RTOS based Password Lock on STM32

Hossain Muhammad Faysal – 28th October 2023

Introduction

In the super-loop (bare-metal) approach of coding a microcontroller, we can get decent outputs from projects where the considerations like concurrency, predictability are not the major concerns. But, as we try to put more functionalities in the code, the overall system responsiveness gets hampered. Again, there are systems where the considerations mentioned should be at the center of the project planning. A security device like password lock system demands for features like faster responsiveness, on-time action. In order to get the features, we introduce an operating system called the Real Time Operating System (RTOS). The RTOS resolves the problems raised by bare-metal coding where the code gets complicated and certain performance criteria are needed to be met. In this project, an introductory level demonstration is presented by building a password lock system using FreeRTOS on the STM32.

Functional Requirements

- User input from a keypad
- Matching the user input with stored password
- Running a servo motor to demonstrate lock/unlock
- Disabling the system for maximum wrong password attempt

Non-functional Requirements

- LCD display, to view lock/unlock status, hidden input, password attempt count, disable message
- LED indicators

Components (hardware)

- Nucleo-F446RE
- 4 x 4 keypad
- Servo motor (SG-90)
- I2C LCD display (16 x 2)
- LEDs
- Resistors etc.

Components & tools (software)

- FreeRTOS
- STM32CubeIDE
- PuTTY

Block Diagram

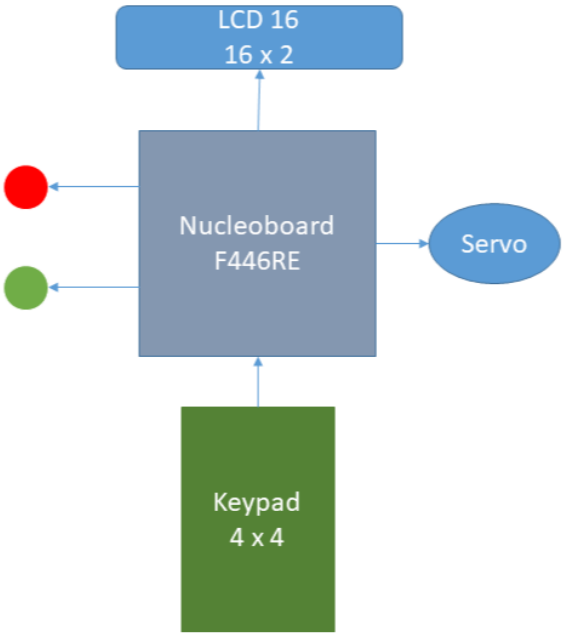

Fig 1: System block diagram

RTOS Overview

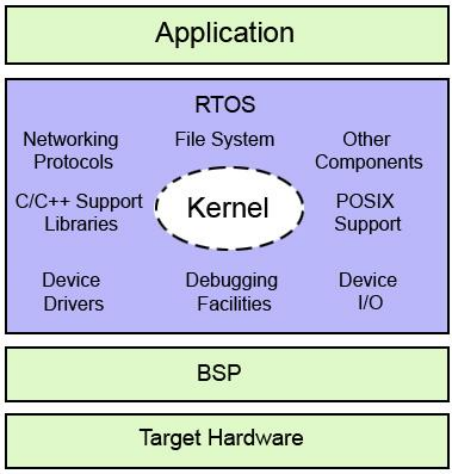

Fig 2: Overview of the RTOS (source: Mepits)

Implementation

According to the functional and non-functional requirements, hardware and software were chosen as components and tools. For coding the microcontroller – firstly, code for individual peripheral was built, then the code snippets were merged together for this project. After that, core logic for the project was coded within the merged code. Furthermore, debugging and fine tuning were performed. Finally, the code was deployed and validated for desired functionalities.

Peripheral configuration

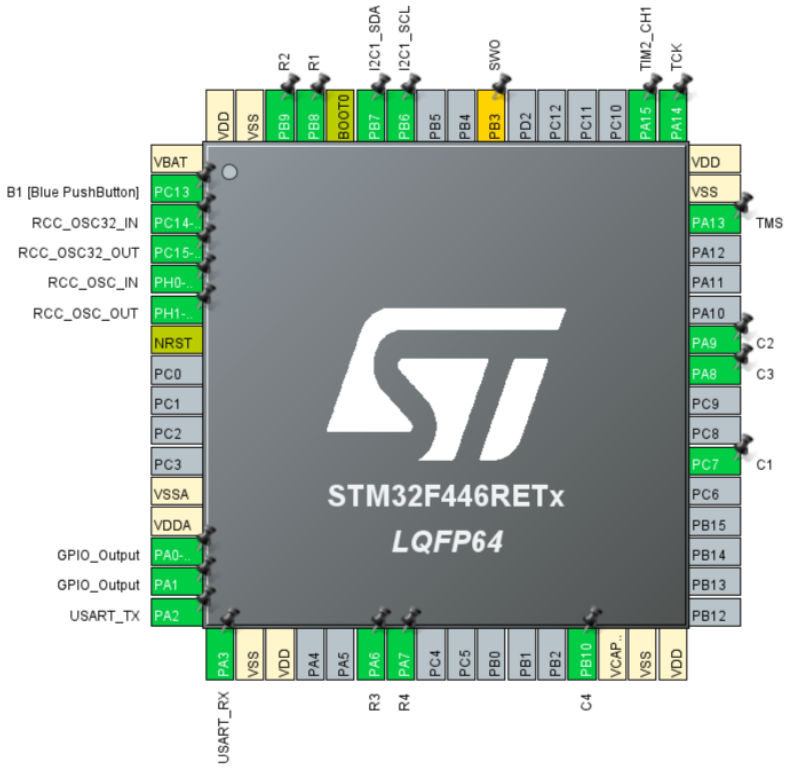

Fig 3: Peripheral configuration in STM32cubeMX

Machine design

State machine approach was used in this codes to switch between different states.

Inputs are – A (as lock button), B (to get enter PIN screen), numeric input (0 to 9) as PIN.

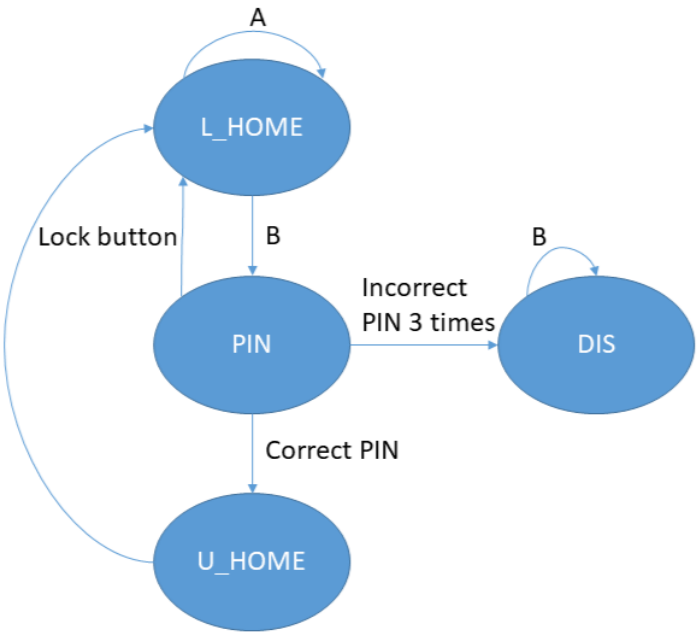

Fig 4: State machine sequence diagram

LCD configuration

I2C1 with 100,000Hz frequency was used. For running the LCD display with RTC, liquidcrystal_i2c.c, liquidcrystal_i2c.h libraries were used. Before setting up the I2C for LCD, an I2C scanner code was run to view the I2C address of the LCD on PuTTY. The address was found to be 0x3F.

Keypad configuration

For the 4 x 4 keypad, the rows were set as inputs with internal pull-ups, the columns were set as outputs. The rows sequentially send signal that is to be detected by the input with the button press.

RTOS configuration

FreeRTOS with 2 tasks were used in the project, named as Task1 and Task2. Task1 had a higher priority than Task2. Task1 was designated to receive keypad inputs and perform logical operations for the system. Task2 mainly was to display outputs and run the LEDs. As operations and inputs are more important, they were set at a higher priority than displaying and indicating.

Servo configuration

For generating PWM, Timer 2 was used. Channel 1 was used to get Timer 2 output.

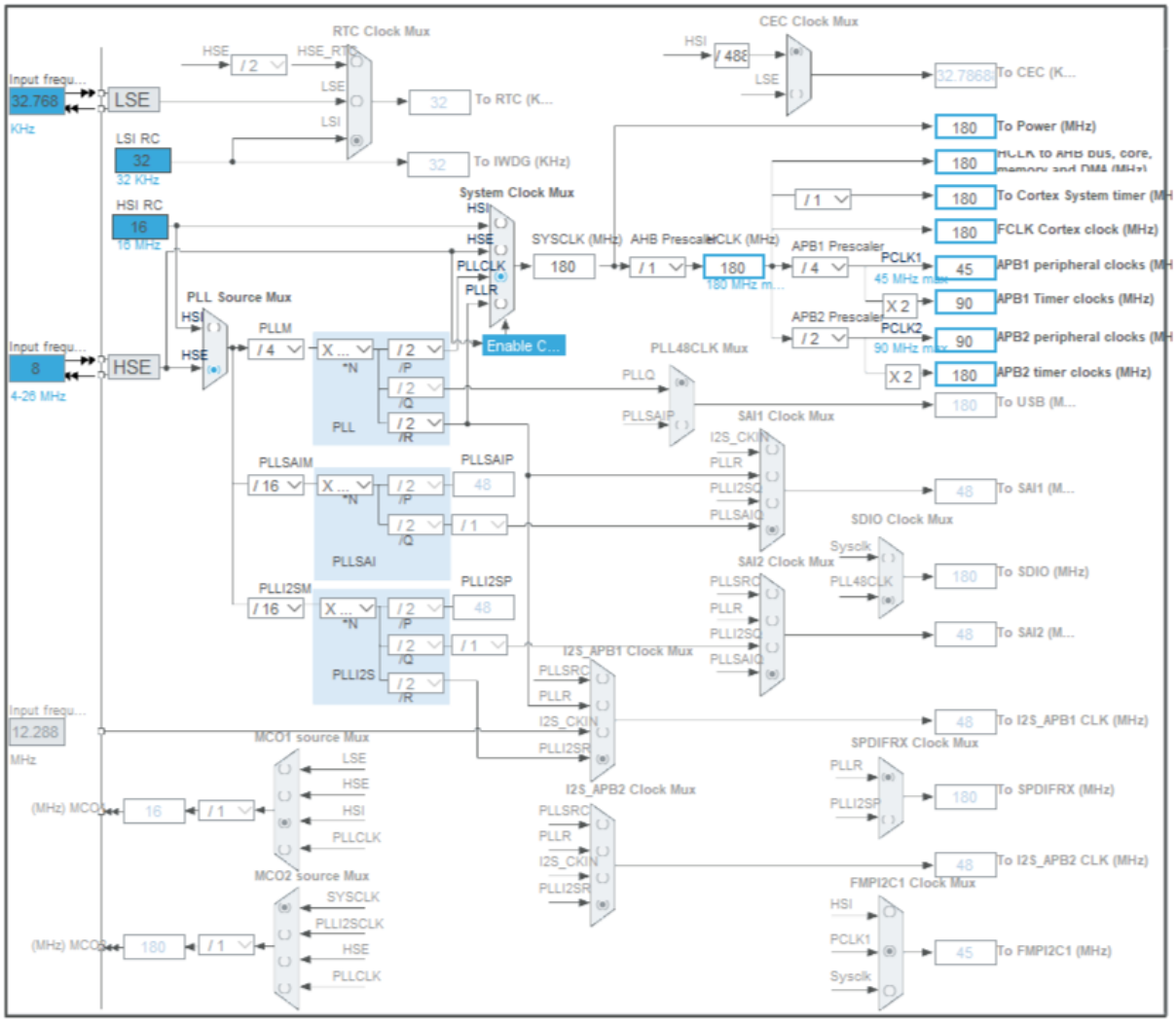

Fig 5: Clock configuration in STM32cubeMX

Servo that is being used uses a 50Hz signal to operate.

And so the following values were chosen,

System frequency = 180MHz

Pre-scaler = 3600

Counter Period = 1000

Setting these values, we get 20ms period for the PWM. Servo changes position with 0.5ms to 2.5ms pulse width. For that the CCR1 values were,

CCR1 = 25 (for 0 degree position)

CCR1 = 125 (for 180 degree position)

Password handling

There was a const char set in the code as set password. The code takes input from the user and stores it in an array, which is simply compared with the preset password.

Keypad layout

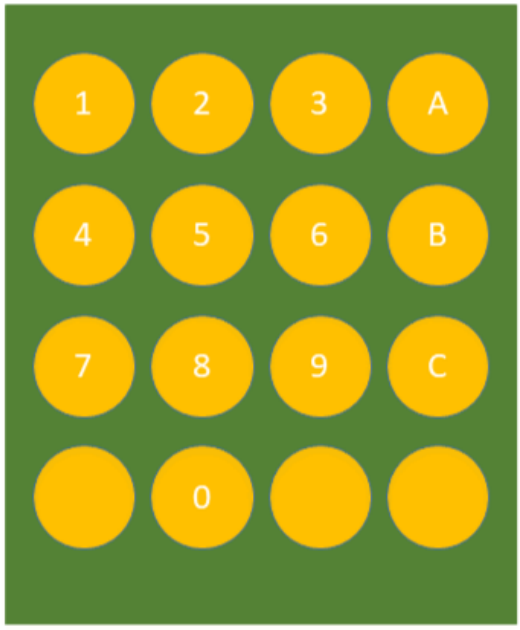

Fig 6: Keypad layout

Here, the numeric keys (0 to 9) are to take PIN input. The A key is the lock button (instant lock). The key B takes the user to the input PIN screen. And, the key C functions as a backspace. Rest of the 3 keys are unused in this project.

Final implementation

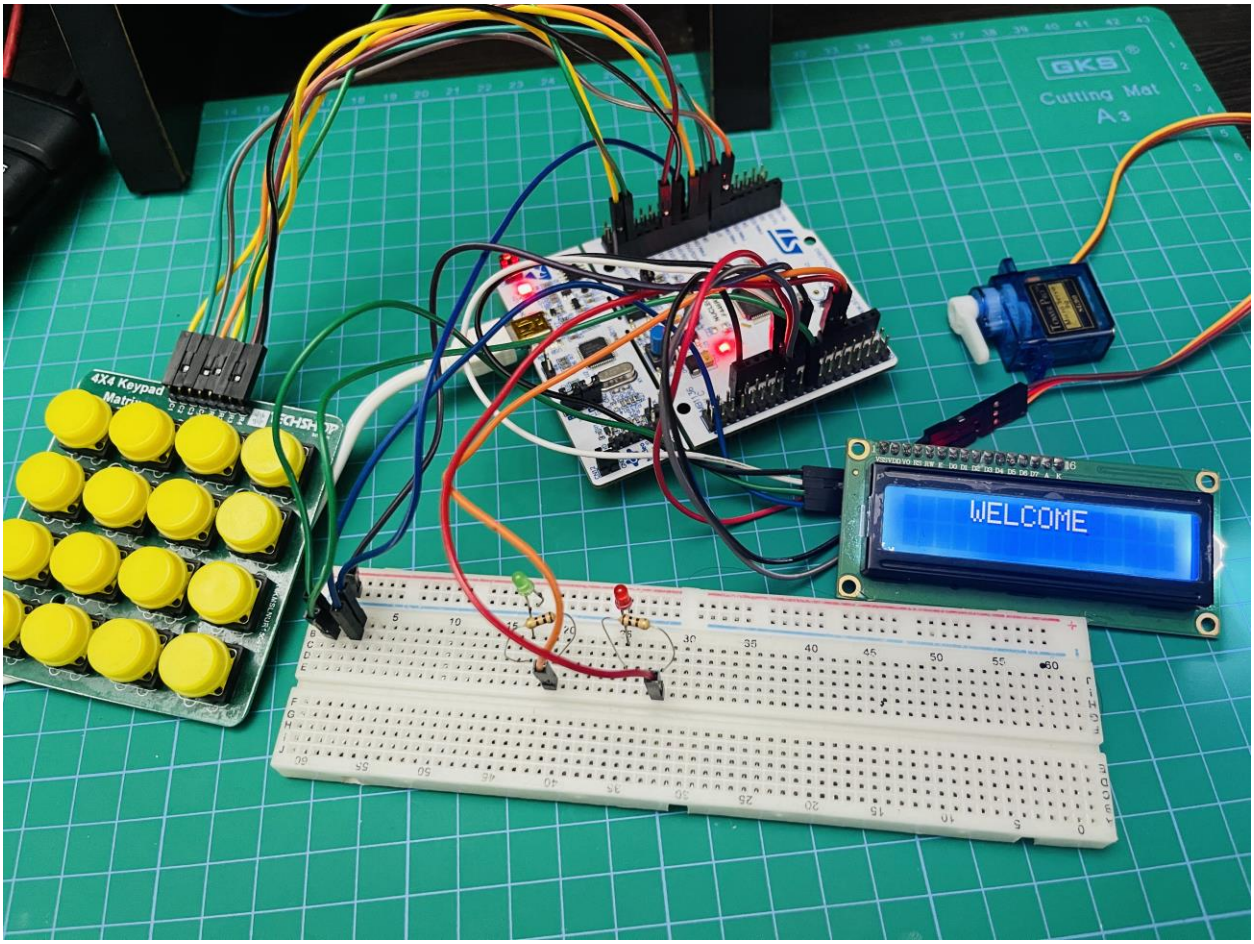

Fig 7: Final implementation of the system

Issues, Debugging

For debugging, printf function was used to view the output within the IDE. Also serial monitor software like PuTTY was used to examine the bugs to be fixed. There were multiple little issues that were fixed; for example – a variable overflow in the code that was causing unwanted behavior, and later was found and fixed. For the keypad, a basic debounce code was put to improve input.

Functionality Validation

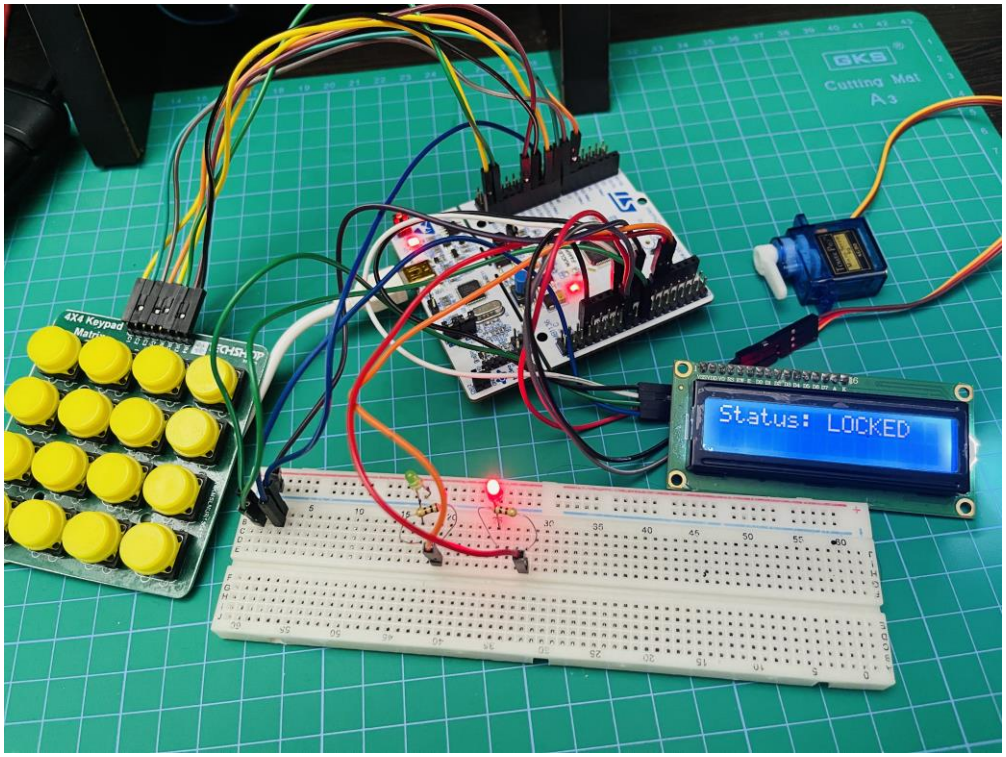

Fig 8: System locked

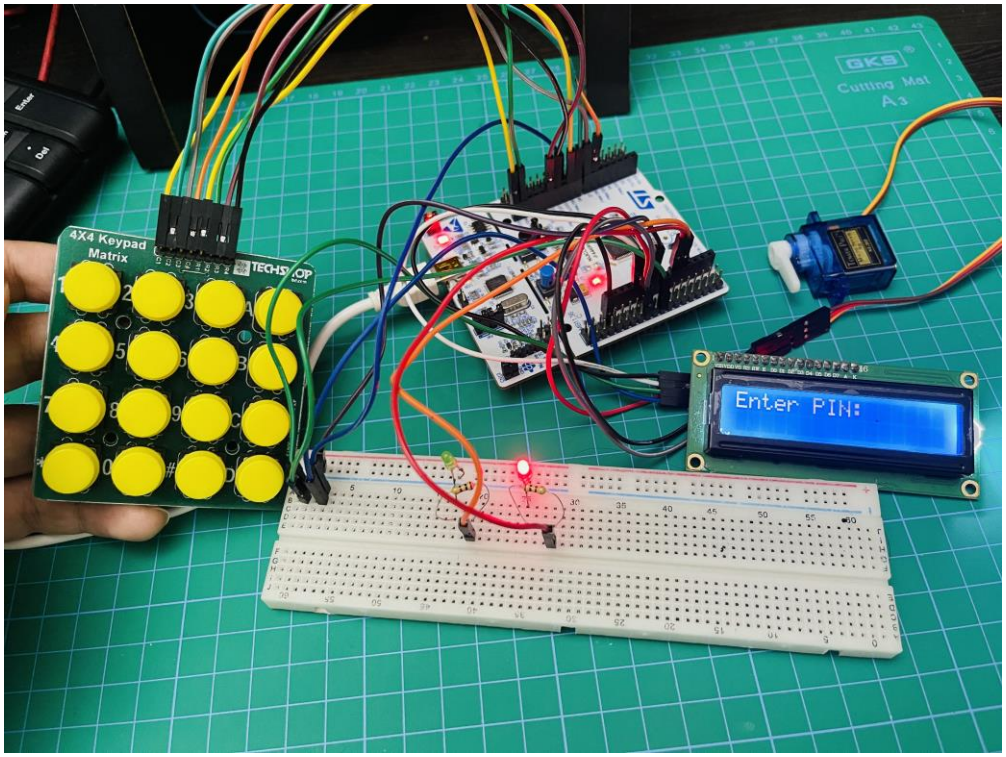

Fig 9: System is asking for PIN

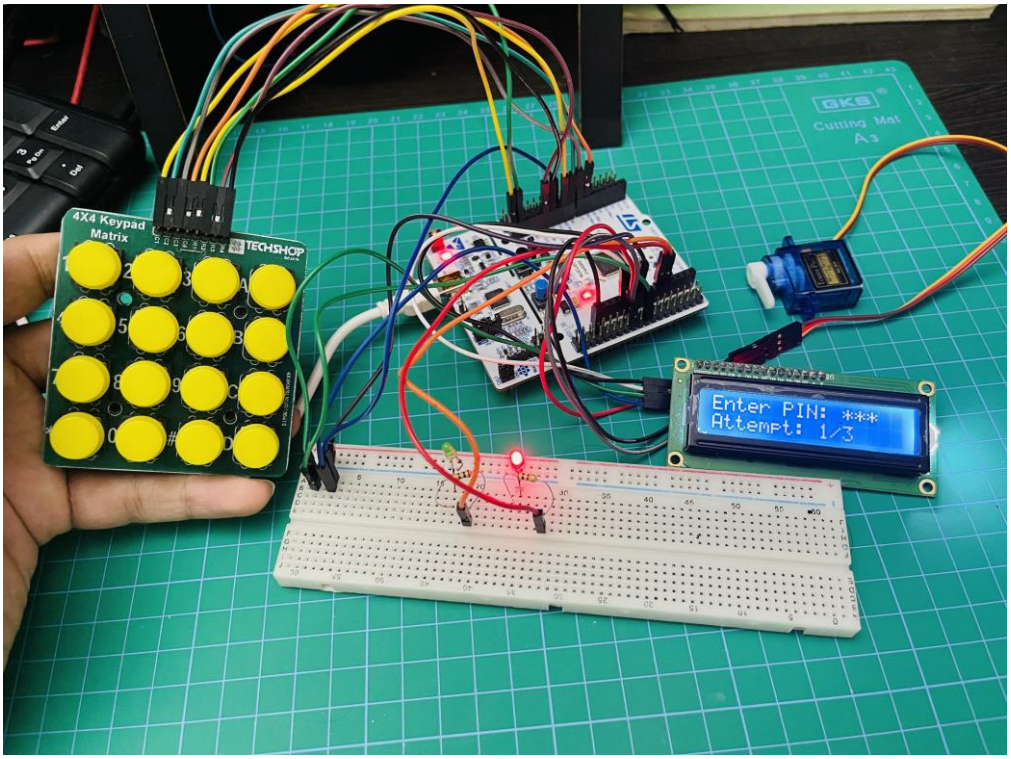

Fig 10: Attempted wrong PIN once

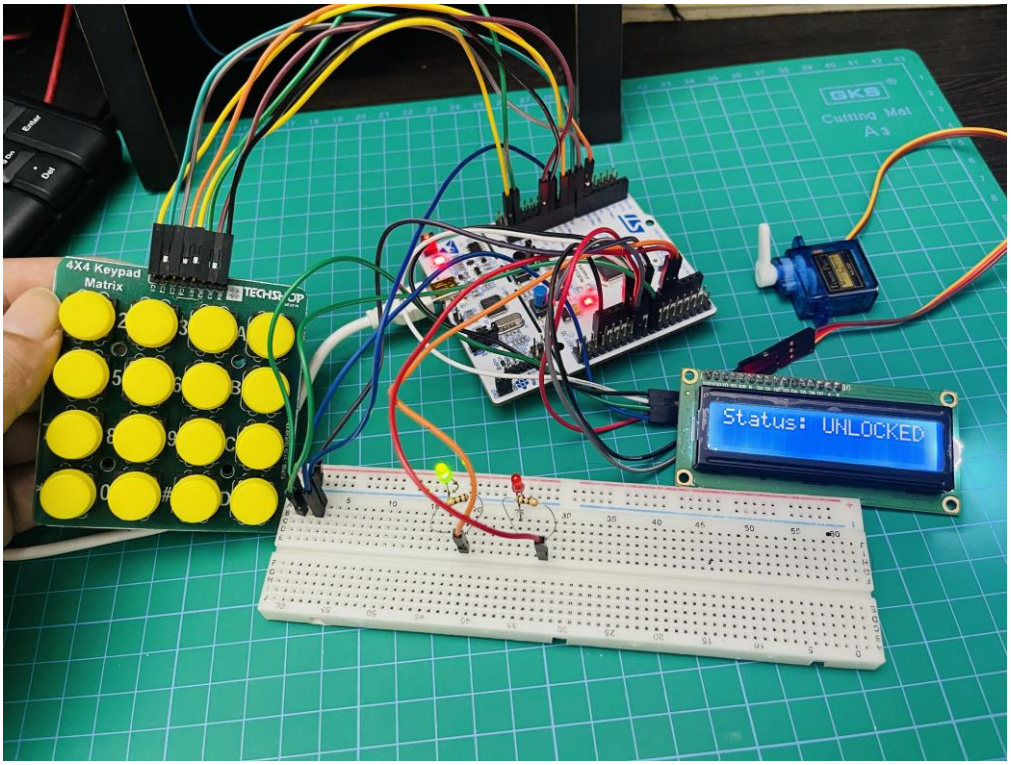

Fig 11: System unlocked

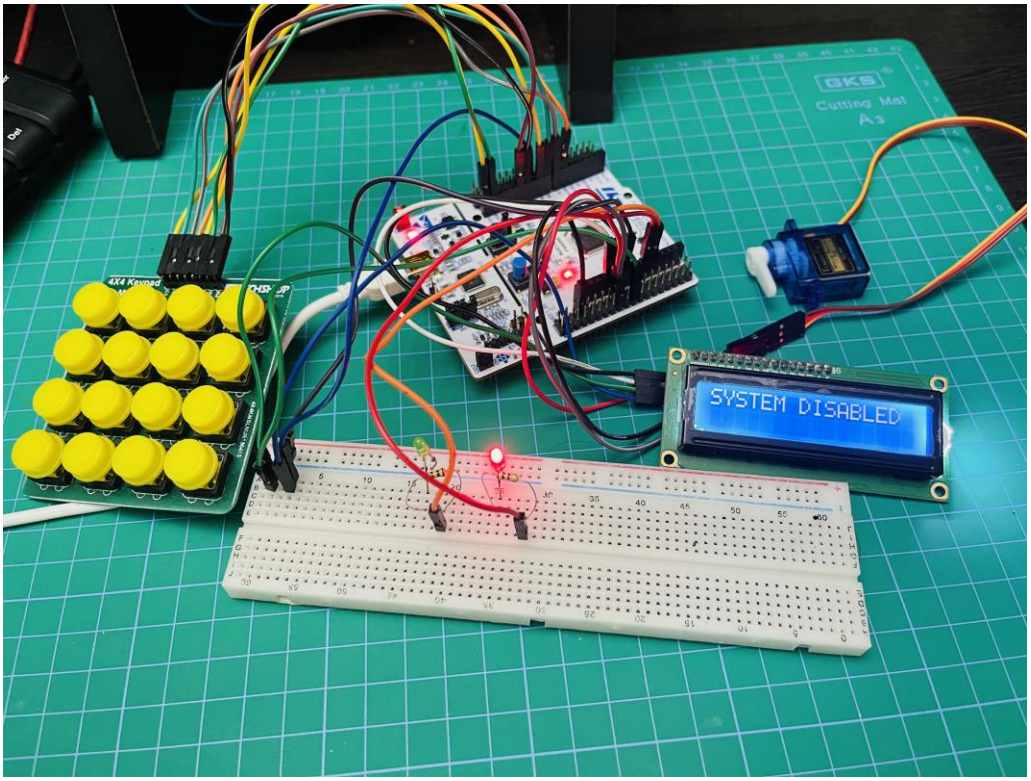

Fig 12: System disabled

Summary

In this simplified password lock project, the use of RTOS is demonstrated with the help of STM32cubeMX within STM32cubeIDE. Deterministic response – quick response with button press, PIN verification, and instant lock with button press were achieved. Also, the system was able to run tasks like – monitoring keypad input with logical performance, displaying information on LCD concurrently. The deterministic response and concurrency was limited or unavailable on the bare-metal approach.

Appendix

Code link