Politecnico di Milano - Department of Electronics, Information and Bioengineering

STM microcontroller microphone game

[Coding Project]

Student Odri Tomas / Ivona Skorjanc
ID 895061 / 895116

Course Embedded Systems and Advanced Operating Systems
Academic Year 2017-2018

Advisor Federico Terraneo
Professor William Fornaciari

July 19, 2018

Contents

1 Introduction

The main idea of this project was to get familiar with the Miosix [1] real-time operating system and use drivers for different peripherals on the STM32F4 development board (Figure 1). This was done by implementing a game using a microphone embedded on the board, as well as frequency detecting code. The peripherals used were the MEMS (micro-electromechanical) microphone, serial connection, random number generator, audio jack, light-emitting diode and the user push button. The microphone driver used also exploits the ADC (analog-to-digital converter), DMA (direct memory access), I2S (Inter-IC Sound) and SPI (Serial Peripheral Interface) peripherals. To use the serial connection, the board was connected using USB TTL cable.

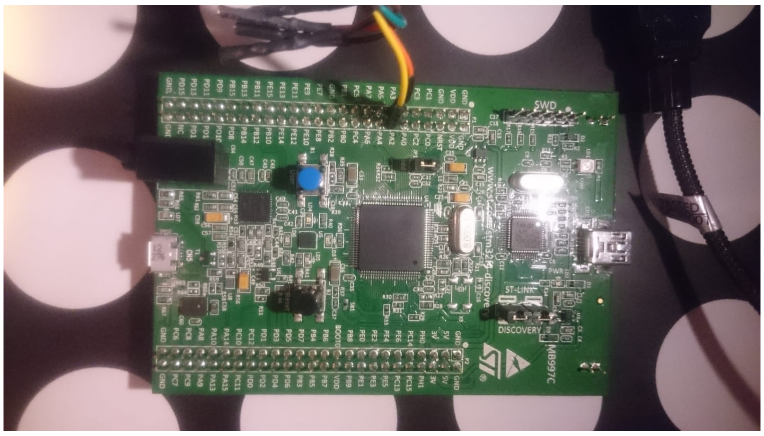
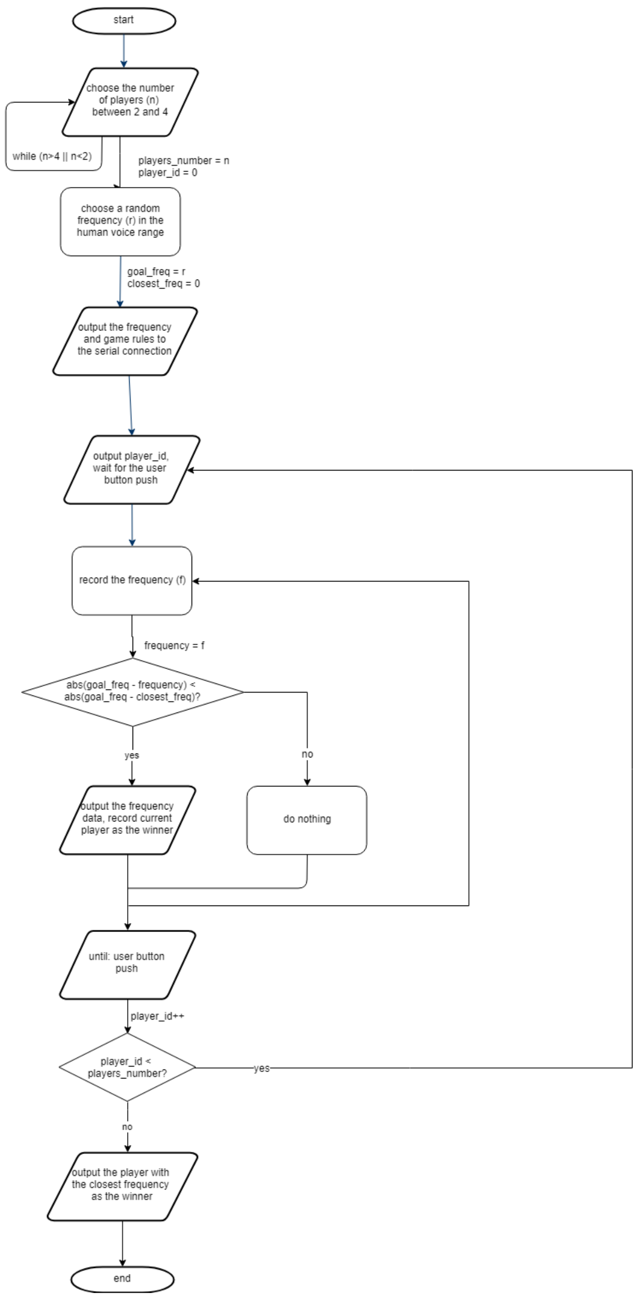


Figure 1: STM32f4 microcontroller connected with the USB TTL Serial cable.

2 Design and implementation

2.1 Game logic



start
choose the number of players (n) between 2 and 4
while (n>4 || n<2)
players_number = n
player_id = 0
choose a random frequency (r) in the human voice range
goal_freq = r
closest_freq = 0
output the frequency and game rules to the serial connection
output player_id, wait for the user button push
record the frequency (f)
frequency = f
abs(goal_freq - frequency) < abs(goal_freq - closest_freq)?
yes
no
output the frequency data, record current player as the winner
do nothing
until: user button push
player_id++
player_id < players_number?
yes
no
output the player with the closest frequency as the winner
end

Figure 2: The game flowchart diagram.

The game can be played between a minimum of 2 and a maximum of 4 players. The microcontroller plays a frequency, and the player who can whistle closest to the given frequency wins. The details can be read from the flowchart diagram shown on Figure 2. First, on the serial prompt the users are asked to input the number of players using the keyboard. If the number isn't from the [2,4] interval, an error message is printed and the user is asked to repeat the input until he chooses a correct number. The program also uses a random generator to choose the goal frequency that the players have to repeat. That frequency is outputted as sound via the audio jack, as well as a number written on the serial prompt. The program further instructs the users to input their frequencies by whistling one by one to the microphone on the STM board. When it's the turn of each one of the players, their player id is indicated on the serial prompt, along with the instructions to input the whistle. Each player should first push the user button to start the recording. After whistling at a desired frequency, the player should push the user button again to end the recording. While the frequency is being recorded, if the current player gets closer to the goal frequency than the previous ones, his successful attempt is followed by a notification on the serial prompt. He is then recorded as the current winner of the game. This loop is repeated until all the players have completed their turns, and the player who is last recorded as the winner wins the game.

2.2 Peripheral drives

This project is based on an existing project of another student (Giovanni Beri). In his application, he recorded sound from an embedded microphone. To start or stop the recording, the user button should be pressed. We went through the existing programming code, understood it, edited it and added parts specific for our application. The whole project has been executed on an STM32F4 Discovery board that leverages the performances of the STM32F407VGT6 high performance microcontroller and also allows different application development, including applications featuring audio. The STM32F407VGT6 microcontroller features a 32-bit Cortex M4 ARM with FPU core, 1 Mbyte of Flash memory and 192 Kbyte of RAM. It features a ST-LINK in-circuit debugger and programmer, used for flashing the memory and debugging our project. The game logic is executed in a sequential order. The users are first asked to define the number players. The application checks if the input is really a number between 2 and 4. Afterwards, the hardware random number generator gives the frequency the players are trying to reach. After knowing the frequency the players record a whistle sound, one after another by pressing the user button. At the end of the game the player closest to the required frequency wins.

The Miosix operating system is included into the workspace. It is used to access the device peripherals (buttons, LEDs), the random number generator, the serial protocol port, threads and interrupts.

2.2.1 Serial

The serial connection is used to receive the number of players from the keyboard of the computer and also to display game instructions and information on the terminal of a personal computer or similar machine. The communication is established using the USART (Universal Synchronous and Asynchronous Receiver-Transmitter) interface. As defined in the board settings file, the speed of the data transmission is 57600, while the connection of the development board and terminal is established using an TTL serial cable connected to pins PA2 and PA3 of the board and USB port of a personal computer. For viewing the received messages and send further messages through the serial port communication, a Putty desktop application can be used on personal computers with a Windows OS, or the screen terminal command on Linux based and Mac OS X operating systems.

2.2.2 Random number generator

Random number generators (RNG) can be divided into two basic classes: deterministic RNGs (pseudo random number generators, that produce a sequence of bits from an initial value called seed) and non-deterministic RNGs (true random number generators, where randomness depends on some unpredictable physical source). Some STM32 microcontrollers, including the one used in this project (STM32F407VG), have embedded RNG hardware peripherals based on an analog circuit. The circuit generates a continuous analog noise used on the RNG processing. Miosix already has a class that helps with handling the hardware random number generator. To use the RNG it was needed to include the RNG files into the project, create an instance of the RNG and simply call the get method that returns a 32-bit random integer. Finally, the RNG drivers were also added to the Makefile. The RNG was used to generate a random frequency that players need to try to reach; hence the frequency is chosen from the range of 500 to 4500 Hz.

2.2.3 User button

The user button is used for starting and stopping the microphone recording sessions. It should be pressed after the number of players is defined and the goal frequency is generated. As written in the board user manual, the button is connected to the PA0 pin of the discovery board and drivers already exists in the Miosix project. The buttons are programmed to use interrupts with whom they signalize that they have been pressed and the application could proceed with another depending function.

2.2.4 LEDs

Functions for using the light-emitting diodes are defined in the board support package header in the existing Miosix project. In this application LEDs are used in a simple way; just to help players identify the state of recording. When the red LED is turned on the microphone is recording a sound and by pressing the button again, the microphone stops recording and the red led turns off.

2.2.5 Onboard audio capability

The STM32F407 evaluation board is equipped with a MP45DT02 MEMS microphone, alongside a CS43L22 digital-to-analog convertor (DAC). The MP4DT02 microphone produces a stream of digital pulse-density modulation (PDM) samples. Each individual bit in the stream is a sample. The amplitude of the audio waveform can be inferred from the density of the pulses/samples. When the amplitude of the wave increases, the density of the 1's in the stream also increases. To be able to use information from the PDM stream, it is necessary to convert the data into a stream of PCM (pulse-code modulation) signals. This is a more familiar waveform, it's characterized with sampling the wave in regular intervals and then each sample is quantized to the nearest value within a range of digital steps.

There are three main steps to convert these PDM samples into PCM:

1. Oversample the PDM microphone by a value x
2. Drive the sample through a low pass filter
3. Downsample the filtered data by a value x.

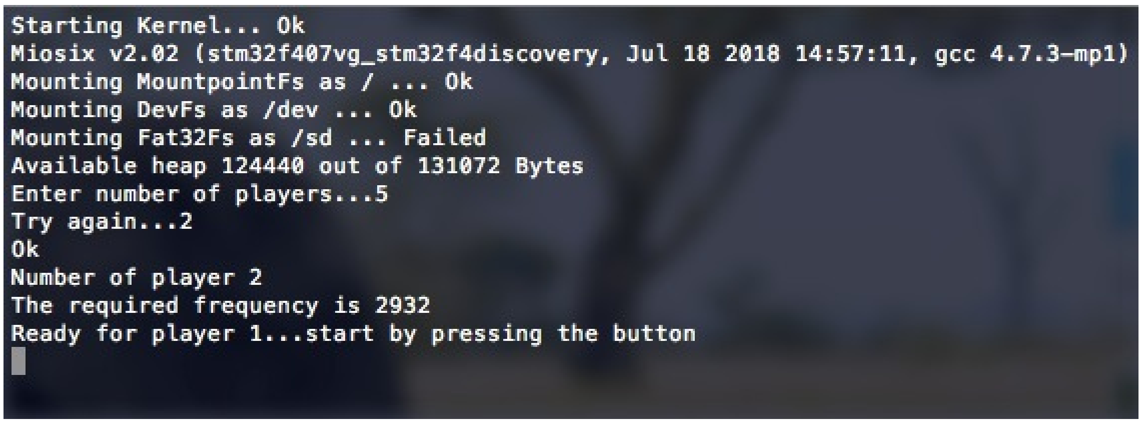
The last two steps are known as decimation.

The microphone is connected to pins PB10 and PC03, where PB10 is used for the input sampling clock and PC3 for the output data samples. The pins are both controlled by the I2S hardware driver and functioning programming code was provided from a past student project. Upon initialization of an instance from the Microphone class, we pass the function for calculation frequencies (with four placeholders) as the callback function. After pressing the button the first time, each player start the microphone recording session. The start function is then called that receives two parameters, the id of the player whose turn it is and the required frequency all the players want to meet in the game. Since audio caption is memory consuming, an DMA interface was used to implement the buffer storage. DMA is used in order to provide high-speed data transfer between peripherals and memory or memory and memory. Data can be quickly removed with the DMA without any CPU action; this helps the CPU to be free for other actions. Upon the call of the start function we allocate memory for several buffers: buffers with the microphone data that is being processed and that is ready for processing, also 2 buffers with ready and processing decimated data and the last one that keeps the FFT (fast Fourier transformation) sample values. The buffers use an interrupt routine to signalize that they are not empty. In the main loop, if a buffer isn't empty processes and filters the PDM samples and finally converts 16 1 bit PDM samples into 1 PCM sample. The filter used in the data processing is the cascade integrator-comb filter that with the combination of one or more integrators, followed by comb sections, actually serves as a decimating filter. After retrieving PCM sample, we use a fast Fourier transformation, so we could obtain a frequency specter of the audio signal. After having frequency samples, we calculate their amplitude and define the index of the frequency with the highest amplitude. This index is used in the callback function to calculate the frequency of the sound. In the callback function, we compare the calculated frequency with the game goal frequency. If the player produces a frequency that has, so far, been closest to the goal frequency, the player id is remembered as the current winner id. The function providing the FFT functionalities are located in the math_fun folder of the project and added to the Makefile.

The board audio DAC is used to output the sound on the audio jack connector. The microcontroller controls the audio DAC thorugh the I2C interface. The first idea was to produce a sound wave of the randomly generate frequency on the audio jack. We have based our prototyping on an existing code that reproduces a sound of the sound trombone on the audio jack. Afterwards, we successfully produced a sound wave of a wanted frequency on the audio jack. After trying to incorporate these prototypes to our application, we have concluded that the system occupies a lot of memory and required multiple writing to and reading from a file system. The sound generation part of the project uses an existing player class with reproduced an ADPCM sound class object. Also a wave generation project was used and modified to get a wav format file that can be reproduced using the player class on the microcontrollers.

3 Results

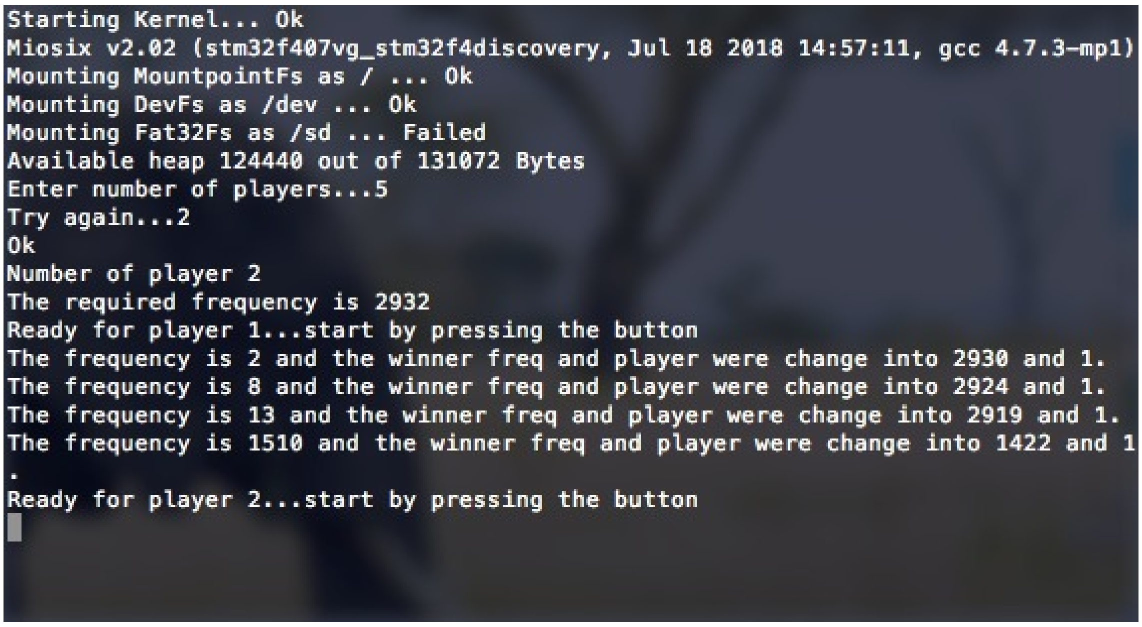
The following screenshots show the game flow output on the serial prompt. At the Figure 3 we can see the response of the game when entering a wrong number of players, as well as the output of a randomly generated frequency.



```
Starting Kernel... Ok
Miosix v2.02 (stm32f407vg_stm32f4discovery, Jul 18 2018 14:57:11, gcc 4.7.3-mp1)
Mounting MountpointFs as / ... Ok
Mounting DevFs as /dev ... Ok
Mounting Fat32Fs as /sd ... Failed
Available heap 124440 out of 131072 Bytes
Enter number of players...5
Try again...2
Ok
Number of player 2
The required frequency is 2932
Ready for player 1...start by pressing the button
```

Figure 3: Game screenshot

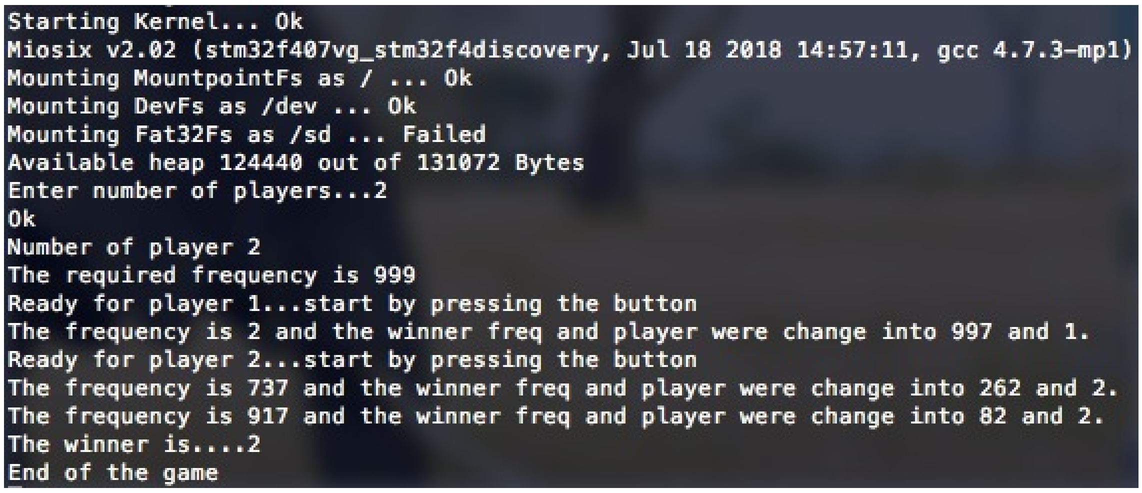
The first player starts by pressing the user button. Every time he gets closer to the goal frequency, his attempt is followed by an informative output (Figure 4 and Figure 5), which indicates the frequency of the player's whistle and the difference between the current and the goal frequency. At the end of the game, the final winner is printed on the screen. Pressing the user button in unexpected moments, doesn't enable sound recording, hence, it's only possible to record a sound when it is a certain players move.



```
Starting Kernel... Ok
Miosix v2.02 (stm32f407vg_stm32f4discovery, Jul 18 2018 14:57:11, gcc 4.7.3-mp1)
Mounting MountpointFs as / ... Ok
Mounting DevFs as /dev ... Ok
Mounting Fat32Fs as /sd ... Failed
Available heap 124440 out of 131072 Bytes
Enter number of players...5
Try again...2
Ok
Number of player 2
The required frequency is 2932
Ready for player 1...start by pressing the button
The frequency is 2 and the winner freq and player were change into 2930 and 1.
The frequency is 8 and the winner freq and player were change into 2924 and 1.
The frequency is 13 and the winner freq and player were change into 2919 and 1.
The frequency is 1510 and the winner freq and player were change into 1422 and 1
.
Ready for player 2...start by pressing the button
```

Figure 4: Game screenshot

As it can be seen from previous screenshots, the results aren't perfect. A part from the randomly generated number, the sound should also indicate the goal frequency. The FFT frequency detection code was causing problems by detecting a lot of null or too low values, but the other calculated frequencies were in the expected whistle range.



```
Starting Kernel... Ok
Miosix v2.02 (stm32f407vg_stm32f4discovery, Jul 18 2018 14:57:11, gcc 4.7.3-mp1)
Mounting MountpointFs as / ... Ok
Mounting DevFs as /dev ... Ok
Mounting Fat32Fs as /sd ... Failed
Available heap 124440 out of 131072 Bytes
Enter number of players...2
Ok
Number of player 2
The required frequency is 999
Ready for player 1...start by pressing the button
The frequency is 2 and the winner freq and player were change into 997 and 1.
Ready for player 2...start by pressing the button
The frequency is 737 and the winner freq and player were change into 262 and 2.
The frequency is 917 and the winner freq and player were change into 82 and 2.
The winner is....2
End of the game
```

Figure 5: Game screenshot

4 Conclusions and Future Works

In conclusion, all the peripherals were successfully accessed and manipulated separately. The game logic was implemented completely. The major curriculum areas from class were applied on a real-life problem. However, to completely finish the idea of the project, issues with sound generation and FFT should be resolved. The FFT should be optimized, while another future goal could be to enable players to first hear the wanted frequency on the audio jack and make the game even more interactive.

References

[1] F. Terraneo. Miosix operating system, website: https://miosix.org/index.html.

[2] Stmicroelectronics. reference manual rm0090 for stm32f407. 2017.

[3] Stmicroelectronics, an5012: Analog-to-digital audio conversion example using stm32l4 series microcontroller peripherals.

[4] Stmicroelectronics, an3397: Audio playback and recording using the stm32f4discovery.

[5] Stmicroelectronics, an4841: Digital signal processing for stm32 microcontrollers using cmsis.

[6] A. Barr. Probing pdm: Mp45dt02 and stm32f407, website: https://www.theunterminatedstring.com/probing-pdm/.

[7] T. Majerle. Stm32f4 fft example, website: https://stm32f4-discovery.net/2014/10/stm32f4-fft-example/.