EE4144
Final Project Report: Word Match

Calvin Ong

Introduction

Word match is a game that I designed on the STM32F4 Discovery board. The game involves the board's microphone and three of its LEDs. A person speaks into the microphone. Depending on how loud the person speaks, one of the three LEDs light up – LED1, LED3 or LED4. Based on whichever LED lights up, a random animal word is shown in the output screen.

Description

To implement the design, the STM32F4 Discovery board and type A to mini-B USB cable were used. The main implementation for the design was in the C code written on Keil uVision. For the library utilizing the board's microphone, the code from Lab 4 Part 3.2 was used.

A few of the board's peripherals is utilized- the ADC, RCC, SPI, I2C, GPIO and TIM.

The CMSIS arm cortex math library is also utilized for calculations involving the sound loudness from the microphone.

Implementation

To access the microphone, the code from Lab 4 Part 3.2 was used. The microphone library code is accessed via the following two headers:

#include "microphone_functions.h"

#include "sound_loudness.h"

An array size was needed to determine the appropriate number of samples to take before determining the exact sound loudness. This is defined as the global variable:

#define AUDIO_BUF_SIZE 2048

The samples are kept in an array of floats:

float audio_buffer[AUDIO_BUF_SIZE];

To output the random animal words, an array of strings were needed. However, in C, strings are not supported. Instead, an array of char pointers were created, which pointed to the strings.

*char *sound[] = {"Cow","Duck","Geese","Dog","Cat","Pig"};*

In order to choose a random value from the sound array to output, three global variables are kept. They are volatile to ensure that these variables are updated and to make them as random as possible. Their values correspond to the indices in the sound array.

volatile uint32_t soft = 0;

volatile uint32_t medium = 0;

volatile uint32_t loud = 0;

In the main function, the board peripherals had to be initialized:

SystemInit();

Buttons_Initialize();

LED_Initialize();

led_timer_init();

microphone_init();

microphone_start();

A local variable is kept, which is designated to be the actual sound loudness calculated after 2048 samples are taken. Another local variable is kept as the current index of the sound array.

uint32_t audio_buffer_index = 0;

float sound_loudness;

A thread is created to continuously run the program. Within the thread, there is 1 subthread, which continuously reads samples from the microphone's external environment. These samples can be sound or vibrations (such as tapping the microphone). Within the main thread, two variables are kept. One is to keep track of the number of samples currently read, and the other is a pointer to an array to read the sample into.

uint32_t n_samples; // number of samples read in one batch from the microphone

*uint16_t *audio_samples;*

```
while(1) {
        audio_samples = microphone_get_data_if_ready(&n_samples);
            if (audio_samples != 0) break;
}
```

Within the same main thread, the samples from the *audio_samples are continuously read into the sound array until 2048 samples are read.

```
for(int i = 0; i < n_samples; i++){
        audio_buffer[audio_buffer_index ++] = (int16_t) audio_samples[i];
        if (audio_buffer_index >= AUDIO_BUF_SIZE) break;
}
```

Once the array is filled with 2048 samples, the sound loudness is calculated via the board's built-in *calculate_sound_loudness* function. Based on the sound loudness, the PWM brightness is set to 90%. if it less than 500, LED1 is turned on and all others are off. If it is greater than 10000,

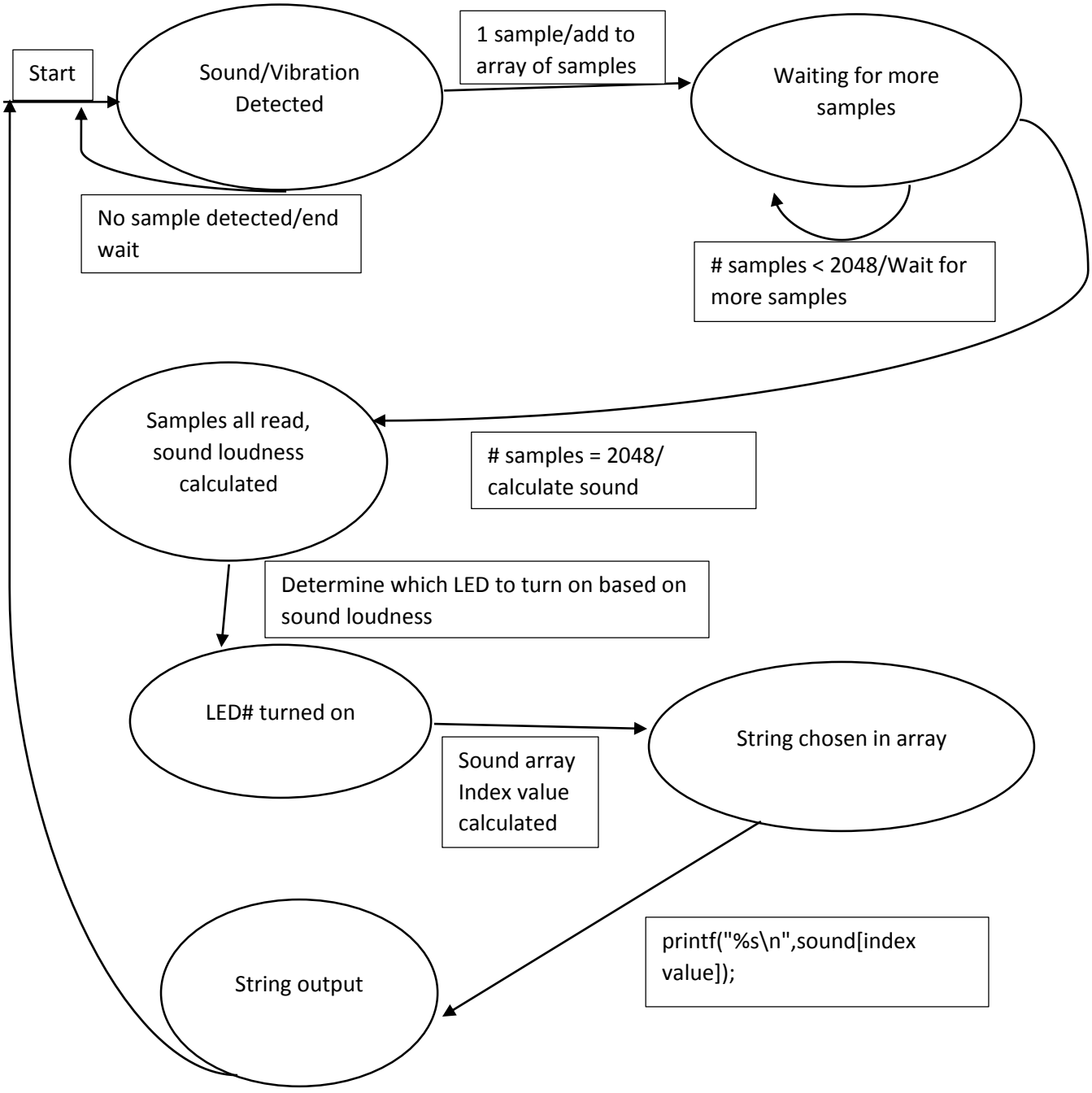
then LED 4 is turned on. Otherwise, the PWM frequency is randomly set. The LEDs are set using the board's PWM *setTIM4_PWMDuty* function. After the LEDs are turned on, the sound array indices (soft, medium, loud) are incremented. To ensure that the indices do not go past the size of the array, the modulus operation is performed. The modulus dividend is 5, since that is the max index of the array. Once the index is chosen, the value of that index in the array is outputted. The *audio_buffer_index* variable is reset back to 0, and the thread continues to run.

```
if (audio_buffer_index >= AUDIO_BUF_SIZE)
        {
                sound_loudness = calculate_sound_loudness(audio_buffer,
AUDIO_BUF_SIZE);


                        // use sound_loudness to control brightness of LED


                if(sound_loudness < 500) {
                        setTIM4_PWMDuty(1,90);
                        setTIM4_PWMDuty(2,0);
                        setTIM4_PWMDuty(3,0);
                        setTIM4_PWMDuty(4,0);
                        soft++;
                }
                else if(sound_loudness > 10000) {
                        setTIM4_PWMDuty(4,90);
                        setTIM4_PWMDuty(1,0);
                        setTIM4_PWMDuty(2,0);
                        setTIM4_PWMDuty(3,0);
                        loud++;
                }
                else {
                        int var = 20 + 50*log(3.14) *((sound_loudness - 500)/(10000 - 500));
                        setTIM4_PWMDuty(3,var);
```

```
            medium++;
        }
        uint32_t softIndex = soft % 5;
        uint32_t mediumIndex = medium % 5;
        uint32_t loudIndex = loud % 5;
        printf("%s\n",sound[softIndex]);
        printf("%s\n",sound[mediumIndex]);
        printf("%s\n",sound[loudIndex]);

        audio_buffer_index  = 0;
        }
}
```

Diagrams

A state diagram is shown to illustrate this game's process:



Conclusion

The game utilized key concepts of embedded systems. A state diagram was needed to show how an external input from a sensor was translated into electrical signals to generate a desired output, where the board acted as an actuator and computational system. The board's peripheral library for the microphone had to be understood- the concept of taking multiple samples from an environment to generate a single aggregated value. Taking multiple samples is important as the concept of button bouncing extends to signals as well, where signal waves are often erratic in short periods of time. Pulse-width-modulation (PWM) is another concept, where a signal (ie LED light) is manipulated to have a certain brightness and frequency at which it blinks. In order to control this PWM, knowledge of which channels are connected to which LEDs and the sensitivity of the microphone and other peripherals are necessary.