zero and checks continuously for the value to reach the required amount. Using timer, after no button presses for 10 seconds the recorder returns to idling.

Implemented clearing and activating functions for keypad. Implemented clearing function for seven segment display.

5. Hardware



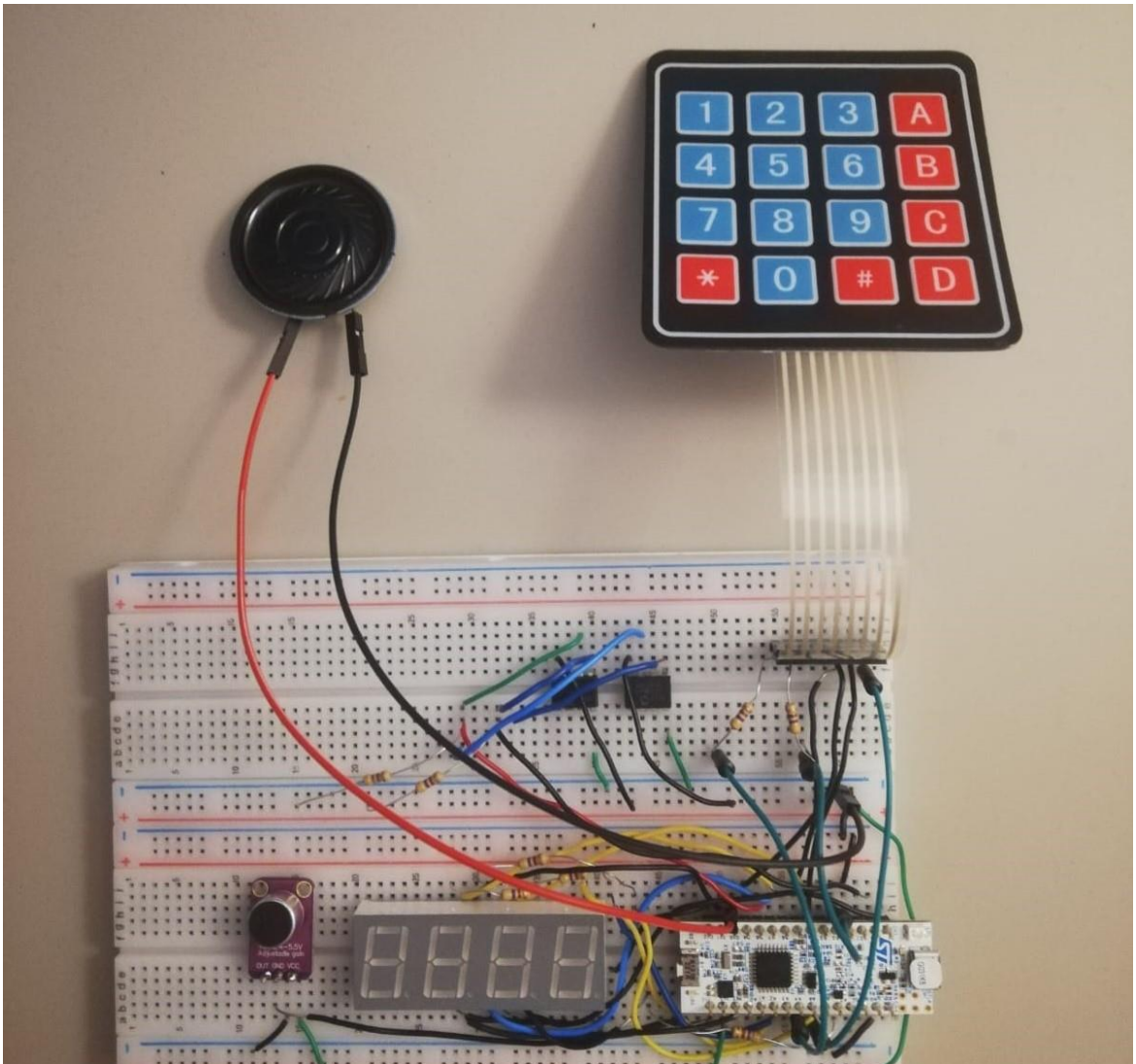
Breadboard is set by connecting 4-Digit Seven Segment Display and 4x4 Keypad to the breadboard and connecting the pre-determined GPIO pins. Connected the speaker to PB9 pin. Connected the EEPROM's, using PB7 as SDA, PB8 as SCL. Connected the microphone module with integrated amplifier to PA0. Used 8 resistors to pull-up or pull-down, resistors rated 470 ohms to be exact, are used to limit the currents flowing through the SSD and Keypad. Since on both of these components have some paths that connect with each other serially, I've used only one current limiting resistor per path. Resistors are connected to the row pins of the keypad and the digit pins of the SSD. While debugging and testing, I sometimes used a jumper to connect between column and row pins of the keypad rather than actually pressing the keypad because keypad inputs sometimes had some irregularities, however I am not sure if this is my fault or the components fault, I did not get any feedback from my previous uses of this component.

My speaker is working, I can get desired outputs via PWM. I could not get any input from microphone, I've tested it by only using 3.3 V and GND from the MCU and connected the speaker directly to output of the microphone, still could not get any output from it. I've soldered the connectors to the module and confirmed continuity via multimeter.

Total component and cost list given below;

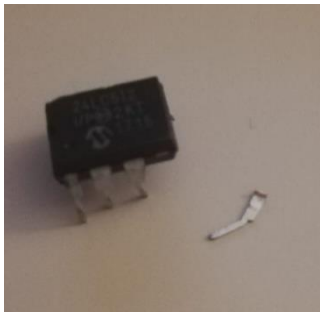
#	Part Name	Link	Amount	Unit Price	Total Price
1	STM32 G031K8T Development Board	https://www.empastore.com/stm32-islemci-kiti-nucleo-g031k8	1	135.41 TL	135.41 TL
2	4-Digit Seven Segment Display Common Anode	https://www.direnc.net/14mm-4lu-anot-display	1	7.21 TL	7.21TL
3	470 Ohm Resistor	https://www.robotistan.com/14w-470r-direnc-paketi-10-adet	8	0.046 TL	0.37 TL
4	MAX4466 Microphone Module	https://www.direnc.net/max4466-elektret-mikrofon-modulu	1	18.26 TL	18.26 TL
5	24LC512 EEPROM	https://www.direnc.net/24lc512--64kx8-i2c2-wire-serial-eeprom	2	9.82 TL	19.64 TL
6	8R 0.5W Speaker	https://www.direnc.net/8r-05w-83db-36x5mm-hoparlor	1	6.70 TL	6.70
4	4x4 Keypad	https://www.direnc.net/4x4-mebram-tus-takimi	1	5.41 TL	5.41 TL
6	Breadboard	https://www.direnc.net/tekli-breadboard	2	8.56 TL	17.12 TL
7	Jumper Cable M-M	https://www.direnc.net/40-adet-erkek-erkek-jumper-20cm	20	0.093 TL	1.86 TL
				Total	211.98 TL

Breadboard layout picture given below;



6. Conclusion

In this project I've learned a lot about PWM and ADC operations even though I could not utilize them altogether. I have utilized most of what I've learned from previous labs and homework's and had to add new skills on top of that. I had to correct and redesign my previous function, for example my old SSD printing functions cause my PWM to not work. Altogether this project was the most frustrating one for me. On other projects I had resources necessary but any fault I made was my own error but in this project and last few labs I felt like I could not find enough related resource to completed required tasks. I do not mean any disrespect.



6.1 Problems, Failures and Improvement Ideas

I still had problems leftover from my previous labs and projects, but I did not get any feedback from these. I did not receive my notes from those said labs. I don't know if I keep using broken functions and keep doing the same errors. On top of that I've purchased a new pair of breadboard because I previously used only one, and one board would not have enough space to complete this task. However, as you know new breadboards have very stiff pins and this resulted in a broken pins on one of my EEPROM's. In my test, I connected microphone output directly to the speaker but could not hear anything. I have changed the gain pot on the back of the microphone module and soldered it to the pins and tried that failed as well.

7. References

1. RM0444 Reference Manual, STM32G0x1 advanced Arm®-based 32-bit MCUs, ST Microelectronics
2. 24LC512 Datasheet, Microchip

8. Code

```
/*  Project 3
    main.c
    Digital Voice Recorder
    Berat Kizilarmut, 171024086 */
#include "stm32g0xx.h"
volatile float time, ava,sample, print_value = 0; /* Creating the global variab
les*/
volatile uint32_t buffer[1600], buffer_counter = 0; /* Microphone input, EEPROM
 write buffers */
volatile uint32_t delaycount = 0; /* Global delay counter */

enum state_enum /* Setting the state enum */
{
    idle = 0, track = 1, play = 2, pause = 3, record = 4, delete = 5, status =
6, full = 7, start = 8
```

```
};
enum state_enum state; /* State object */

void delay_ms(volatile uint32_t ms) /* Set as volatile to prevent optimization
*/
{
    delaycount = 0; /* Set counter as zero */
    while(1) /* Continuously checks the counter */
    {
        if(delaycount == ms)
        return;
    }
}

void ssd_clear(void) /* Clear the display */
{
    GPIOA->ODR &= ~(15U << 4); /* Clear Digits */
    GPIOB->ODR &= ~(127U); /* Set A B C ... as zero */
}

void ssd_printdigit(uint32_t digit) /* Prints the values depending on the numbe
r */
{
    switch(digit)
    {
        case 0: /* Printing 0 */
            GPIOB->ODR |= (0x40U);
            break;

        case 1: /* Printing 1 */
            GPIOB->ODR |= (0x79U);
            break;

        case 2: /* Printing 2 */
            GPIOB->ODR |= (0x24U);
            break;

        case 3: /* Printing 3 */
            GPIOB->ODR |= (0x30U);
            break;

        case 4: /* Printing 4 */
            GPIOB->ODR |= (0x19U);
            break;

        case 5: /* Printing 5 */
            GPIOB->ODR |= (0x12U);
            break;

        case 6: /* Printing 6 */
            GPIOB->ODR |= (0x2U);
            break;

        case 7: /* Printing 7 */
            GPIOB->ODR |= (0x78U);
            break;

        case 8: /* Printing 8 */
            GPIOB->ODR |= (0x0U);
```

```
            break;

        case 9: /* Printing 9 */
            GPIOB->ODR |= (0x10U);
            break;
    }
}

void ssd_print(uint32_t op, float value)
{
    uint32_t d1=0, d2=0, d3=0, d4=0;
    switch(op) /* Printing operation depending on the operational input */
    {
        case 0: /* Printing idLE */
            ssd_clear();
            /* Print idLE (Idle) on SSD */
            GPIOB->ODR |= (123U); /* (123U = 0b1111011) for i */
            GPIOA->ODR |= (1U << 4); /* Set D1 High */
            ssd_clear();
            GPIOB->ODR |= (33U); /* (33U = 0b0100001) for d */
            GPIOA->ODR |= (1U << 5); /* Set D2 High */
            ssd_clear();
            GPIOB->ODR |= (71U); /* (71U = 0b1000111) for L */
            GPIOA->ODR |= (1U << 6); /* Set D3 High */
            ssd_clear();
            GPIOB->ODR |= (6U); /* (6U = 0b0000110) for E */
            GPIOA->ODR |= (1U << 7); /* Set D4 High */
            ssd_clear();
            break;

        case 1: /* Printing Track Number */
            ssd_clear();
            /* Print TrA (Track) on SSD */
            GPIOB->ODR |= (78U); /* (78U = 0b1001110) for T */
            GPIOA->ODR |= (1U << 4); /* Set D1 High */
            ssd_clear();
            GPIOB->ODR |= (47U); /* (47U = 0b0101111) for r */
            GPIOA->ODR |= (1U << 5); /* Set D2 High */
            ssd_clear();
            GPIOB->ODR |= (8U); /* (8U = 0b0001000) for A */
            GPIOA->ODR |= (1U << 6); /* Set D3 High */
            ssd_clear();
            ssd_printdigit(value); /* Print Value to D4 */
            GPIOA->ODR |= (1U << 7); /* Set D4 High */
            ssd_clear();
            break;

        case 2: /* Printing Played back track */
            ssd_clear();
            /* Print PLb (Playback) on SSD */
            GPIOB->ODR |= (12U); /* (12U = 0b0001100) for P */
            GPIOA->ODR |= (1U << 4); /* Set D1 High */
            ssd_clear();
            GPIOB->ODR |= (71U); /* (71U = 0b1000111) for L */
            GPIOA->ODR |= (1U << 5); /* Set D2 High */
            ssd_clear();
            GPIOB->ODR |= (3U); /* (3U = 0b0000011) for b */
            GPIOA->ODR |= (1U << 6); /* Set D3 High */
            ssd_clear();
```

```
            ssd_printdigit(value); /* Print Value to D4 */
            GPIOA->ODR |= (1U << 7); /* Set D4 High */
            ssd_clear();
            break;

        case 3: /* Printing Pause */
            ssd_clear();
            /* Print PAUS (Pause) on SSD */
            GPIOB->ODR |= (12U); /* (12U = 0b0001100) for P */
            GPIOA->ODR |= (1U << 4); /* Set D1 High */
            ssd_clear();
            GPIOB->ODR |= (8U); /* (8U = 0b0001000) for A */
            GPIOA->ODR |= (1U << 5); /* Set D2 High */
            ssd_clear();
            GPIOB->ODR |= (65U); /* (65U = 0b1000001) for U */
            GPIOA->ODR |= (1U << 6); /* Set D3 High */
            ssd_clear();
            GPIOB->ODR |= (18U); /* (18U = 0b0010010) for S */
            GPIOA->ODR |= (1U << 7); /* Set D4 High */
            ssd_clear();
            break;

        case 4: /* Printing Recording */
            ssd_clear();
            /* Print rcd (Record) on SSD */
            GPIOB->ODR |= (47U); /* (47U = 0b0101111) for r */
            GPIOA->ODR |= (1U << 4); /* Set D1 High */
            ssd_clear();
            GPIOB->ODR |= (39U); /* (39U = 0b0100111) for c */
            GPIOA->ODR |= (1U << 5); /* Set D2 High */
            ssd_clear();
            GPIOB->ODR |= (33U); /* (33U = 0b0100001) for d */
            GPIOA->ODR |= (1U << 6); /* Set D3 High */
            ssd_clear();
            ssd_printdigit(value); /* Print Value to D4 */
            GPIOA->ODR |= (1U << 7); /* Set D4 High */
            ssd_clear();
            break;

        case 5: /* Dont print anything for delete */
            break;

        case 6: /* Printing Status availability */
            ssd_clear();
            /* Print AvA (Available) on SSD */
            GPIOB->ODR |= (8U); /* (8U = 0b0001000) for A */
            GPIOA->ODR |= (1U << 4); /* Set D1 High */
            ssd_clear();
            GPIOB->ODR |= (99U); /* (99U = 0b1100011) for v */
            GPIOA->ODR |= (1U << 5); /* Set D2 High */
            ssd_clear();
            GPIOB->ODR |= (8U); /* (8U = 0b0001000) for A */
            GPIOA->ODR |= (1U << 6); /* Set D3 High */
            ssd_clear();
            ssd_printdigit(value); /* Print Value to D4 */
            GPIOA->ODR |= (1U << 7); /* Set D4 High */
            ssd_clear();
            break;
```

```
        case 7: /* Printing Full */
            ssd_clear();
            /* Print FULL (Full) on SSD */
            GPIOB->ODR |= (14U); /* (14U = 0b0001110) for F */
            GPIOA->ODR |= (1U << 4); /* Set D1 High */
            ssd_clear();
            GPIOB->ODR |= (65U); /* (65U = 0b1000001) for U */
            GPIOA->ODR |= (1U << 5); /* Set D2 High */
            ssd_clear();
            GPIOB->ODR |= (71U); /* (71U = 0b1000111) for L */
            GPIOA->ODR |= (1U << 6); /* Set D3 High */
            ssd_clear();
            GPIOB->ODR |= (71U); /* (71U = 0b1000111) for L */
            GPIOA->ODR |= (1U << 7); /* Set D4 High */
            ssd_clear();
            break;

        case 8: /* Printing 4 digit ID number */
            ssd_clear();
            /* Print idLE (Idle) on SSD */
            GPIOB->ODR |= (0x79U); /* 1 */
            GPIOA->ODR |= (1U << 4); /* Set D1 High */
            ssd_clear();
            GPIOB->ODR |= (0x78U); /* 7 */
            GPIOA->ODR |= (1U << 5); /* Set D2 High */
            ssd_clear();
            GPIOB->ODR |= (0x0U); /* 8 */
            GPIOA->ODR |= (1U << 6); /* Set D3 High */
            ssd_clear();
            GPIOB->ODR |= (0x2U); /* 6 */
            GPIOA->ODR |= (1U << 7); /* Set D4 High */
            ssd_clear();
            break;
    }

}

void ADC_COMP_IRQHandler(void)
{   /* Analog to Digital Converter Comparator Handler */
    uint8_t microphone_input = (uint8_t) (ADC1->DR);
    buffer[buffer_counter] = microphone_input;
    ADC1->ISR |= (1U << 3); /* Clear pending status */
}

void op_play(void)
{   /* Selected track playback function */

}

void op_record(void)
{   /* Selected track recording function */

}

void op_status(void)
{   /* Track availibity check function */

}
```

```c
void op_delete(void)
{   /* Track deletion function */

}

void state_machine(void)
{   /* Continiously running state machine */
    if(state != play)
    TIM17->CCR1 = 0; /* Set output duty cycle as zero to silence */

    switch(state) /* Operations depending on the states */
    {
        case 0: /* Idle State*/
            ssd_print(idle,0);
            break;

        case 1: /* Track Selected State */
            ssd_print(track,sample);
            break;

        case 2: /* Track Playback State*/
            op_play();
            ssd_print(play,sample);
            break;

        case 3: /* Program Pause State */
            ssd_print(pause,0);
            break;

        case 4: /* Track Recording State */
            op_record();
            ssd_print(record,time);
            break;

        case 5: /* Track Deletion State */
            op_delete();
            ssd_print(delete,0);
            break;

        case 6: /* Program Status Check State */
            op_status();
            ssd_print(status,ava);
            break;

        case 7: /* Cannot record to track, track is full state */
            ssd_print(full,0);
            break;

        case 8: /* Starting State */
            ssd_print(start,0);
            break;
    }
}

void clear_rows(void)
{
    /* Setting all the rows as low */
    GPIOA->ODR &= ~(36864U); /* ~(36864U) = ~(1001000000000000) */
}
```

```
void activate_rows(void)
{
    /* Setting all the rows as high */
    GPIOA->ODR |= (36864U); /* 36864U = 1001000000000000 */
}

void keypad_check(void)
{   /* This function checks for the button row and column */
    if (state == start) /* We should not be in start state if we arrive here */
    state = idle;
    /* Setting Row 1 (PA12) as 1, others as zero to check */
    clear_rows();
    GPIOA->ODR |= (1U << 12);
    if((GPIOA->IDR >> 8) & (1U)) /* Checking CR1, PA8 */
    { /* C1 and R1 means button "1" */
        if(state == idle)
        {
            sample = 1;
            state = track;
        }
    }

    if((GPIOA->IDR >> 9) & (1U)) /* Checking CR2, PA9 */
    { /* C2 and R1 means button "2" */
        if(state == idle)
        {
            sample = 2;
            state = track;
        }
    }

    if((GPIOA->IDR >> 10) & (1U)) /* Checking CR3, PA10 */
    { /* C3 and R1 means button "3" */
        if(state == idle)
        {
            sample = 3;
            state = track;
        }
    }

    if((GPIOA->IDR >> 11) & (1U)) /* Checking CR4, PA11 */
    { /* C4 and R1 means button "A" */
        if(state == track)
        state = record;
    }

    delay_ms(50); /* Waiting 50 ms to stabilze*/
    /* Setting Row 2 (PA15) as 1, others as zero to check */
    clear_rows();
    GPIOA->ODR |= (1U << 15);
    if((GPIOA->IDR >> 8) & (1U)) /* Checking CR1, PA8 */
    { /* C1 and R2 means button "4" */
        if(state == idle)
        {
            sample = 4;
            state = track;
        }
    }
```

```
    if((GPIOA->IDR >> 9) & (1U)) /* Checking CR2, PA9 */
    { /* C2 and R2 means button "5" */
        if(state == idle)
        state = status;
    }

    if((GPIOA->IDR >> 10) & (1U)) /* Checking CR3, PA10 */
    { /* C3 and R2 means button "6" */
        if(state == track)
        state = delete;
    }

    if((GPIOA->IDR >> 11) & (1U)) /* Checking CR4, PA11 */
    { /* C4 and R2 means button "B" */
        if(state == track)
        state = play;

        if(state == play)
        state = pause;
    }
    activate_rows(); /* Setting the rows high before leaving the check function
*/
}



void EXTI4_15_IRQHandler (void)
{
    delaycount = 0; /* Setting up for timeout */
    keypad_check(); /* Get button information */
    delay_ms(500); /* Delay so function does not run continiously */
    EXTI->RPR1 |= (15U << 8); /* Clearing pending PA8 to PA11 */
}

void TIM1_BRK_UP_TRG_COM_IRQHandler (void) /* Configuring TIM1 */
{
    delaycount++;
    if(delaycount > 10000)
    state = idle;
    TIM1->SR &= ~(1U << 0); /* Clear pending status */
}

void system_initialize(void) /* Setting up the board and subsystems */
{   /* Enable GPIOA and GPIOB clock */
    RCC->IOPENR |= (3U); /* 3U = 11 */

    /* Setting TIM1 */
    RCC->APBENR2 |= (1U << 11); /* Enable TIM1 Clock */
    TIM1->CR1 = 0; /* Clearing the control register */
    TIM1->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM1->CNT = 0; /* Zero the counter */
    TIM1-
>PSC = 15999; /* Setting prescaler as 16000 to achieve a millisecond on my dela
y function */
    TIM1->ARR = 1;
    TIM1->DIER |= (1U << 0); /* Updating interrupt enabler */
    TIM1->CR1 |= (1U << 0); /* Enable TIM1 */
    NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 1); /* Setting priority */
```

```
    NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn); /* Enabling interrupt */

    /* Setting up the 4x4 Keypad */
    /* Setting up the external button interrupts */
    /* Setup PA8 to PA11 as input to use as columns on the keypad */
    GPIOA->MODER &= ~(255U << 8*2); /* ~(255U) = ~(11111111) */
    GPIOA->PUPDR &= ~(85U << 8*2); /* ~(85U) = ~(01010101) */
    GPIOA->PUPDR |= (170U << 8*2); /* 170U = 10101010 */
    /* Setting up the interrupt operation for columns */
    EXTI->RTSR1 |= (15U << 8); /* Setting the trigger as rising edge */
    EXTI->EXTICR[2] &= ~(1U << 8*0); /* Setting for PA8 */
    EXTI->EXTICR[2] &= ~(1U << 8*1); /* Setting for PA9 */
    EXTI->EXTICR[2] &= ~(1U << 8*2); /* Setting for PA10 */
    EXTI->EXTICR[2] &= ~(1U << 8*3); /* Setting for PA11 */
    EXTI->IMR1 |= (15U << 8); /* Interrupt mask register */
    EXTI-
>RPR1 |= (15U << 8); /* Rising edge pending register, Clearing pending PA8 to P
A11 */
    NVIC_SetPriority(EXTI4_15_IRQn, 3); /* Setting priority */
    NVIC_EnableIRQ(EXTI4_15_IRQn); /* Enabling the interrupt function */
    /* Setting up the pins PA12 and PA15 as output to use a rows on the keypad
*/
    GPIOA->MODER &= ~(3U << 12*2); /* Zero PA12 */
    GPIOA->MODER |= (1U << 12*2); /* Set first bit one on PA12 */
    GPIOA->MODER &= ~(3U << 15*2); /* Zero PA15 */
    GPIOA->MODER |= (1U << 15*2); /* Set first bit one on PA15 */
    /* Have to set the rows as high to detect an initial input */
    GPIOA->ODR |= (36864U); /* 36864U = 1001000000000000 */

    /* Setting up the Seven Segment Display*/
    /* Set PB0 to PB6 as output to use as A B C D E F G pins of the SSD */
    GPIOB->MODER &= ~(16383U); /* Setting first 16 bits as zero */
    GPIOB->MODER |= (5461U); /* Setting odd bits as one */
    /* Set PA4 to PA7 as output to use as D1 D2 D3 D4 pins of the SSD */
    GPIOA->MODER &= ~(65280U); /* Setting bits 8th to 16th as zero */
    GPIOA->MODER |= (21760U); /* Setting odd bits as one from 8th to 16th */

    /* Setting up the ADC */
    /* Setup PA0 as Analog mode */
    GPIOA->MODER |= (3U << 0);
    /* Setting up ADC1/0 for Analog input on PA0 */
    RCC->APBENR2 |= (1U << 20); /* Enable ADC Clock */
    ADC1->CR = 0; /* Clearing the control register */
    ADC1->CFGR1 = 0; /* Clearing the configurator register */
    ADC1->CR |= (1U << 28); /* Enabling voltage regulator */
    delay_ms(1000); /* Waiting for second to voltage to regulate */
    ADC1->CR |= (1U << 31); /* Initialize Calibration Operation */
    while(((ADC1->CR >> 31) & 1U)); /* Check Calibration Bit */
    ADC1-
>CFGR1 |= (2U << 3); /* Setting resolution on configration register as 8 bit */
    ADC1->CFGR1 &= ~(1U << 13); /* Single conversion mode enabled */
    ADC1-
>SMPR |= (0 << 0); /* Sampling Time Register is set as 1.5 ADC Clock Cycles */
    ADC1->CFGR1 |= (3U << 6); /* External trigger selected as TRG3 */
    ADC1->CFGR1 |= (1U << 10); /* External trigger set as rising edge */
    ADC1->CHSELR |= (1U << 0); /* Channel is set as PA0 */
    ADC1-
>IER |= (1U << 3); /* End of conversion sequence interrupt enable is set as 1*/
    ADC1->CR |= (1U << 0); /* ADC Enable bit is set as 1 */
```

```
    while( !(ADC1->ISR & (1 << 0))); /* If ISR is 0 break */
    NVIC_SetPriority(ADC1_IRQn, 2); /* Setting priority */
    NVIC_EnableIRQ(ADC1_IRQn); /* Enabling the ADC interrupt function */
    ADC1->CR |= (1U << 2); /* ADC start conversion bit is set as 1 */
    /* Setting TIM3 to use on microphone input operations */
    RCC->APBENR1 |= (1U << 1); /* Enable TIM3 Clock */
    TIM3->CR1 = 0; /* Clearing the control register */
    TIM3->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM3->CNT = 0; /* Zero the counter */
    TIM3-
>PSC = 1599; /* Setting prescaler as 16000 to achieve a millisecond on my delay
 function */
    TIM3->ARR = 1; /* Auto Reload Register */
    TIM3->CR2 |= (2U << 4); /* Master Mode Selection is set as Update */
    TIM3->CR1 |= (1U << 0); /* Enable TIM3 */

    /* Setting up the PWM */
    /* Setup PB9 as Alternate function mode */
    GPIOB->MODER &= ~(3U << 2*9);
    GPIOB->MODER |= (2U << 2*9);
    /* Choose AF2 from the Mux */
    GPIOB->AFR[1] &= ~(0xFU << 4*1);
    GPIOB->AFR[1] |= (2U << 4*1);
    /* Setting TIM17 for PWM on PB9 */
    RCC->APBENR2 |= (1U << 18); /* Enable TIM17 Clock */
    TIM17->CR1 = 0; /* Clearing the control register */
    TIM17->CR1 |= (1U << 7);  /* Auto Reload Preload Enable */
    TIM17->CNT = 0; /* Zero the counter */
    TIM17->PSC = 5; /* Prescaler */
    TIM17->ARR = 255; /* Auto Reload Register */
    TIM17->CR1 |= (1U << 0); /* Enable TIM17 */
    TIM17->CCMR1 |= (6U << 4); /* Capture/Compare mode register */
    TIM17->CCMR1 |= (1U << 3);
    TIM17->CCER |= (1U << 0); /* Capture/Compare enable register */
    TIM17->CCR1 |= (1U << 0); /* Capture/Compare register 1 */
    TIM17->BDTR |= (1U << 15); /* Break and Dead-Time register */

    /* Setting up I2C */
    /* PB7 will be used as SDA */
    GPIOB->MODER &= ~(3U << 2*7); /* Setup PB7 as Alternate function mode */
    GPIOB->MODER |= (2U << 2*7);
    GPIOB->OTYPER |= (1U << 7); /* Output type set as open-drain */
    GPIOB->AFR[0] &= (0xFU << 4*7);
    GPIOB->AFR[0] &= (6U << 4*7); /* Choosing AF6 */
    /* PB8 will be used as SCL */
    GPIOB->MODER &= ~(3U << 2*8); /* Setup PB8 as Alternate function mode */
    GPIOB->MODER |= (2U << 2*8);
    GPIOB->OTYPER |= (1U << 8); /* Output type set as open-drain */
    GPIOB->AFR[1] &= (0xFU << 4*0);
    GPIOB->AFR[1] &= (6U << 4*0); /* Choosing AF6 */
    RCC->APBRSTR1 |= (1U << 21); /* Enabling I2C Clock */
    I2C1->CR1 = 0; /* Clearing I2C Control Register */
    I2C1->CR1 |= (1U << 7); /* Error Interrupt Enabled */
    I2C1->TIMINGR |= (3U << 28); /* Timing Prescaler */
    I2C1->TIMINGR |= (0x13U << 0); /* SCL Low Period */
    I2C1->TIMINGR |= (0xFU << 8); /* SCL High Period */
    I2C1->TIMINGR |= (2U << 16); /* Data Hold Time */
    I2C1->TIMINGR |= (4U << 20); /* Data Setup Time */
    I2C1->CR1 |= (1U << 0); /* Peripheral Enable */
```

```
    NVIC_SetPriority(I2C1_IRQn, 0); /* Setting priority */
    NVIC_EnableIRQ(I2C1_IRQn); /* Enabling the I2C interrupt function */


    state = start; /* Return to idle state */
    /* Active Running */
    while(1)
    {
        state_machine();
    }
}

int main(void) {
    system_initialize(); /* Calling the system initializer */
    while(1) { } /* Endless loop */
    return 0;
}
```