UNIVERSITÀ DEGLI STUDI DI TRENTO

Eagle: recruiting-sw-microcontroller

Author
Michael Bernasconi

Student Number
267681

Contents

Chapter 1

Introduction

1.1 Project Objectives

The purpose of this project is the design and implementation of an embedded system based on the **Nucleo C031C6** microcontroller, with particular attention to the interaction between hardware components and firmware logic. The main goal is to properly configure the peripherals, manage asynchronous events through interrupts, and continuously acquire data using a polling mechanism.

Specifically, the developed firmware is required to demonstrate the following functionalities:

- real-time acquisition and management of an analog signal through **polling**;
- handling of asynchronous digital events through an **interrupt** routine;
- clear and modular organization of the code using the STM32 **HAL** libraries;
- generation of an **analog graph** of the acquired data, including the `RAW`, `NOISE`, and `Moving Average (MA)` modes;
- generation of a **digital graph** of the data obtained through interrupts;
- playback of an **acoustic signal** during continuous analog signal detection.

1.2 Context

The project is based on the `microcontrollers` repository, available at github.com/eagletrt/recruiting-sw. It is an embedded application that simulates a real-time event acquisition and detection system, aiming to integrate analog and digital readings and verify their behavior through serial and graphical output.

Originally, the task required the use of a **Hall sensor** for both analog and digital readings. However, due to a malfunction of the component, the sensor was replaced with a **potentiometer** for the analog part and a **button** for interrupt generation and digital input. This replacement allowed the overall project logic to remain unchanged.

Chapter 2

Project description

2.1 General Overview

The project consists in developing firmware for the **Nucleo C031C6** board, aimed at managing analog and digital signals in real time. The system combines two acquisition modes:

- analog reading in **polling** mode via **ADC1**, used to monitor a variable signal provided by a potentiometer;
- management of digital events through the physical button (USER button) detected in **polling**.

Originally, the project intended the use of a **Hall sensor** for both analog and digital readings. However, due to a malfunction of the sensor, it was replaced with a **potentiometer** for analog readings and with the **USER button** to handle digital events and change the FSM state.

2.2 Architecture and Components

The system is based on the interaction between various hardware peripherals and STM32 HAL software modules:

- **ADC1**: performs analog-to-digital conversion of the signal from the potentiometer;
- **UART2**: handles serial communication for printing values and status messages;
- **GPIO**: used for reading the button state and controlling the status LEDs;
- **USER button**: used to change FSM states through falling-edge detection in polling;
- **Interrupt button**: used to trigger an interrupt and read its digital value when pressed;
- **Green LED**: provides a visual indication of the system state.

2.3 System Operation

The firmware behavior is governed by a **finite state machine (FSM)** that defines the logical flow of the application. The main states are:

- **STATE_INIT**: initial state, where all peripherals are configured and the system starts;
- **STATE_WAIT_REQUEST**: waits for the USER button press to start acquisition;
- **STATE_LISTENING**: acquires the analog signal via ADC in polling mode and transmits the values via UART. In this state, three signal modes are calculated and printed:
  - `RAW`: raw ADC value;
  - `MA (Moving Average)`: value filtered through a moving average over a buffer of 150 samples;
  - `NOISE`: ADC value with added random noise to simulate measurement disturbances.
- **STATE_PAUSE**: temporary pause of acquisition, with the green LED blinking;
- **STATE_WARNING**: warning state, activated when the analog value exceeds a predefined threshold (`POT_THRESHOLD`) for an extended period;
- **STATE_ERROR**: error state, activated when the ADC value exceeds the critical threshold (`POT_THRESHOLDERROR`); in this condition, the microcontroller is reset.

State transitions occur through:

- pressing the USER button (detected in polling);
- exceeding the defined thresholds for the analog signal.

2.4 Communication and Output

The processed data is sent in real time through the **UART2** interface and displayed on a serial terminal. The output shows the signal modes and values converted to millivolts, for example:

```
Mode=RAW   | ADC=1082 -> 871 mV
Mode=MA    | ADC=1062 -> 855 mV
Mode=NOISE | ADC=1106 -> 891 mV
```

Additionally, text messages are sent for the `WARNING` and `ERROR` states, and the green LED provides visual feedback on the current system state.

Chapter 3

Implementation

3.1 Overview

The firmware was implemented on the Nucleo C031C6 board using the STM32 HAL libraries. The main goal during development was to integrate analog and digital inputs in real time, ensuring a clear and modular code structure. The system combines polling and interrupt mechanisms for signal acquisition, while a finite state machine (FSM) governs the logical flow of the application.

3.2 Hardware Circuit

The implemented circuit includes:

- **Nucleo C031C6** microcontroller;
- **Potentiometer** connected to ADC1 channel A4 for analog input;
- **Interrupt button** connected in pull-up with a 100kΩ resistor;
- **USER button** for FSM state changes;
- **On-board green LED** for visual feedback;
- power and ground connections.

The following figure shows the implemented circuit:

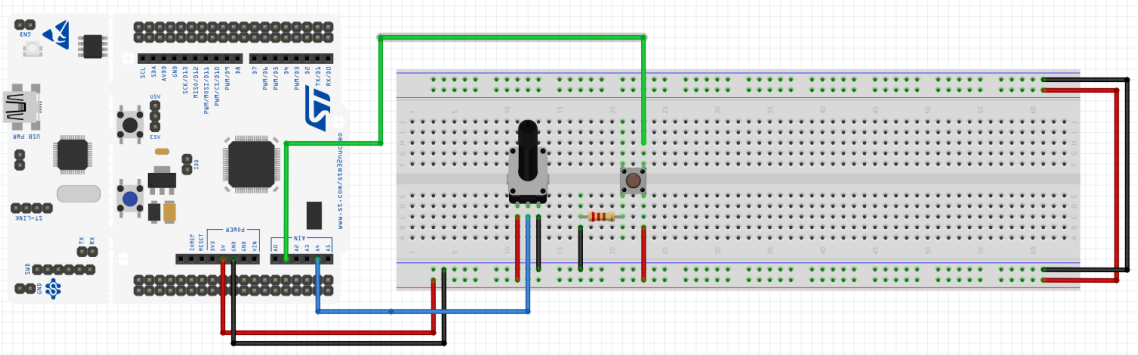

Figure 3.1: Circuit diagram.

3.3 Software Architecture

The firmware is organized into the following main modules:

3.3.1 Finite State Machine (FSM)

The system logic is implemented through an FSM with six main states:

- **STATE_INIT**: initial state, where all peripherals are configured and the system immediately transitions to **STATE_WAIT_REQUEST**;
- **STATE_WAIT_REQUEST**: waits for the USER button press to start signal acquisition;
- **STATE_LISTENING**: acquires the analog signal via ADC in polling mode, calculates three signal modes (RAW, MA, NOISE), and transmits the values via UART;
- **STATE_PAUSE**: temporarily pauses acquisition, with the green LED blinking;
- **STATE_WARNING**: activated when the analog value exceeds `POT_THRESHOLD` for more than 5 seconds;
- **STATE_ERROR**: activated when the ADC value exceeds `POT_THRESHOLDERROR`.

State transitions occur either through pressing the USER button (detected via polling) or when the analog signal exceeds the predefined thresholds.

3.3.2 Analog Signal Acquisition

ADC1 is configured for single 12-bit conversions. Polling is used to continuously acquire the potentiometer value. The firmware calculates:

- `RAW`: raw ADC value;
- `MA (Moving Average)`: average of the last 150 samples using a circular buffer;
- `NOISE`: RAW value with added random noise to simulate disturbances.

All values are converted to millivolts and transmitted via UART2 to a serial terminal. The data are then visualized in real time on graphs, and during continuous acquisition, an acoustic signal is produced.

3.3.3 Digital Event Handling

The USER button is read using polling, detecting the falling edge to change FSM states. Additionally, an interrupt callback for the button (`HAL_GPIO_EXTI_Callback()`) sets a flag when the interrupt button is pressed,

allowing asynchronous event handling. The callback also generates a UART message to confirm the button press.

3.3.4 UART Communication

UART2 is initialized at 115200 baud, 8N1 configuration. All analog values and FSM state messages (WARNING, ERROR) are sent over this interface to provide real-time monitoring.

3.4 Implementation Details

3.4.1 Moving Average and Noise Functions

- `computeMovingAverage()`: updates a circular buffer and returns the average of RAW values.
- `addRandomNoise()`: adds random noise to the RAW ADC value.

3.4.2 FSM Handling

The FSM is implemented in the main loop using a `switch-case` statement. Each state defines:

- LED behavior (ON, OFF, blinking);
- ADC acquisition and value processing (for the LISTENING state);
- state transitions based on button press or thresholds.

3.5 Data Visualization

The acquired values are displayed graphically on three different plots corresponding to the three applied filters:

- **RAW plot**: shows the raw ADC values;
- **MA (Moving Average) plot**: shows values filtered with a moving average;
- **NOISE plot**: shows values with added random noise.

Figure 3.2 shows three screenshots of the graphs obtained during firmware execution.

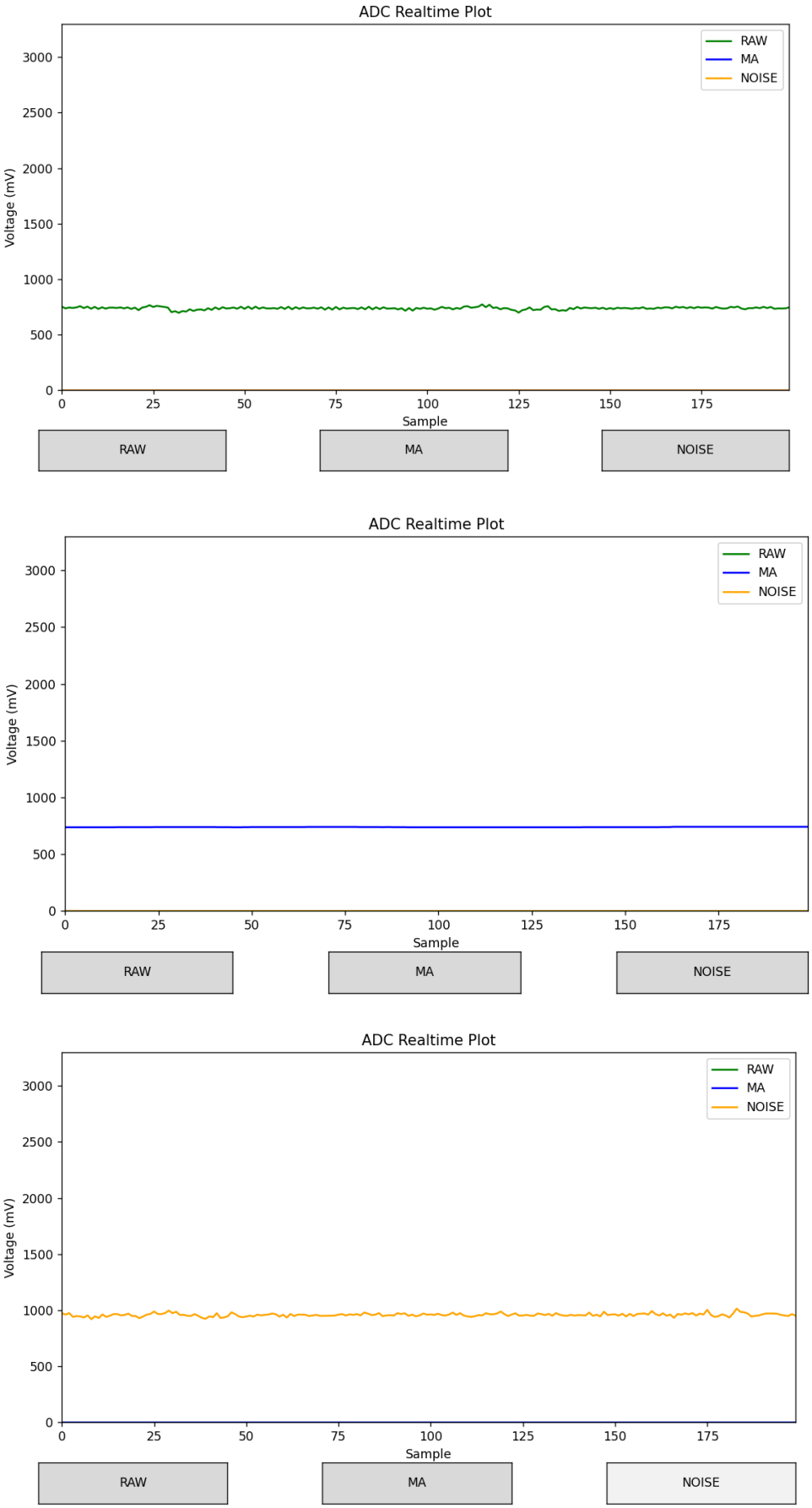

Figure 3.2: RAW, Moving Average, and NOISE plots.

Chapter 4

Problems and Solution

Chapter 5

Conclusion