Politecnico di Milano - Department of Electronics, Information and Bioengineering

POLITECNICO
MILANO 1863

AOS Microcontroller Project

[Buzzer Game]

Student Tobias Rasmussen
ID 894983

Student Knut Rogde
ID 895058

Course Advanced Operating Systems
Academic Year 2017-2018

Advisor Giuseppe Massari
Professor William Fornaciari

September 12, 2018

Contents

1 Introduction

1.1 Task

The goal of this project is to implement a simple game on a micro-controller. The micro-controller we ended up using was a standard NUCLEO-F411RE board(see figure 1). By using this particular board and utilizing a open-source OS called Miosix, we were able to create a simple game we call: "Buzzer".
To realize the game we had to use the internal analog digital converter on the board as well as implementing some buttons. To do this drivers had to be written for both ADC and the buttons.

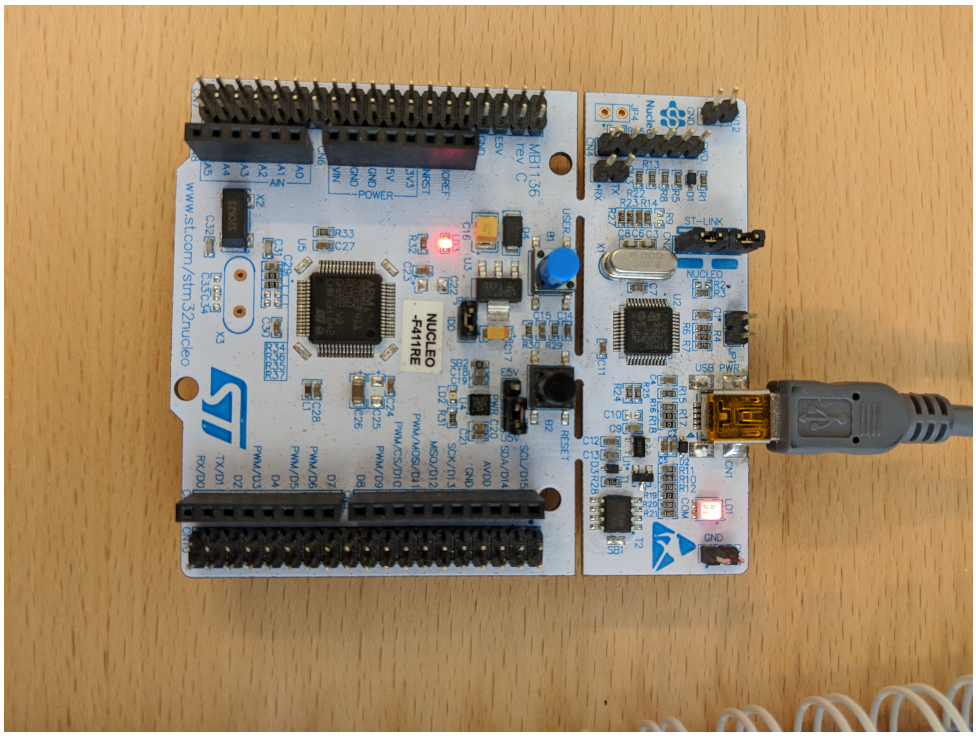

Figure 1: The Nucleo F411RE.

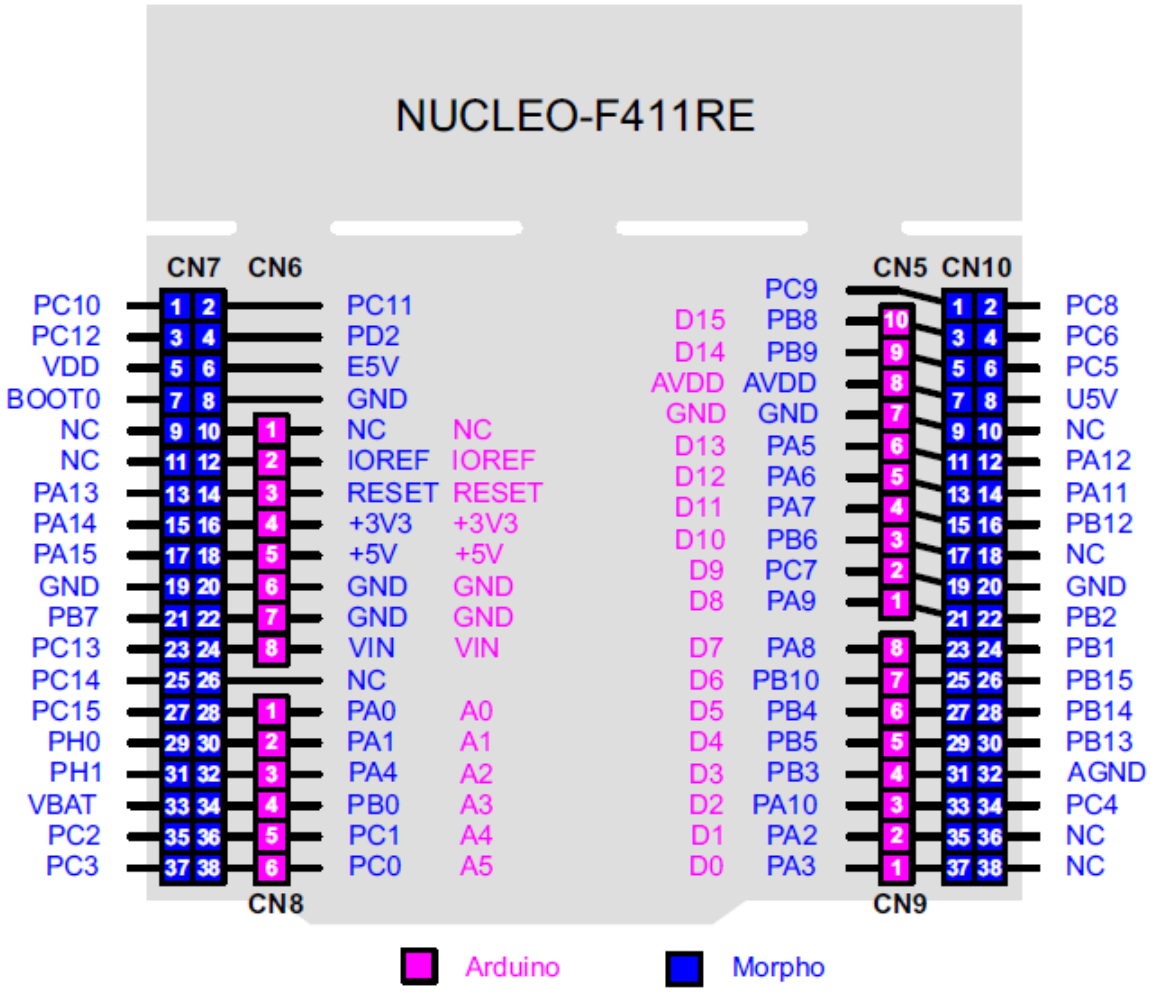

Figure 2: The Pin configuration on the Nucleo.

2 Design and Implementation

2.1 Game Design

The game's idea is simple: Two players compete in guessing the right tone. At the start of a players round, the buzzer makes a random tone for some seconds. Afterwards the player will guess/replicate the same tone by turning the potentiometer. The buzzer will now continuously change tone according to how the player turns the potentiometer. When the player is ready, he presses a button to confirm his guess. Then its the next players turn.
After both players have guessed a tone, the display announces the winner.

Since we put most of the emphasis on the drivers and circuit design, we chose a simplistic game design. Since we were allowed to use a OS, we utilized the tools that came with said OS. Instead of implementing interrupts, we chose to use threading to make the different modules work concurrently. Thus the need for interrupts and watchdogs disappeared. However this introduces a new possible problem: thread safety. Now we must be careful when using shared variables and protect them accordingly. The pthread library gives us all we need, here we have the tools to spawn threads as well as create mutexes. We divided the game into three concurrent modules:

- ADC-thread
- Buzzer-thread
- Game-thread

Each thread has a specific task. The ADC thread, keeps sampling the input of the ADC to update a shared variable for other threads to use. The buzzer thread, easily enough runs the buzzer. The Game thread controls the buzzer thread, and uses the data from the ADC-thread. Within this thread all the game logic resides.

2.2 Peripheral Drivers

To realize the game we needed the following:

- Display
- Buttons
- Buzzer
- Potentiometer

All these items had to be implemented on a breadboard, and connected to the Nucleo.

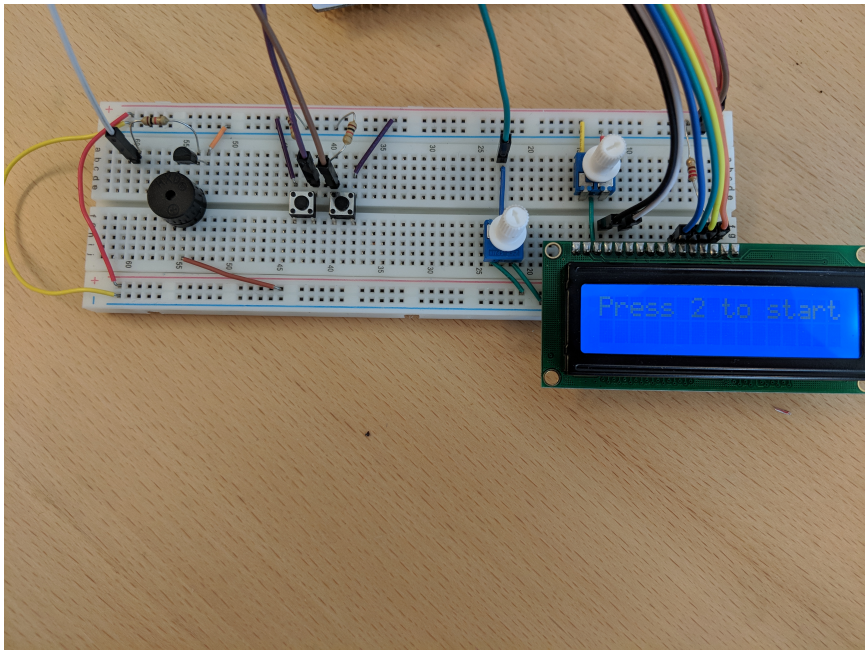

Figure 3: The implementation of the peripherals, on a breadboard.

2.2.1 Display

This was pretty easy to implement, since the Miosix-OS already has a class called: **Lcd44780**. With this class already written, we only had to set up the

display and define the which pins we wanted to use. Since we had no possibility to use serial for writing to screen, we connected the display. This proved to work, almost just as well as the serial connection. However with serial we could have been able to run diagnostics, which could have been helpful in the development phase. The final PIN-configuration:

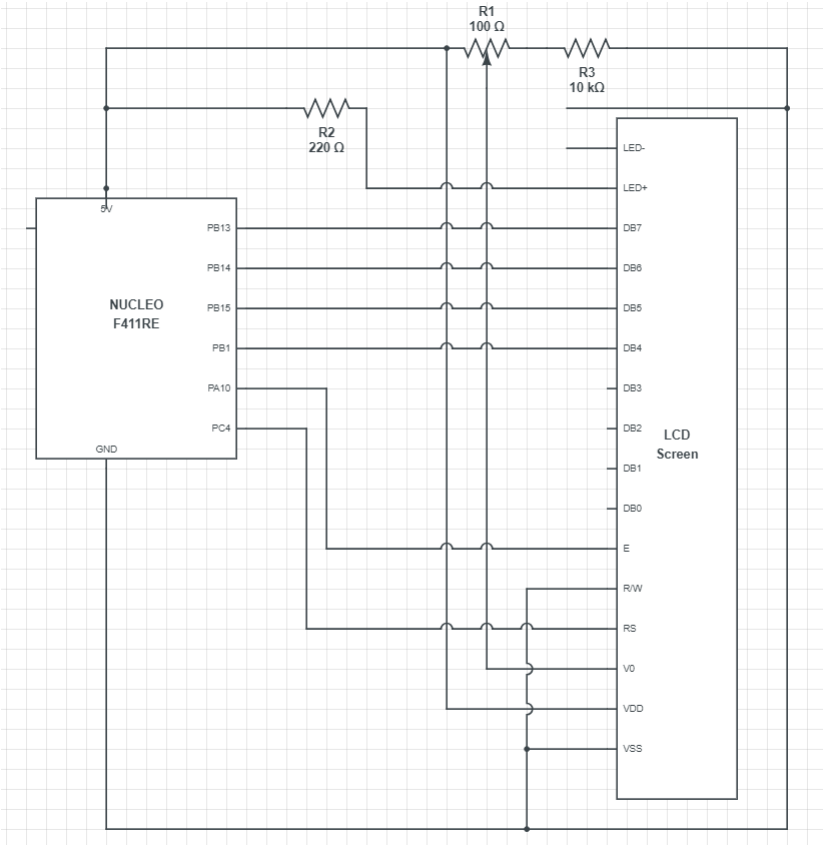

Figure 4: Circuit drawing of the LCD-display.

Display Port	**PIN**
D4	PB13
D5	PB14
D6	PB15
D7	PB1
RS	PC4
E	PA10

This setup worked flawlessly, however we did make some minor changes to the class. These changes were solely to make it more customized for our use.

2.2.2 Buttons

The buttons are pretty simple to implement. The only thing you need is to set the desired pins to input, then poll on these pins. In our case we used: **PA1** for button 1, and **PA0** for button 2. Also see figures 5 and 2.

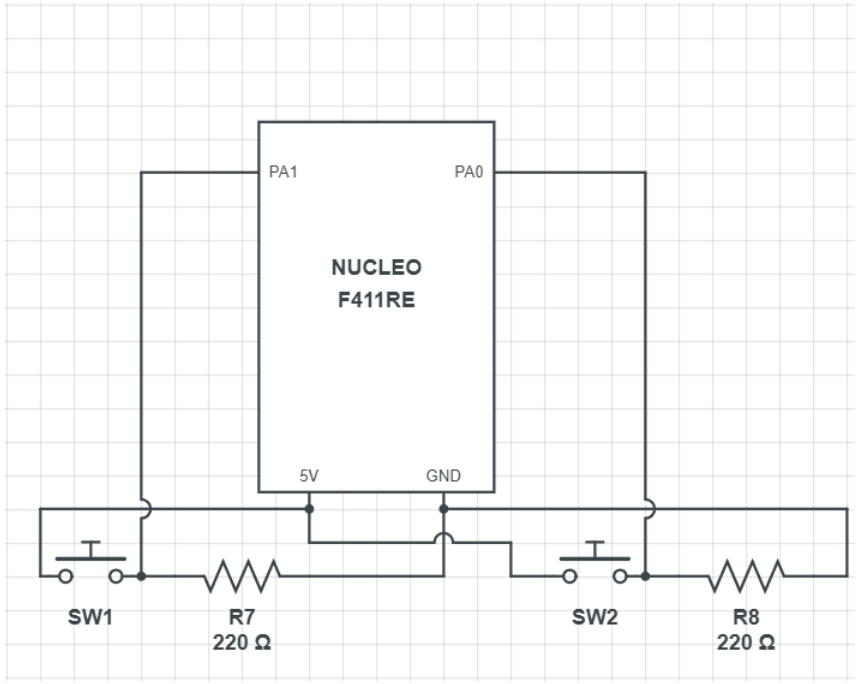

Figure 5: Circuit drawing of the external-Buttons implemented.

2.2.3 ADC

This is were we use our potentiometer. By sampling the voltage from the potentiometer, we can play a tone correlating to the position of the potentiometer(see figure 6 and 3. This way the player can choose a tone, while it plays continuously. To realize this we had to write a ADC-driver, which we chose to wrap into a C++ Class.
Firstly we have to enable the Port-b clock, this is done by writing bit 1 in **RCC−AHB1ENR** to 1. Thus enabling the clock. Further more we need to enable read write access, by enabling the **RCC−APB2ENR** register. To set the resolution for the conversion, we can modify bit 24 and 25 in **ADC1−CR1**. We set them to 0 and 1 respectively(8-bit resolution).
To set the sampling-time we must modify **ADC1−SMPR2**-Register, we set ours to be 112 cycles.
Further we find that the ADC-prescaler is set by the 17th and 16th bit in the **ADC−CCR**-register. We set ours to $\frac{clk}{8}$.
Upon looking at the **ADC1−CR2**-register, we see that there are a lot of options we need to consider.

- Toggle Continuous (set Bit 1).
- Right side allignment (clear Bit 11).
- Turn on ADC (set Bit 0).
- Start conversion(set Bit 30).

In the end we decided not to implement neither watchdog nor interrupt, the reason being our desire to utilize the OS ability for concurrency.

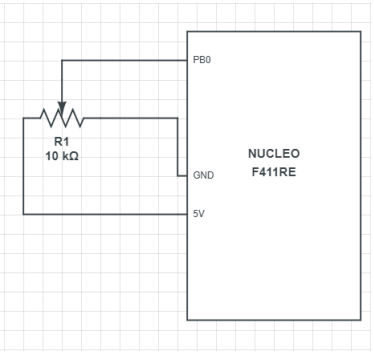

Figure 6: Circuit drawing of the player controlled potentiometer.

2.2.4 Buzzer

The buzzer was fairly simple to implement, we only need to create a PWM signal. This is done by turning desired pin high/low, with a specific interval. The circuit was fairly simple, as you can see in figure 7. We chose to use a transistor, since drawing power straight from the pin proved to be very unstable.

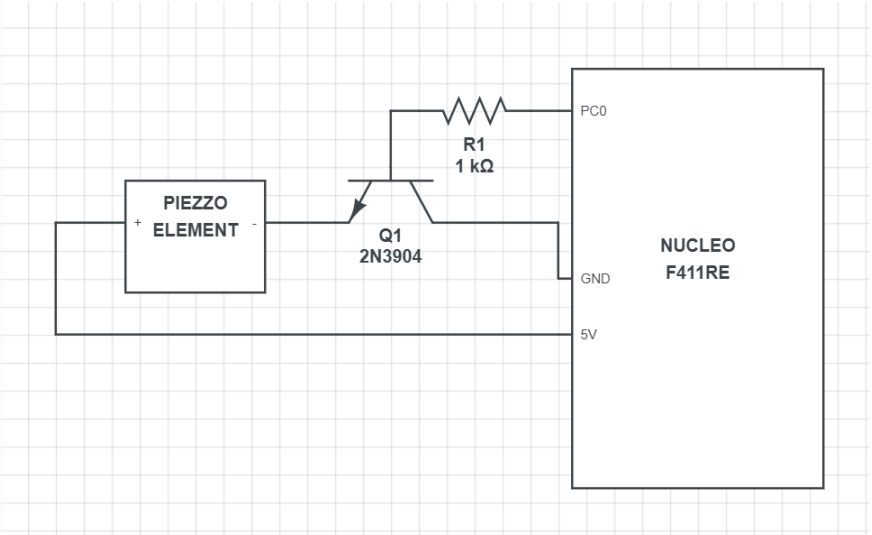

Figure 7: Drawing of the buzzer circuit.

3 Results

After a lot of testing we concluded with that the is sufficient and works well. We even constructed a 24 hour stress test, where we short circuited the buttons to simulate usage. After 24+ hours, the program ran just as consistant as after a restart. This proves that the system is feasible as a real-time application, and proves to some extent that it is a sturdy build.

4 Conclusion

This projects goal was to utilize a open source OS to program a micro controller with peripherals. The main task being using the OS to write a ADC-driver, and then make use of said driver within a real-time application. A task which is accomplished, not without trouble. It is worth mentioning that our group did not get a board from the school(there were too few.), so we had to buy one our selves. This meant that we did not have the discovery kit. This meant that we could not debug our board, making it quite hard to start the project. However after a whole lot of good old "trying and failing", we managed to create what we intended. But it should be mentioned that it would have been a lot easier with the possibility to use serial.