Berkay İPEK 2304814

EE 447 Term Project:

Final Report

Audio Frequency Based Stepper Motor Driver

Introduction

In this report, implementation procedure to the project is given in detail. Project was to build a stepper motor driver system based on an applied audio signal's frequency. Project will be discussed in five main parts. Initialization part is given to specify which modules are used. Connection part is to inform the user about all cable connections. The third part is Sampling Audio, which consisting of taking input from microphone and adjusting data for FFT. The next part is frequency detection. In this part, I will describe how I detect the frequency from the output of FFT. Final part is user interface, which consists of three sub-parts, LED, Stepper Motor, and Screen.

Initialization

In this part, general prospective for necessary initialization are going to be given, as you can see in table 1.

Table 1: Initialization for ports and their purposes

Port-pin	**Purpose**
PE3	ADC0 module
PB0-3	Stepper Motor Driver
PF0 & PF4	Switches on Board
PF1-3	LED on board
None	SysTick
PA2-5	SSI0 module

Connection

In this part, general prospective for necessary connection of the project are going to be given, as you can see in table 2.

Table 2: Connections necessary for the project

Port-Pin	**Connected to**
3.3V	Vdd for Microphone
GND	Stepper Motor Driver Ground Pin GND for Microphone GND for Screen
VBUS	Stepper Motor Driver V+ Pin
PE3	Out pin for Microphone
PB0	IN1 for Stepper Motor Driver Ground Pin
PB1	IN2 for Stepper Motor Driver Ground Pin
PB2	IN3 for Stepper Motor Driver Ground Pin
PB3	IN4 for Stepper Motor Driver Ground Pin
PA2	CLK for Screen
PA3	CE for Screen
PA4	DC for Screen
PA5	DIN for Screen

Sampling Audio

The first step of this project is to take audio signal as an input and calling FFT. One of requirements for this project was to do sampling in Systick interrupt with the frequency of 2kHz. Therefore, while initializing this module, reload value of SYSCTRL is to be 2000. Moreover, to be consistent with dealing multiple interrupts, Priority level is given as 4. Then, program will call this interrupt handler with frequency of 2000Hz. In a single call, program will wait until ADC finishes sampling. Then, the current signal level is to be recorded at the memory. When another

call is coming, another signal level will be recorded at the memory, consecutively. In other words, storing data is like to be an array. This method will be done until array length reaches 256 elements. After this condition is met, the function FFT can be called, and output will be decided.

As given manual states, there is a default offset in the microphone module. To be handle with it, sampled data should be subtracted by 1.25V. If we consider 3.3V is max voltage and its sampling data is to be 0xFFF. Offset can be considered as 1552. (1552 ~= (1.25/3.3) *(0x FFF)). That's why program will subtract this value from the sampled data. Moreover, FFT will be waiting for 16 bits to be analyzed whereas sampled data is going to be in 12 bits. Therefore, to be more precise, we are going to shift these 12 bits to higher side of sampled data by calling LSL #4. Then, we are going to store these values to array.

Discussion from now on can be explained with a flowchart in figure 1. When array length becomes 256, the FFT function is going to be called. FFT part is going to be discussed in the next part.

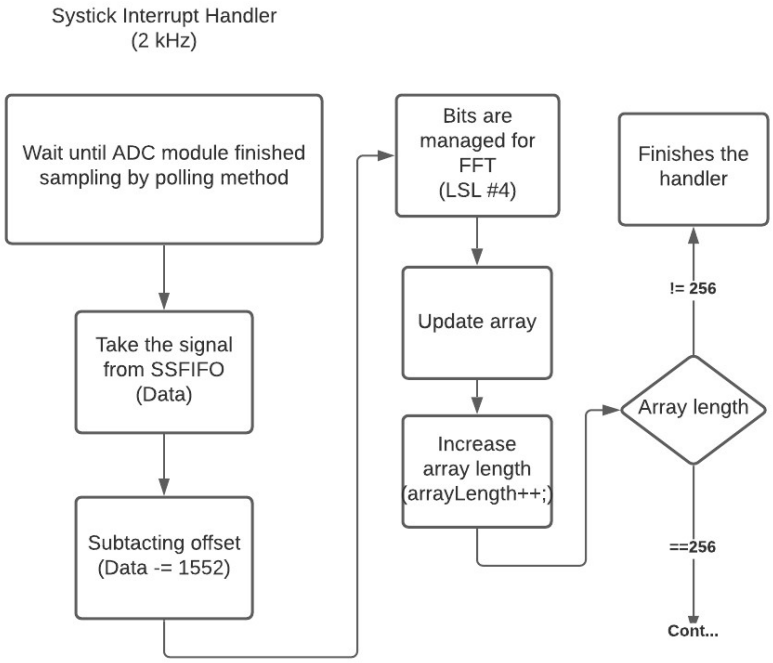

Figure 1: Flowchart for Sampling Audio

Frequency Detection

As the previous part states, this detection algorithm will be called in systick handler with matching array length condition. Therefore, it will be done in each 256 sampling. The first part is to call FFT function that exits in CMSIS library. Program will call the subroutine arm_cfft_sR_q15_len256. As an output, it will modify the given array stored in memory. It will switch each sampled data item with real and imaginary magnitudes of frequencies. Thanks to this modification, the program will be able to detect most dominant frequency. The algorithm, to detect dominant frequency and its magnitude value, is as follows:

- Calculating magnitude:

As manual states, it is known that program will give output in format of 0x AB CD EF GH (ABCD + j EFGH). Program is expected to calculate $ABCD^2 + EFGH^2$ and store that value in the same location again. Program will do the same procedure for top side of value (first 2 bytes, ABCD) and bottom side of it (last 2 bytes, EFGH). The procedure is to take these values and multiply themselves (MUL R1, R1). However, when it comes to negative numbers, it will cause some problems. Before taking square of negative numbers, program should be able to get its absolute value. Therefore, the first thing to do is to check number is negative or not. To do so, program will do AND operator on that 2 byte-data with 0x8000. If this operation results in 0x8000, that means sign bit is 1, (negative number). On the other hand, if result is 0x0000, then number should be positive since sign bit is zero.

If the number is positive, then do simple MUL operator to take square.

If the number is negative, then subtract 0xFFFF from this data to get absolute value of it. Afterwards, taking square can be done.

After $ABCD^2$ and $EFGH^2$ is calculated, these numbers are going to be added. Let's say result is 0xKLMNOPRS. This number is going to written into the memory address of 0xABCDEFGH. The short explanation for this algorithm is shown in figure 2.

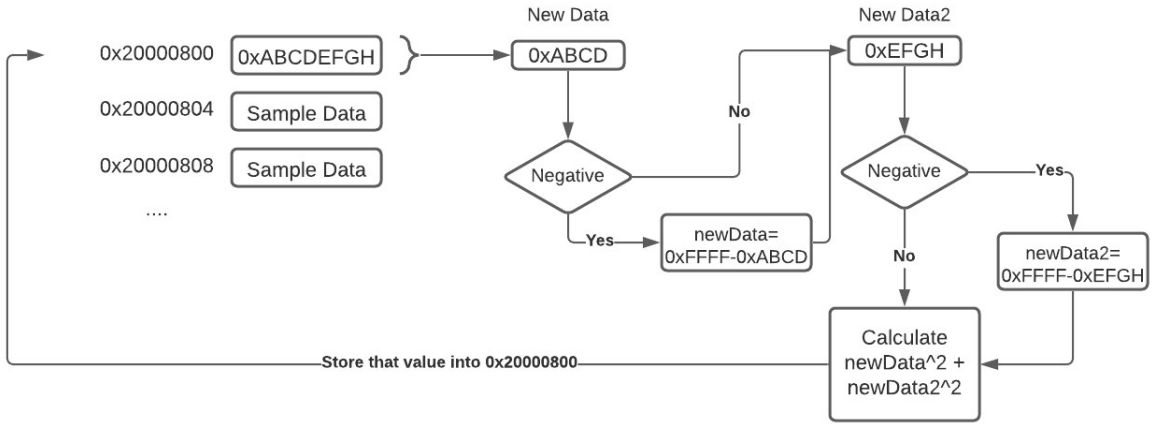

Figure 2: Flowchart for Calculation Magnitude

- Finding Dominant Frequency and its Magnitude

Since all transformed data are converted to its magnitude value. It is easy to get maximum value of magnitude among all transformed data. Program is going to assign a register to 0x00 value. This register will help the program to store maximum value of magnitude. Also, since it is iterating all elements in array via an offset register, the program will be able to get index value and magnitude value at the same loop. However, index is not going to be frequency. To get the frequency, the program will multiply the index number by 8. The reason is that

$$f=i*Fs/N$$

$$f=i*2k/256 \sim= 8*i$$

However, in the program that I wrote, incrementation of index is four. Therefore, in the program, offset (index) value is multiplied by 2.

After all these operations, frequency and magnitude values are stored at the memory.

User Interface

1) LED on board:

As one of the requirements says, LED condition is dependent of most dominant frequency and its magnitude value. If system detect maximum magnitude as below *Magnitude Threshold* value, LED should be off. Therefore, the first thing to do is checking these two values. If it is above *Magnitude Threshold* value, the system will decide color of LED. This color depends on the value of dominant frequency.

If (dominant frequency < *Low Frequency Threshold):*

RED color is shined

If (*Low Frequency Threshold* < dominant frequency < *High Frequency Threshold):*

GREEN color is shined

If (dominant frequency < *High Frequency Threshold):*

BLUE color is shined

While implementing this algorithm, I firstly compared the magnitude and if it is lower, then 0x00 is written to LED data and finishes Interrupt Handler. (If not) I compared the dominant frequency with Lower Threshold value and if it is lower than *Low Frequency Threshold*, then 0x02 is written to LED data and finishes Interrupt Handler. (If not) I compared the dominant frequency with *High* Threshold value and if it is lower than *High Frequency Threshold*, then 0x04 is written to LED data. (If not) 0x08 is written to LED data.

2) Stepper Motor:

To give steps for all time, Timer0A module is utilized. In each call of interrupt subroutine, its data is shifted right or left according to direction, which is stored at the memory and encoded as 0x00 and 0x01. (Default it is 0x01) At the end of shifting data, speed is again adjusted. Speed of motor is dependent of TAILR register of the timer. I tried some values to calculate which speed is proper, and results were 0x2ED4 is for maximum speed rotation and 0x36F0 is for minimum speed rotation. Moreover, frequency range is between 000 and 999 Hz. As a result, linear relation is built between TAILR and frequency value. It was about speed =

14064 -12 * frequency. When magnitude of dominant frequency is reaching to threshold, this speed calculation is done. That's why when magnitude is lower than threshold value, it retains its speed.

In the main subroutine, I did all initializations, and wait for a pressed buttons SW1 and SW2. After realizing this button direction is changed and stored at the memory. Since direction is decided in Timer0A interrupt module, its rotation is changed immediately.

It should be noted that we are using PORTF_DATA register. Therefore, it is important to mask other bits since LED are also changeable within input frequency.

3) Screen:

To be able to display anything on the screen SSI module is used. As a remainder, PortA is assigned to this SSI module.

There is a subroutine (CursorSettings) that adjusts where to write data on the screen. Firstly, data of port A is written to open Command Mode (clearing D/C). In this mode, we are adjusting screen, not printing anything. Then H is cleared and sent to SSI module. X and Y address are given accordingly. Again, these data are sent to SSI module. While giving data to data register of SSI module, the program is going to wait until SSI is not busy. Then after doing these data adjustments, data of port A is written 0x40 to close Command Mode.

After adjusting this configuration, where to put pixels, a subroutine is written called as printR2Hex. This subroutine is printing R2 register in decimal on the screen. To convert hexa to decimals, I used the algorithm in CONVRT. However, this subroutine is printing only three digits. Therefore, I firstly take hundreds digit and set the location on the screen location (R10, 5). Then, it is applied to tens digit and print it on the screen location of (R10, 10). Lastly, units digit is printed on the screen location of (R10, 15). Also, this printR2Hex subroutine is calling CursorSettings inside of it.

To print a number, I adjusted pixels manually. The table as follows:

```
10  NUMBERS         DCB     0x3C, 0x42, 0x81, 0x42, 0x3C ;0
11                  DCB     0x00, 0x00, 0xFF, 0x00, 0x00 ;1
12                  DCB     0xE1, 0x91, 0x89, 0x85, 0x83 ;2
13                  DCB     0x91, 0x91, 0x91, 0x91, 0xFF ;3
14                  DCB     0x0F, 0x08, 0x08, 0x08, 0xFF ;4
15                  DCB     0x9F, 0x91, 0x91, 0x91, 0xF1 ;5
16                  DCB     0xFF, 0x91, 0x91, 0x91, 0xF1 ;6
17                  DCB     0x01, 0x01, 0x01, 0x01, 0xFF ;7
18                  DCB     0xFF, 0x91, 0x91, 0x91, 0xFF ;8
19                  DCB     0x9F, 0x91, 0x91, 0x91, 0xFF ;9
```

Figure 3: Numbers table in printR2Hex subroutine

To give these numbers, I firstly wrote each number into 8x5 pixels. Then, which pixels should be filled is decided and the number is written in a that way. For example, for the number five:

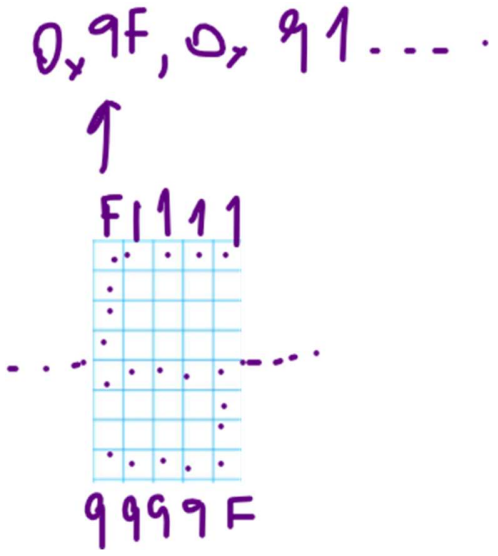

Figure 4: The encoded method for the number 5

Thanks to this table, we can print a number is very easy. Each byte should be sent to SSI module. According to the desired number, there should be an offset to access its bytes. This offset is 5*Number. In other words, initial byte of number 0 is at the address of NUMBERS whereas number 1 is at the address of NUMBERS+ 5*1. By using this algorithm, each digit is printed. After each printing operations, some delay operation is embedded.

This printing operation is done for frequency thresholds, and current frequency value. In the SysTick interrupt handler, when a frequency is detected, it is updated. Therefore, when there is no dominant frequency, it is printing with a high speed so that any number is readable. However, when there is a dominant frequency, it is printing in a stable way, which is expected.

Conclusion

For this report, I explained my solution in a detail way. Firstly, we have looked at connection and initializations of modules. Then, I tried to explain how I take audio signal and detect the frequency. Finally, the user interface is discussed. How I turned on LED's or rotate the stepper motor or display the digits of frequencies on the screen. In this project, laboratory lectures of EE447 were very helpful to me, and this project was a great chance to cover all learned modules at the same time.