Fingerprint security system: Final report

Hayden Jin & Erika McCluskey

April 3rd, 2022

ENEL351

For: Robert Martens & Tessa Herzberger

Table of Contents

1.0 Detailed Overview

As a general overview, our final project for ENEL351 models a fingerprint security system. The user is welcomed with a display message on an LCD display. The user presses a button to verify their identity and if they are an authorized user, a buzzer will sound and a green LED will turn on. In order for the user to be authorized, they must make a sound (ex: say their name) while scanning their fingerprint. A sound sensor will detect noise and if there is no noise, the user will not be authorized regardless if their fingerprint is in the system. When the user makes a sound, scans their finger, and is not authorized, a red LED will turn on. In both cases, authorized or non-authorized, in addition to the LED, the user will also see the status displayed on the LCD display.

To operate the system, our system has three buttons, one to verify a fingerprint, one to add a new fingerprint, and one to reset the system. In an ideal world, adding a new fingerprint and resetting the system would only be available to an administrator but given the time constraints and the scope of our project, these functionalities will be available to all users.

2.0 Functionalities

- Has a sound sensor to sense sound when scanning the fingerprint
- Verify a fingerprint
- Add a fingerprint to the system
- Reset the system
- Display status of the scan to the user on an LCD display
- Display the status of the scan to the user via a red / green LED
- Notify the status of a successful scan to the user by sounding a buzzer

3.0 List of user controls and indicators:

3.1 Inputs

- Add user button - Digital input
    - When the add user button is pressed, the user may scan their finger print for it to be stored. If successful or unsuccessful, the user will be made aware by the LCD display and the LEDs.
- Verify user button - Digital input
    - When the verify user button is pressed, the user may scan their finger print for it to be verified. The user must make a consistent sound as they scan their finger so the sound sensor is also activated at the same time. If verification is either successful or unsuccessful, the user will be made aware by the LCD display and the LEDs.
- Reset system button - Digital input
    - When the reset system button is pressed, all fingerprints stored in the system are cleared and the user is made aware by the LCD and LEDs.
- Fingerprint sensor - Digital input
    - The fingerprint sensor reads and verifies fingerprints. It communicates with the microcontroller using USART.
- Red LED - Digital input
    - Used to display status of the system
- Green LED - Digital input
    - Used to display status of the system
- Sound sensor - Analog input
    - Used to detect sound when a fingerprint is being verified. Signal is considered to be high once the sound made reaches a certain level

3.2 Outputs

- LCD screen - Digital output
    - Shows the user the status of the system as well as providing instructions when needed.
- Buzzer - Digital output
    - Notifies users of successful authorization. Uses PWM to communicate with microcontroller

4.0 Block Diagram

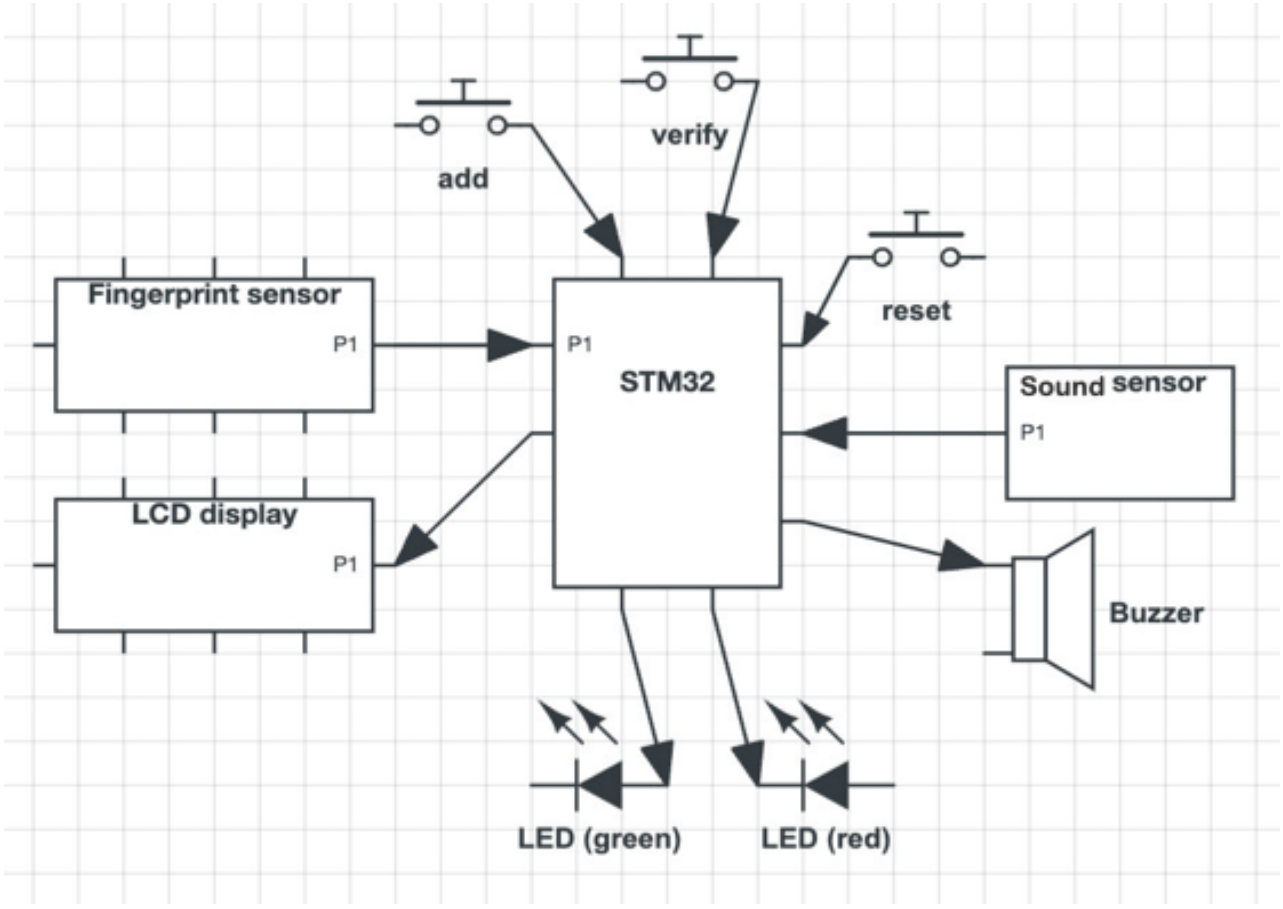

Image 1 - Block diagram

As seen above in image 1, the block diagram shows a simple overview of the system. The block diagram contains major system components and arrows which indicate whether said components are inputs or outputs as viewed from the microcontroller. A more detailed schematic can be found on the separate schematics document.

5.0 List of equipment

- STM32 Cortex M3 Microcontroller
- Two breadboards
- Cardboard box
- Wires
- Fingerprint sensor
- Buzzer
- LCD display
- Three push buttons
- Two LEDs
- Various resistors
- Computer (as a power source)

6.0 Construction and Analysis

Hardware Construction

Construction of the board went fairly well without many issues. This can be attributed to our project being well planned from the start, as it was rare for us to need to rebuild certain circuits. We went with an iterative approach when constructing our project, building and testing each component before starting on the next part. This made errors easy to spot which dramatically reduced the amount of time we needed to spend on fixing bugs. Once all of the individual circuits and parts were complete, we then moved certain components around as necessary to improve the visual look of the device. This included shortening wires, moving components to more accessible areas, and replacing wires to have a common scheme (black: ground, red: power).

Once the system as a whole was tested and working properly. We secured the breadboard components to the cardboard housing, effectively finishing our embedded system.

Code Construction

Similar to the hardware construction, the code construction went relatively smoothly as well. Our approach was to code in the components one by one as it would be easier to test and troubleshoot if we did it like this. We began by initializing the components we were confident with: the LEDs and the push-buttons. The initialization of these components is fairly straightforward as it simply requires enabling the port clocks and configuring the MODE and CNF bits within the GPIO registers to the appropriate values. Once this was complete and working, we initialized the buzzer by enabling the port clocks, setting up the GPIO pins, and setting up the TIM3 timer. The buzzer uses pulse width modulation to operate and therefore requires TIM3 to function. The buzzer does not actually care for the duty cycle percentage it is receiving but rather just cares about whether it is zero or one. If zero, the buzzer is off, if one, the buzzer is on.

Following the LEDs, the push-buttons, and the buzzer, the next thing we worked on was the fingerprint sensor. The fingerprint sensor was definitely our most complex component as it requires a lot of initialization (i.e. uses multiple resources in the MCU). The manufacturer of the sensor provided code to operate the sensor which gave us a huge helping hand when working with it. While all of the fingerprint sensor's initialization was being handled with the HAL libraries, we scrapped most of that code to write our own initialization functions to help us gain a better understanding of what was being used. We started by doing the GPIO pins as this was something simple. The next thing was to configure and enable the appropriate DMA channels to enable the interrupts. Given the fingerprint sensor was using USART, we needed to configure DMA channel 4, for USART1_TX, and 5, for USART1_RX, as these are the two channels necessary for USART1 interrupts. The only thing left to set up in order for the fingerprint sensor to operate was the USART1 configuration. This configuration is being handled by the library functions provided by the sensor's manufacturer. The reason we did not configure this manually was simply due to the fact that the HAL libraries used to configure the USART1 was also configuring a variable of struct data type that was being used in all of the functions necessary to operate the fingerprint sensor (i.e. the library functions). At this point, the fingerprint sensor was up and running.

The next step was to configure the LCD display. The coding for this was relatively simple as the pins were all intialized to general-purpose I/O push-pull 50MHz and the rest of the code was sending data to the GPIO->ODR registers to send commands to the LCD. Once we had the LCD configuration complete, all of our components were configured and operational. The final step was to write the main body to operate the system and this was fairly simple and trivial. The code that operates our system is essentially just a switch statement that handles the cases based on which push-button was pressed. While it would have been easy to throw all of the code into the main body, we separated our files strategically to allow for more readable code. The files are separated as such: configuration code, LCD configuration and operation code, fingerprint sensor library code, command code (ex: turn_on_green_LED(), flash_LED(),

reset_buzzer(), etc), and code that was written with the help of STMicroelectronics (MCU) or Waveshare (fingerprint sensor manufacturer). The most important reason for separating our code from the main body was to make the main more readable. In our opinion, it is incredibly difficult to understand the operation of the system with multiple consecutive lines that are modifying registers. Instead, we opted to write all of the code that addressed registers into functions and give those functions good names; for example, turn_on_green_LED() as opposed to GPIOA->BSRR |= GPIO_BSRR_BS3. The function call is much easier to understand what is going on.

Configuring our components one by one made the process a lot smoother for us as opposed to if we did everything at once since we knew that if our components were no longer working, it had something to do with the component configuration we were currently adding into the code. Getting the proper configuration for all the components was slightly challenging but by tackling them one at a time, it made the process easier and the result was clean code and a working system.

7.0 Testing and Troubleshooting

While working on our project, we ran into several issues which required troubleshooting. The most difficult component that we needed to troubleshoot ended up being the LCD display. The main source of trouble that we encountered was not due to the complexity of the LCD, but more so due to the fact that it was missing the necessary information on its datasheet. Being an older model display, the datasheet we found online did not have the proper pin connections for all 16 pins. Initially, we had several pins unconnected, as we were not sure where they should go. After consulting Tessa and several of our classmates, we learned that not only did the LCD require 2 voltage inputs and 2 ground connections (one for the one board chip and one for the backlight), it also needed the contract pin (V0) to be grounded. This issue caused us many hours of troubleshooting, especially the strange logic of why the contract pin would work at 0v instead of 3.3v. However, once we were able to turn on both the back light and the character display, the rest of the LCD was fairly simple to set up.

Testing was performed in several ways, including unit tests for individual components, physical testing of hardware to ensure proper functionality, and integration tests to ensure the system was running as intended. Unit tests were written to test each component on their own, this helped with the testing of hardware in addition to ensuring individual components were not defective. Having tests for all of our components also helped during debugging, as it allowed us to isolate the bug by testing each component one by one when necessary.

8.0 Conclusion

Drawing from the knowledge gained during the labs performed this semester, we were able to complete our project within a reasonable time frame. Our project incorporated many aspects that were learned just this semester, including GPIO, USART, and PWM. Looking back, it is cool to see how small amounts of knowledge can be combined to create something that resembles a real life electronic device. This project also provided us valuable first time experience in creating a system incorporating both software and hardware. It was great seeing our code interacting with objects in the real world, as our work is often hidden behind a screen. We have no doubt that the lessons learned throughout this course and project will benefit us greatly in both our capstone project and future careers.