**VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY**

**UNIVERSITY OF TECHNOLOGY**

**----o0o---**


# PROJECT REPORT

## TOPIC:

# CHRONOSYNC – EMBEDDED DIGITAL CLOCK

| | | |
|---|---|---|
| **Subject:** | Embedded System | |
| **Subject code:** | EE3427 | |
| **Instructor:** | MSc. Bui Quoc Bao | |
| **Semester:** | 242 | |
| **Class:** | TT01 | |
| **Group:** | 01 | |
| **Group's member:** | Nguyễn Duy Ngọc | 2251036 |
| | Bùi Đình Trung Nam | 2251032 |
| | Nguyễn Hùng Minh | 2251030 |

**Ho Chi Minh City, May 2025**

# TASK TABLE

| Name | ID | Task name | Contribution | Rating |
|---|---|---|---|---|
| Nguyễn Duy Ngọc | 2251036 | | 100% | Good |
| Bùi Đình Trung Nam | 2251032 | | 100% | Good |
| Nguyễn Hùng Minh | 2051030 | | 100% | Good |

# TABLE OF CONTENTS

# A. INTRODUCTION

## 1. Background and Motivation

Embedded systems play a crucial role in modern digital devices, from consumer electronics to industrial automation. Among these applications, digital clocks represent a fundamental yet essential embedded system that requires precise timekeeping, efficient hardware-software interaction, and user-centric interface design. The development of such a system provides an excellent opportunity to explore real-world applications of microcontroller programming, peripheral interfacing, and power optimization.

The ChronoSyn project was initiated as a comprehensive learning experience to integrate multiple aspects of embedded system design into a cohesive, functional product. The motivation behind this project is threefold:

- **Practical Application of Embedded Knowledge:** By designing a digital clock from scratch, the team is able to apply theoretical knowledge gained from coursework—including real-time constraints, interrupt-driven programming, and low-power techniques—into a tangible product.
- **Exploration of Component-Level Integration:** The project encourages hands-on experience with various hardware modules such as E-Ink displays, real-time clocks (DS3231SN), non-volatile memory (AT24C64D EEPROM), and Bluetooth modules (HC-05). This integration broadens the team's exposure to widely used industry components and communication protocols like $I^2C$, SPI, and UART.
- **Focus on Usability and Expandability:** The inclusion of a mobile app interface using MIT App Inventor reflects the team's emphasis on enhancing user interaction and accessibility. This wireless control interface allows users to configure alarms and time settings remotely, simulating real-world smart device functionality.

Through this project, the team not only aims to build a reliable embedded digital clock system, but also to gain a deeper understanding of multidisciplinary embedded design—ranging from circuit design and firmware development to user experience (UX) and mobile connectivity.

## 2. Project Objectives

The objective of this project is to design and implement a compact, low-power digital clock system – ChronoSyn – based on the STM32F103C8T6 microcontroller. To achieve this, the project is divided into the following structured tasks:

- **Task 1: Component Analysis and Communication Integration**: Study the operating principles and technical specifications of key modules, including the E-Ink display, HC-05 Bluetooth module, DS3231SN real-time clock (RTC), AT24C64D EEPROM, and the STM32F103C8T6 microcontroller. Emphasis is placed on integrating these components using $I^2C$, SPI, and UART protocols to ensure reliable communication between peripherals and the MCU.

- **Task 2: Power Supply Design and Energy Management**: Analyze the operation of boost converters and low-dropout regulators (LDOs) to design an efficient power supply. Explore strategies for reducing power consumption. Select appropriate components and calculate electrical parameters to ensure stability, safety, and long battery life.
- **Task 3: Hardware Design and Enclosure Development**: Design the complete schematic and PCB layout using Altium Designer, integrating all functional blocks such as the MCU, display, memory, and input interface. Fabricate the PCB and develop a custom 3D-printed enclosure that matches the board dimensions, protects components, and enhances usability and aesthetics.
- **Task 4: System Integration and Functional Testing**: Combine the hardware and firmware subsystems into a fully functional prototype. Perform iterative testing to validate feature operation, resolve integration issues, and ensure overall system stability under various conditions.
- **Task 5: Performance Evaluation and Refinement**: Evaluate the system in terms of timing accuracy, responsiveness, power efficiency, and user interface usability. Compare the results with the original specifications, identify limitations, and propose improvements or extensions for future development.
- **Task 6: Mobile Application Development**: Design and implement a companion mobile application using MIT App Inventor. The app communicates with the embedded system via Bluetooth, allowing users to configure time, set alarms, and send control commands wirelessly, thereby enhancing accessibility and interaction.

This comprehensive project not only demonstrates core embedded systems principles, but also highlights cross-platform integration, real-time processing, and power-aware design in a complete, user-friendly product.

## 3. System Specifications

### *3.1. Functional Requirements*

**FR-1: The system must have time display functionality.**

- FR-1.1: The system shall display the current time.
- FR-1.2: The system shall update the displayed time once every second.
- FR-1.3: The displayed format shall include hour, minute, day of week (DoW), date of month (DoM), month, and year in the format HH:MM, DD:MM:YY.
- FR-1.4: The system shall retain real-time data across power loss.

**FR-2: The system must be able to store alarms.**

- FR-2.1: The system shall allow users to configure up to 10 alarms.
- FR-2.2: Each alarm shall consist of: hour, minute, second, DoW or DoM, alarm mode (DoW/DoM/unused), and an ON/OFF flag.

- FR-2.3: The system shall compare current time and stored alarms every second to detect matches.
- FR-2.4: Upon matching, the system shall activate a buzzer and show the alarm status on screen.
- FR-2.5: Alarm configurations shall be retained across power cycles.

**FR-3: The system must allow time and alarm configuration via physical buttons.**

- FR-3.1: The buttons shall support incrementing, decrementing, and navigating between configuration fields.
- FR-3.2: The system shall support both short press and long press detection to accelerate user input.

**FR-4: The system must allow alarm review and editing.**

- FR-4.1: The system shall allow users to browse existing alarms by slot index.
- FR-4.2: The system shall allow users to toggle an alarm's ON/OFF state while reviewing.
- FR-4.3: The system shall support editing a selected alarm.

**FR-5: The system must allow time and alarm configuration wirelessly.**

- FR-5.1: The system shall support a mobile application for wireless configuration.
- FR-5.2: The system shall communicate with the app via bluetooth.
- FR-5.3: The app shall support time synchronization and alarm setup from a smartphone.

**FR-6: The system shall provide system additional features.**

- FR-6.1: System Options shall include:
    - FR-6.1.1: Clear all alarms.
    - FR-6.1.2: Perform factory reset.
    - FR-6.1.3: Run display test pattern.
    - FR-6.1.4: Show distributor information.

**FR-7: The device must be compact, portable, and physically robust.**

- FR-7.1: The device shall have a case made of Mica or ABS plastic for lightweight and durable protection.

**FR-8: The device must support low power consumption and extended battery operation.**

- FR-8.1: The device shall include an E-Ink display to minimize power consumption.
- FR-8.2: The device shall be powered by a rechargeable lithium-ion battery to support long-duration operation.
- FR-8.3: The system shall display the remaining battery level in percentage on the screen.

### *3.2. Constraints and Performance Requirements*

**C-1: Timing and Accuracy**

- C-1.1: The system shall ensure time deviation is less than ±3 ppm at room temperature.
- C-1.2: Alarm matching logic shall be executed within 50 ms per main loop iteration.
- C-1.3: On power loss, the system shall maintain accurate real time for at least 30 days.

**C-2: Alarm Storage**

- C-2.1: The system must retain all alarm data across power loss.
- C-2.2: Total memory usage for alarm storage shall not exceed 50 bytes.
- C-2.3: Storage read/write operations shall complete in less than 10 ms.

**C-3: Input Responsiveness**

- C-3.1: A short button press shall be detected with latency under 30 ms.
- C-3.2: A long button press shall be registered after holding the button for 600 ms.
- C-3.3: Button debouncing shall ensure fewer than 1 false trigger per 100 presses.

**C-4: Power and Energy Constraints**

- C-4.1: The system shall operate on four 2000 mAh, 3V lithium-ion batteries.
- C-4.2: The system shall function for at least 90 days on a full charge in idle conditions.
- C-4.3: The system shall consume less than 3 mW during typical operation.
    - C-4.3.1: The system shall perform periodic operations (e.g., real-time updates, alarm checks, battery polling) once per second to minimize power usage.
    - C-4.3.2: The display shall consume no more than 0.02 mW in all operating modes.
        - C-4.3.2.1: The display refresh rate shall be limited to 1 Hz when displaying real time, and shall only refresh on value changes in other modes.

**C-5: Physical Dimensions**

- C-5.1: The overall device dimensions shall not exceed 15 cm × 15 cm × 8 cm.
- C-5.2: The total weight of the device shall be less than 300 g.

## B. MATERIALS AND METHODS

### 1. List of components

| No. | Component | Quantity | Function |
|---|---|---|---|
| 1 | DC5521 Power Jack 1A | 1 | Standard DC input for external power supply or charger. |
| 2 | SS54 Schottky Diode 5A 40V | 1 | Drops the input voltage before the inductor in the buck converter. |
| 3 | SMD CDRH127 Inductor 68µH 3.2A | 1 | Enables current regulation and energy storage in the buck converter circuit. |
| 4 | TPS54231DR Buck Converter | 1 | Steps down 12V input to lower voltages (5V) efficiently for system operation. |
| 5 | AMS1117 Regulator 3.3V 1A | 1 | Provides a stable 3.3V output from the 5V rail to power the STM32 and other 3.3V devices. |
| 6 | SMD Capacitor 100µF | 4 | Used for power decoupling and smoothing voltage across supply rails. |
| 7 | TAJC476K016RNJ 2312 Tantalum Capacitor 47uF | 2 | Acts as a local bypass capacitor to stabilize voltage and reduce noise near sensitive ICs. |
| 8 | STM32F103C8T6 Microcontroller | 1 | Main controller managing timekeeping, UI, display, and communication. |
| 9 | SMD HC49-S Crystal Oscillator 8MHz | 1 | Provides system clock for STM32 core. |
| 10 | FC-135 3215 Crystal 32.768kHz | 1 | Provides a low-frequency clock source option for the STM32 core. |
| 11 | DS3231SN RTC Module | 1 | High-accuracy real-time clock with backup capability. |
| 12 | AT24C64D-SSHM-T EEPROM 64Kbit | 1 | Stores alarm settings and user configuration permanently. |
| 13 | Tactile Buttons 12 x 12 mm | 5 | Buttons for user input, each with short/long press detection. |
| 14 | SMD 0805 Resistor 10kΩ | 5 | Pull-up/down resistors for buttons and logic-level conditioning. |
| 15 | SMD 0805 Capacitor 100nF | 5 | Used for hardware debouncing of push buttons by filtering out signal noise and eliminating false triggers. |
| 16 | Waveshare E-Ink Display 2.9" | 1 | Low-power, high-contrast display for time and status. |
| 17 | SMD 0805 LED | 3 | Status indicators for 5V rail, 3.3V rail, and the PC13 GPIO pin. |
| 18 | 9056-TS Buzzer 80dB | 1 | Emits sound when an alarm is triggered. |
| 19 | MMBT2222A NPN Transistor 40V 0.6A | 1 | Used as a low-side switch to drive current to the buzzer. |

| | | | |
|---|---|---|---|
| 20 | HC-05 Bluetooth Module | 1 | Enables wireless setup and control via smartphone app. |
| 21 | 2-Pin Female XH2.54 Cable | 5 | Used to easily connect buttons to the PCB. |
| 22 | 2-Pin Male Header XH2.54 | 5 | Mates with XH2.54 cables to provide sturdy, pluggable connections for signals on the PCB. |
| 23 | 8-Pin Male Header XH2.54 | 1 | Used for display signal interfaces (SPI). |
| 24 | 8-Pin Female XH2.54 Cable | 1 | Paired with XH2.54 headers for SPI interface. |

## 2. Selection Criteria and Justifications

The components selected for the ChronoSyn embedded digital clock were chosen based on critical engineering criteria: functional reliability, power efficiency, cost-effectiveness, compactness, and ease of integration. Each category below outlines the rationale for component selection in terms of system requirements and performance optimization.

### *2.1. Microcontroller Unit (MCU)*

**Component**: STM32F103C8T6 (ARM Cortex-M3, 72MHz)

**Justification**:

- Offers sufficient computational power to handle real-time clock tracking, alarm scheduling, display control, and Bluetooth communication.
- Provides a wide range of peripheral interfaces: $I^2C$ (RTC, EEPROM), SPI (E-Ink), UART (Bluetooth).
- Low power consumption and supported by STM32CubeIDE and HAL drivers for efficient firmware development.

### *2.2. Power Supply*

**Components**: TPS54231 Buck Converter, AMS1117-3.3V Regulator, SS54 Schottky Diode, CDRH127 Inductor, 100µF Electrolytic Capacitors, TAJC476K016RNJ 47µF Tantalum Capacitors

**Justification**:

- TPS54231 is a high-efficiency step-down converter capable of supplying 2A output current, ideal for converting 12V DC input to 5V with minimal power loss.
- AMS1117-3.3V linear regulator provides a stable 3.3V output necessary for core system components such as the STM32, RTC, EEPROM, and Bluetooth module.
- The SS54 Schottky diode ensures low forward voltage drop and fast switching, improving conversion efficiency and protecting against reverse current.

- The SMD CDRH127 inductor (68µH, 3.2A) stabilizes current flow through the buck converter, reducing voltage ripple and electromagnetic interference.
- Electrolytic and ceramic SMD capacitors are placed on power lines to suppress voltage spikes, smooth output, and reduce noise coupling.
- TAJC476K016RNJ Tantalum capacitors (47µF) act as bypass and filtering elements, ensuring voltage stability during transient loads and providing local energy storage near the regulators.

### *2.3. Oscillators and Timing Components*

**Component**: 8MHz Crystal (STM32 system clock), 32.768kHz FC-135 Crystal (RTC backup)

**Justification**:

- 8MHz crystal ensures stable and accurate operation of the STM32 MCU.
- 32.768kHz crystal enhances timekeeping accuracy and is compatible with low-power RTC operation.

### *2.4. Display System*

**Component**: 2.9-inch Waveshare E-Ink Display

**Justification**:

- Power-efficient as it only draws current during screen refresh.
- Excellent visibility in ambient light; ideal for static time display.
- Communicates over SPI with STM32, reducing pin usage and enabling fast updates.

### *2.5. Timekeeping*

**Component**: DS3231SN RTC Module

**Justification**:

- Temperature-compensated crystal oscillator for ± 2 ppm accuracy.
- Maintains real-time across power cycles using coin-cell backup.
- Communicates over I²C with minimal code overhead.

### *2.6. Alarm Storage*

**Component**: AT24C64D EEPROM (64Kbit)

**Justification**:

- Non-volatile memory ensures alarm data persists through power loss.
- Fast I²C communication with write cycles well within application constraints.

- 64 Kbit provides ample space for storing up to 10 alarm configurations, along with headroom for future feature upgrades or additional system data.

### *2.7. User Interface – Buttons and Indicators*

**Component**: Tactile Push Buttons (12×12mm), SMD LED 0805 (Power Indicators), 100nF Capacitors, SMD 0805 Resistor 10kΩ

**Justification**:

- Buttons provide a reliable interface for configuration and navigation. Their size ensures usability.
- Dual short/long press recognition expands functionality without additional components.
- SMD LEDs provide visual feedback for system states (5V, 3.3V, heartbeat via PC13 pin).
- 100nF ceramic capacitors and 10 kΩ resistors are used for hardware debouncing of button signals.

### *2.8. Audio and Output Control*

**Components**: 9x4.2mm 80dB Buzzer, MMBT2222A NPN Transistor

**Justification**:

- The buzzer provides audible feedback for alarm activation.
- The MMBT2222A acts as a current driver, switching the buzzer on/off through a GPIO pin safely without overloading the MCU.

### *2.9. Wireless Communication*

**Component**: HC-05 Bluetooth Module

**Justification**:

- Enables wireless communication with a custom mobile app developed using MIT App Inventor.
- UART-based interface simplifies integration with STM32.
- Supports bidirectional control of time and alarms from mobile devices.

### *2.10. Connectors and Wiring*

**Component**: XH2.54 Bus Cables (2-pin and 8-pin), XHB-2A Connectors

**Justification**:

- These connectors simplify module interconnection and allow easy debugging or module replacement.

- Standardized 2.54mm pitch ensures compatibility with PCB and peripheral modules.

## 3. Development Tools

The development of the ChronoSyn embedded system leverages a comprehensive suite of software and hardware tools for circuit design, firmware development, simulation, and debugging. Each tool was selected based on compatibility with the STM32 platform, support for industry standards, and community resources.

### *3.1. Hardware Development Tools*

- **Altium Designer**
  - **Purpose**: Used for schematic capture and PCB layout design.
  - **Justification**: Professional-grade EDA tool with precise control over routing, layout constraints, and multi-layer design, enabling high-reliability circuit boards suitable for compact embedded systems.
- **Soldering Station & Solderer**
  - **Purpose**: Manual soldering of surface-mount and through-hole components.
  - **Justification**: Required for assembling the custom-designed PCB, especially for SMD parts like STM32F103C8T6 and EEPROM.
- **Digital Multimeter & Oscilloscope**
  - **Purpose**: Testing power lines, signal integrity, and time-domain behavior of waveforms (e.g., SPI, UART, PWM for buzzer).
  - **Justification**: Essential for debugging hardware-level faults and verifying timing constraints.

### *3.2. Firmware and Embedded Software Tools*

- **STM32CubeIDE**
  - **Purpose**: Main integrated development environment (IDE) for writing, compiling, and debugging C code on the STM32 platform.
  - **Justification**: Official STMicroelectronics toolchain supporting STM32F1 series with STM32CubeMX integration, HAL drivers, and peripheral configuration GUI.
- **STM32CubeMX**
  - **Purpose**: Peripheral and clock configuration tool with automatic code generation.
  - **Justification**: Simplifies hardware abstraction and generates consistent initialization code, reducing development errors.
- **ST-Link Utility/OpenOCD**
  - **Purpose**: Flashing and debugging firmware via ST-Link.

- **Justification**: Enables low-level programming and real-time debugging with breakpoint and register inspection capability.

### *3.3. Mobile Interface Tools*

- **MIT App Inventor**
  - **Purpose**: Rapid development of the companion mobile app.
  - **Justification**: Offers a block-based visual programming environment suitable for prototyping Bluetooth-based UI without deep Android development expertise.

### *3.4. Version Control and Documentation*

- **GitHub**
  - **Purpose**: Source code management, issue tracking, and collaboration.
  - **Justification**: Facilitates transparent teamwork and history tracking. All design files, firmware, and documents are hosted at: https://github.com/ndNgoc1310/Embedded-System-Project
- **Microsoft Word**
  - **Purpose**: Report writing and formatting.
  - **Justification**: Allows structured documentation of the development process and output files suitable for academic and professional submission.

## 4. Programming and Debugging Methodology

### *4.1. Programming Structure and Naming Convention*

The firmware is developed using the C programming language, structured into modular units with clear separation between application logic and low-level hardware interaction. This is achieved through a multi-layered architecture, including:

- **Application Layer**: Handles system state logic and mode transitions.
- **Middleware Layer**: Manages data encoding/decoding, EEPROM access, and RTC time parsing.
- **Driver Layer**: Interfaces with external peripherals such as the RTC (DS3231), EEPROM (24C32), buttons, buzzer, and E-Ink display.

All functions and variables strictly follow a unified naming convention. Prefixes are assigned according to module ownership, ensuring clarity and preventing namespace collisions. This convention included:

- camelCase for local variables,
- lowercase_with_underscores for global variables and struct members,
- Upper_Camel_Snake_Case for function names, and
- ALL_CAPS_WITH_UNDERSCORES for macros and constants.

This structured organization and naming standard promote code readability, maintainability, and ease of collaboration among team members.

### *4.2. Debugging Strategy*

A disciplined debugging approach was followed throughout development. Using STM32CubeIDE, the following techniques were employed:

- **Live Expressions Monitoring**: Temporary and persistent state variables were tracked in real time to understand logic flow and diagnose anomalies.
- **Breakpoints and Step Execution**: Specific sections of code were paused and stepped through to trace faults and validate assumptions.
- **Hardware-in-the-Loop Testing**: Each update was tested on real hardware to verify actual peripheral behavior and detect timing or synchronization issues not visible in simulation.
- **Modular Testing**: Subsystems (e.g., EEPROM access, time sync, buzzer logic) were individually validated before full integration.

This methodology ensured high firmware reliability, fast bug localization, and efficient conflict resolution across hardware and software boundaries.

## C. SYSTEM DESIGN

### 1. System Architecture Overview

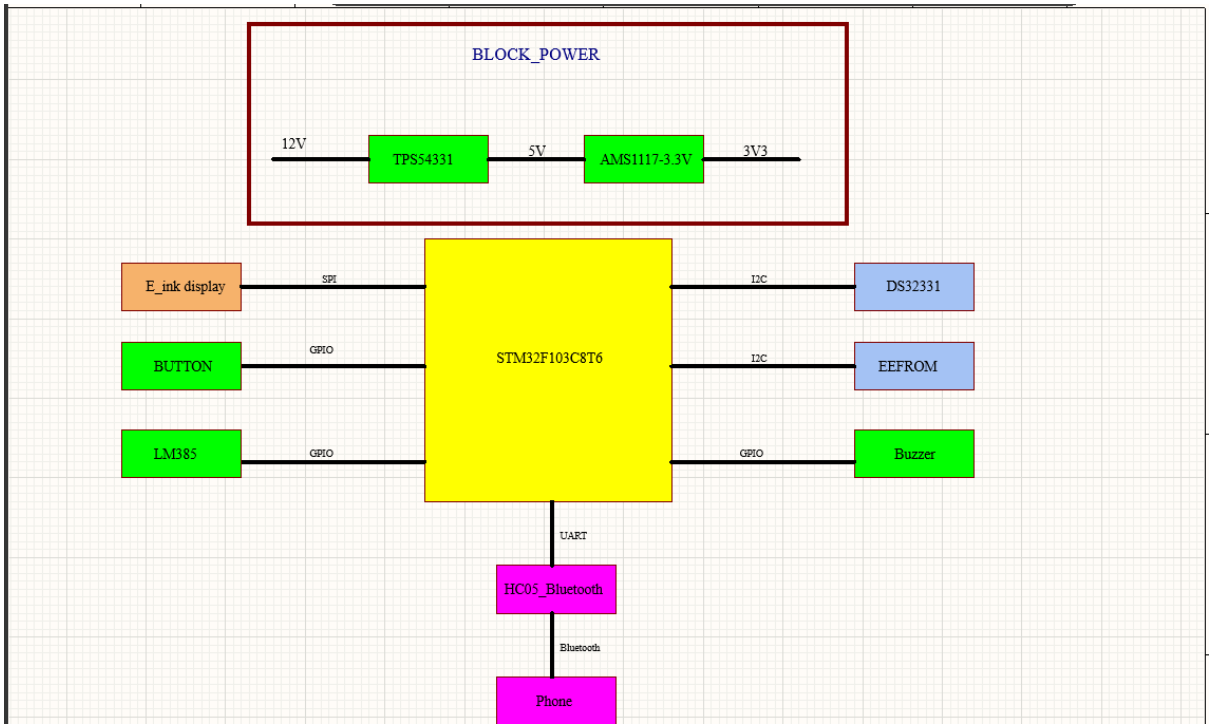



**Figure 1.** System overview

**Phone**

- Task: Communicate with the system via Bluetooth connection.
- Function: User interaction.

**HC05 BLE (Bluetooth Low Energy)**

- Task**:** Wireless connection with the phone via Bluetooth.
- Function:
  - Receive signals from the phone (e.g., time adjustments).
  - Transmit data to the STM32 microcontroller via UART.
  - Bidirectional communication between the phone and MCU to update clock status or adjust time

**IC TPS54331**

- Task: Step down the voltage from 12V to 5V.
- Function: Step down the voltage to 5V to supply BLE and to step down 3V3 through the AMS1117.

**IC AMS1117**

- Task: Step down the voltage from 5V to 3.3V.
- Function: Supply voltage to STM32F103C8T6, E-ink-display and other devices.

**STM32F103C8T6 Microcontroller**

- Task: Control the entire system’s operation.

- Function:
    - Process data received from the BLE module
    - Control the E-Ink display to show time.
    - Process the signal from the buttons.
    - Read and write the data from the DS3231 and EEFROOM
    - Manage power from the boost, buck converters, and backup battery.

**IC DS3231**

- Task: Store the data of time.
- Function: Read and write the time data.

**EEFROOM**

- Task: Read and store data.
- Function: To store the data alarm.

**IC LM385**

- Task: The op amp to amplified the current.
- Function: Like the buffer to measure the voltage.

**Buzzer**

- Task: Generate audible alerts or notifications in response to specific events or conditions within the system.
- Function: Notify the alarm.

**E-ink display.**

- Task: Display the information for the users.
- Function: Display time and setting.

## 2. Hardware Design

*a) Block diagram*

**Power supply**

- In accordance with the design requirements, the product's power consumption should be minimized. Therefore, the team has decided to use the TPS54331 IC (buck DC converter) to step down the voltage to 5V, followed by the ASM1117 to further step down to 3.3V.
- Additionally, the power input will have two voltage sources: one from the adapter and another from the battery.

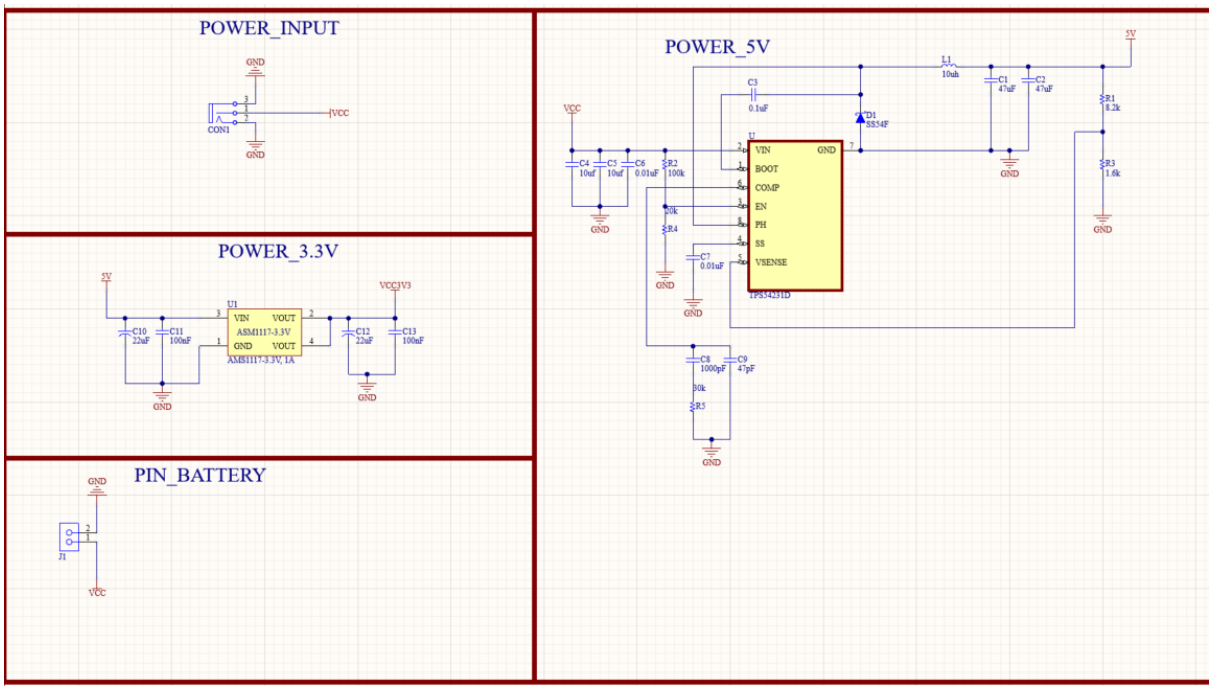



**Figure 2.** Power block.

## MCU block

- Using STM32f103c8t6 is the microprocessor. STM32f103c8t6 use the crystal 8MHz.
- The MCU block has the role of receive the data from the other devices like RTC, HC05, extra and control the E-ink display rightly.

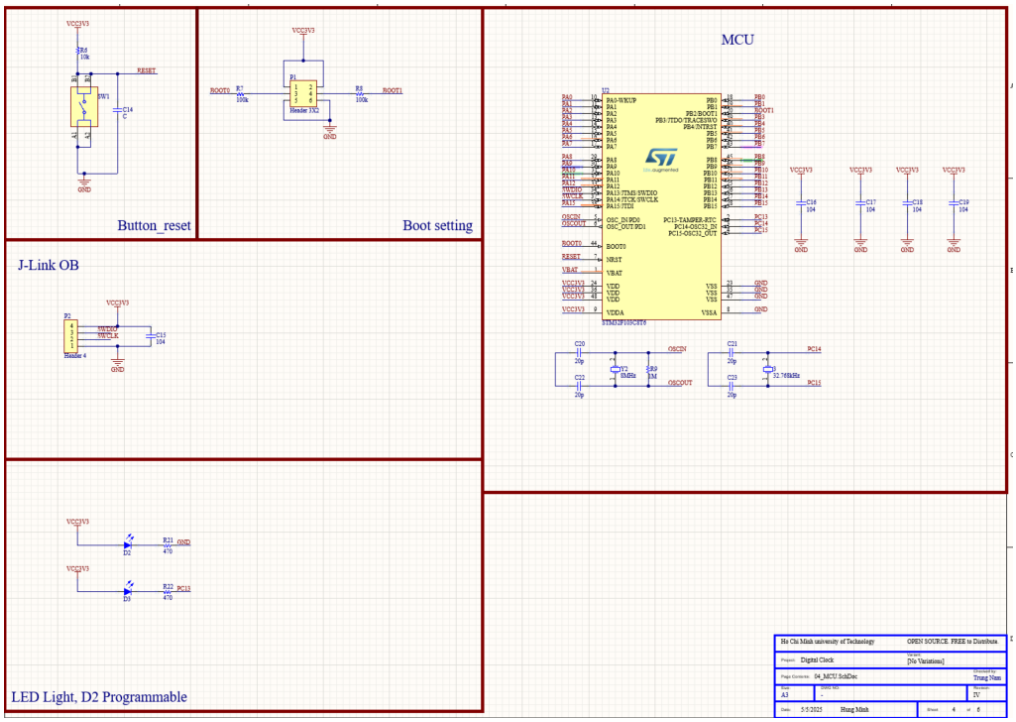



**Figure 3.** MCU block.

## GPIOs block

- The block describes all the connect GPIO in the product, it include:
  - Button control E-ink.

- The IC LM385 to measure the Pin battery.
- The GPIO to control the buzzer.

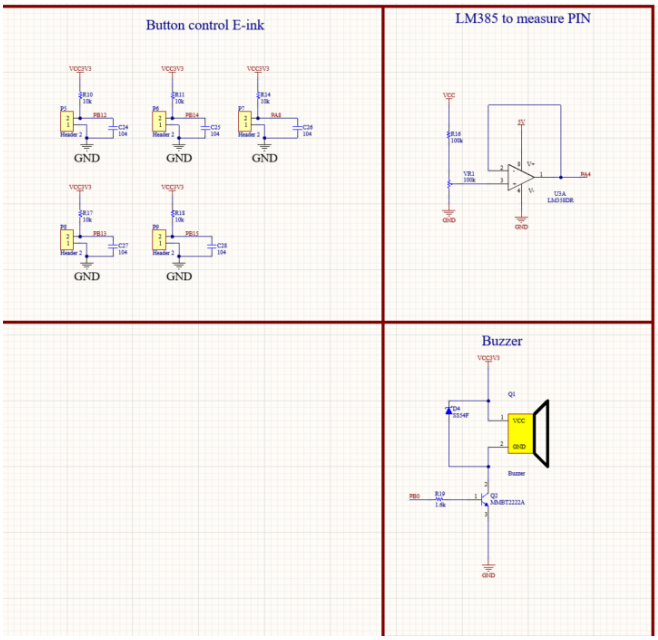



**Figure 4.** GPIOs block.

## Communication Protocols block

The block describes all the connect of protocol in the product:

- The EEFROM will be used by AT24C family and we will implement through the I2C. The same with RTC DS32331 is transferred the data through the I2C.
- The data of module HC-05 will be transmitted by UART while E-ink is used SPI protocol.

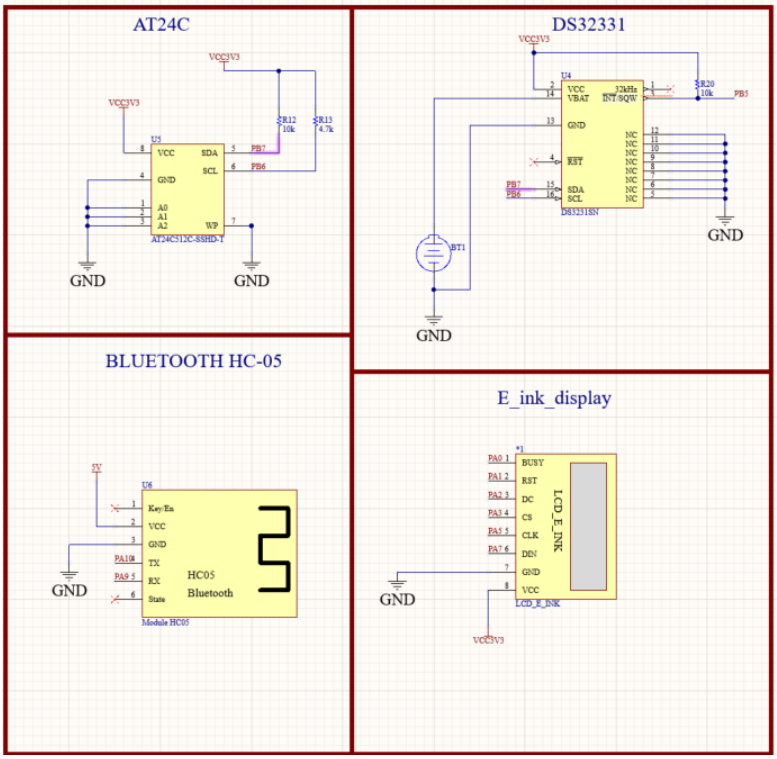



**Figure 5.** Communication Protocol block.

**Layout PCB**

The product is the PCB with 2 layers and the size is 10x10cm.

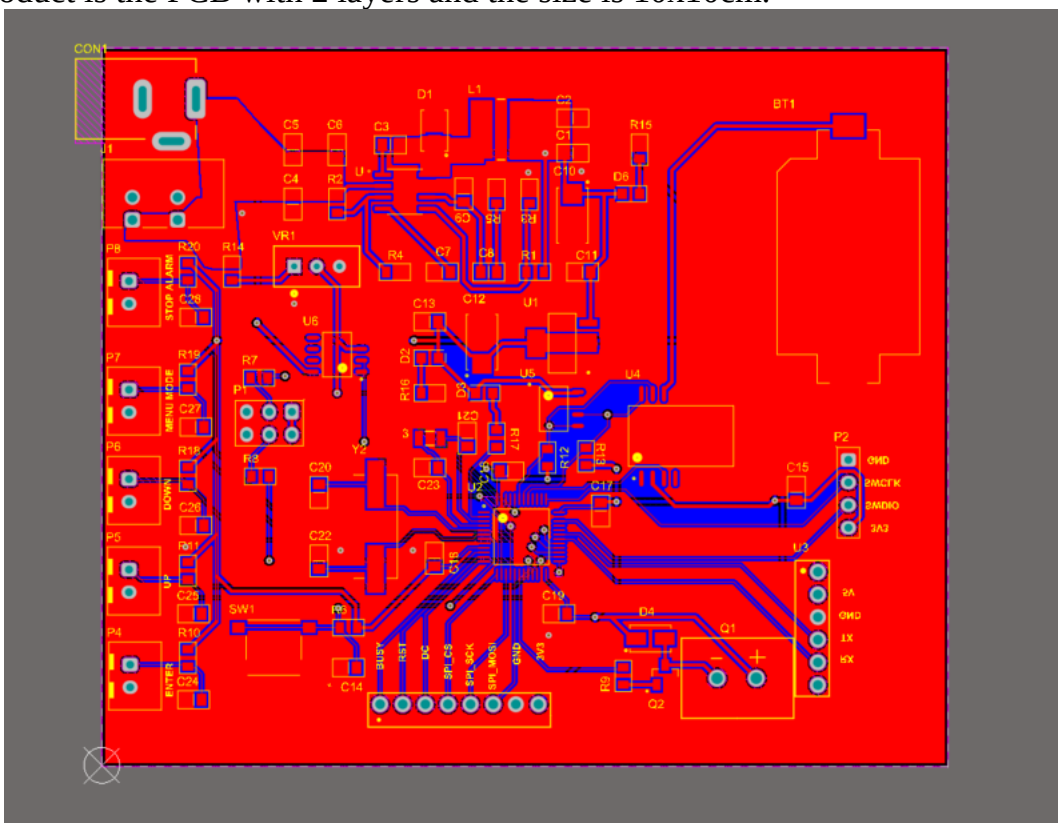


**Figure 6.** The view 2D of the circuit.

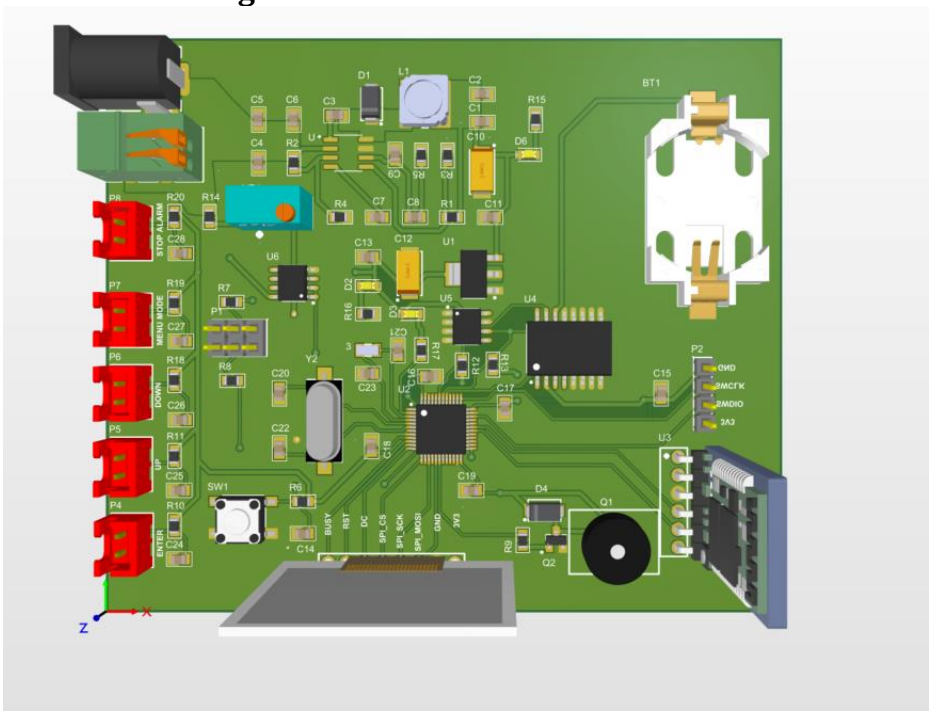


**Figure 7.** The view 3D of the circuit.

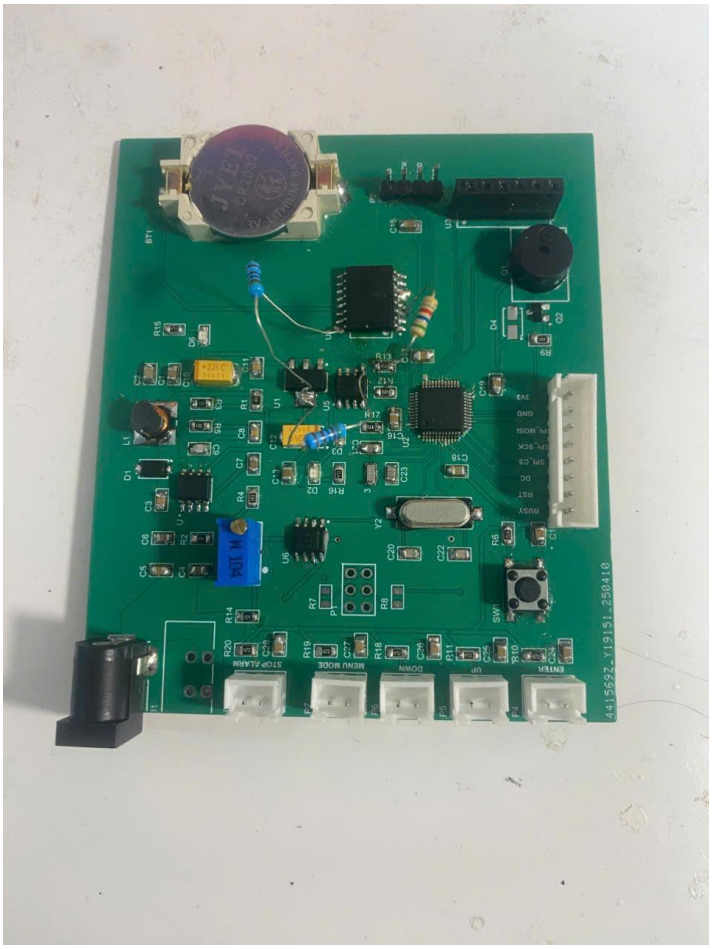


**Figure 8.** The view of the circuit.

## 3. Firmware Design

a) Requirement software

- Ability to process real-time signals ensuring the accuracy of real time.
- Users can adjust the time via Bluetooth: allowing users to easily adjust the time remotely.
- Can store time data when replacing the battery: limiting the need for users to adjust the time each time the battery is replaced.pin.

b) Flow chart:

- The flow chart will be divided into 3 parts:
    - The first part is about the default mode. It will display the data read from the RTC and check the alarm.
    - The second part is the flow chart of the system options choosing mode.
    - The third part is the flow chart of controlling button.
- Default mode:

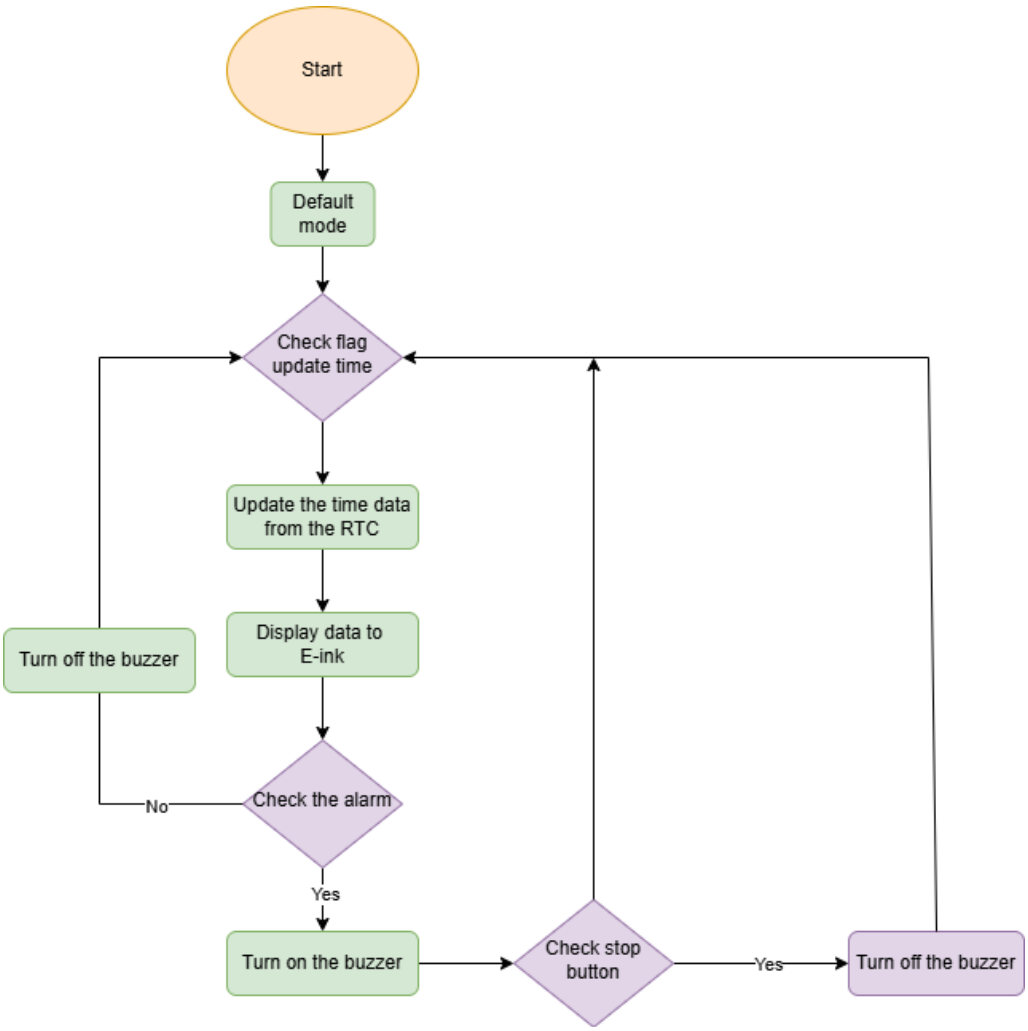



**Figure 9.** Flow chart of Default mode

- System options mode:

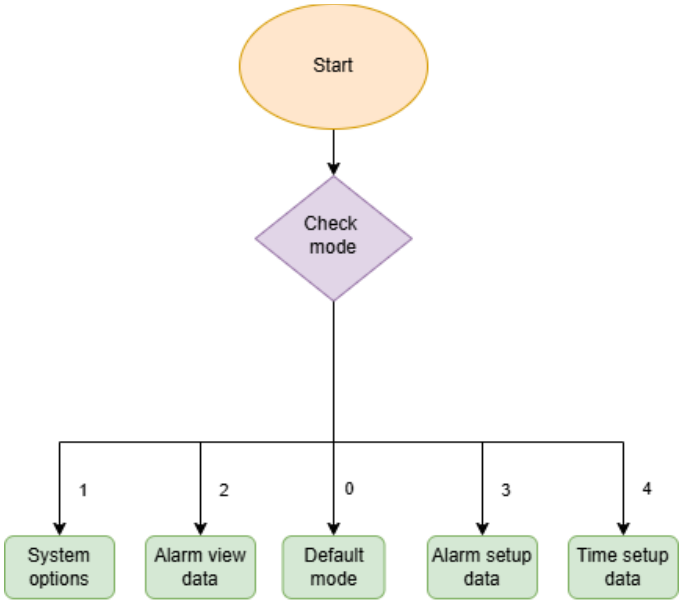



**Figure 10.** Flow chart of System options mode

- Button setting:

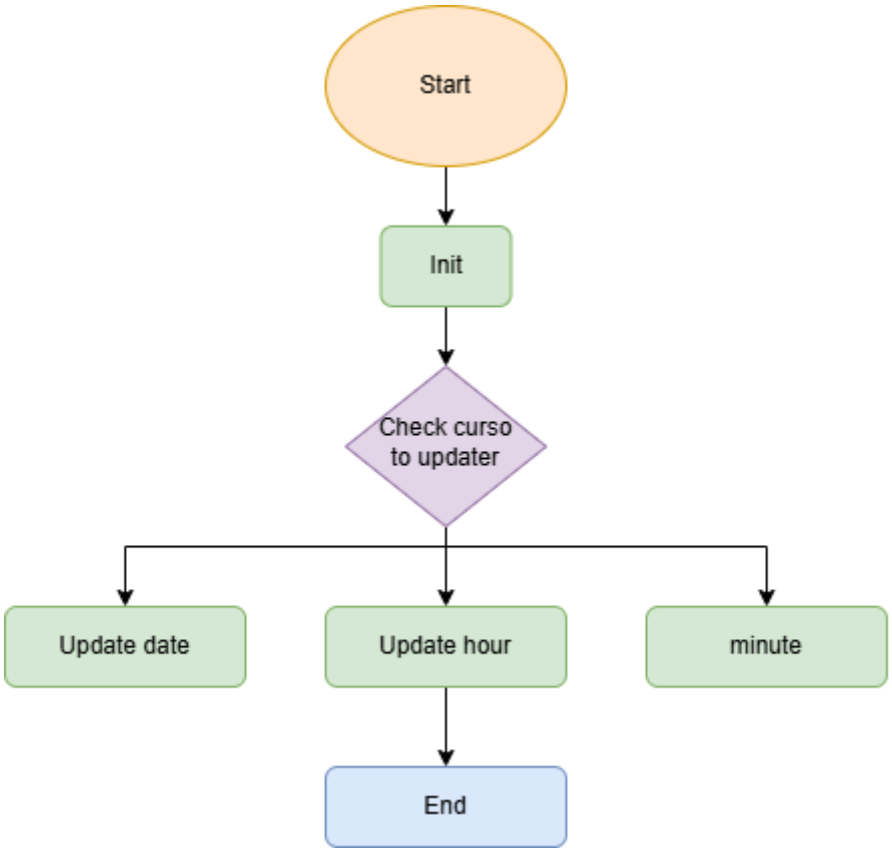



**Figure 11.** Flow chart of Button setting mode

## 4. Mobile Application Design

### *a) Introduction to MIT App Inventor*

- To enhance user interaction and increase system flexibility, the *ChronoSync* embedded digital clock integrates a custom-built Android application developed using **MIT App Inventor**. This mobile app provides a wireless interface for adjusting both the time and alarms of the clock via Bluetooth communication, eliminating the need for manual configuration through physical buttons.
- The integration of the MIT App serves several important purposes in the system. Firstly, it improves **user convenience**, allowing real-time configuration of the clock from a distance without the need to physically access the hardware. Secondly, it enables **multi-mode interaction**, with distinct features such as time adjustment and flexible alarm scheduling for both weekly and monthly routines. Lastly, it demonstrates the practicality of combining embedded hardware with mobile software to build a more modern and responsive embedded system.
- Using MIT App Inventor brings notable advantages:
  - **Rapid development** with a visual programming environment
  - **Cross-platform compatibility** for Android devices
  - **Ease of integration** with Bluetooth modules like HC-05
  - **User-friendly interface** design without advanced coding

- In the *ChronoSync* project, the MIT App plays a crucial role as the **primary interface** between the user and the embedded clock system. It allows seamless communication over Bluetooth, enabling users to manage essential clock settings efficiently and intuitively.

*b) User interface and modes*

MIT App Inventor let user define the interface of the app and It's functions through blocks.

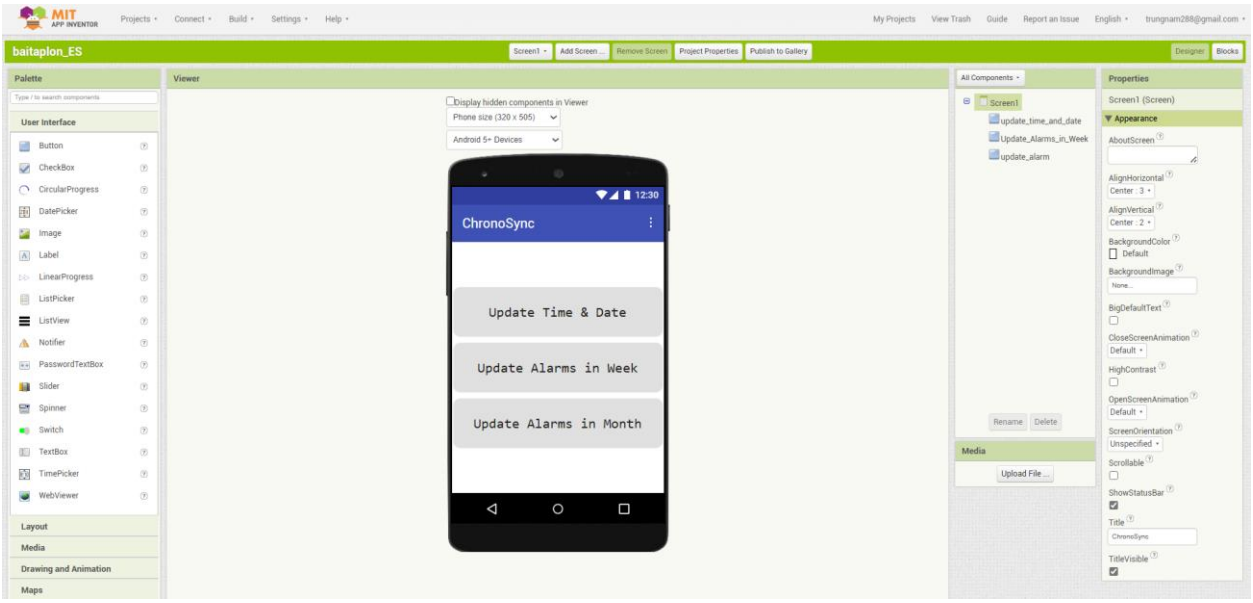



**Figure 11.** Designer mode of MIT App Inventor

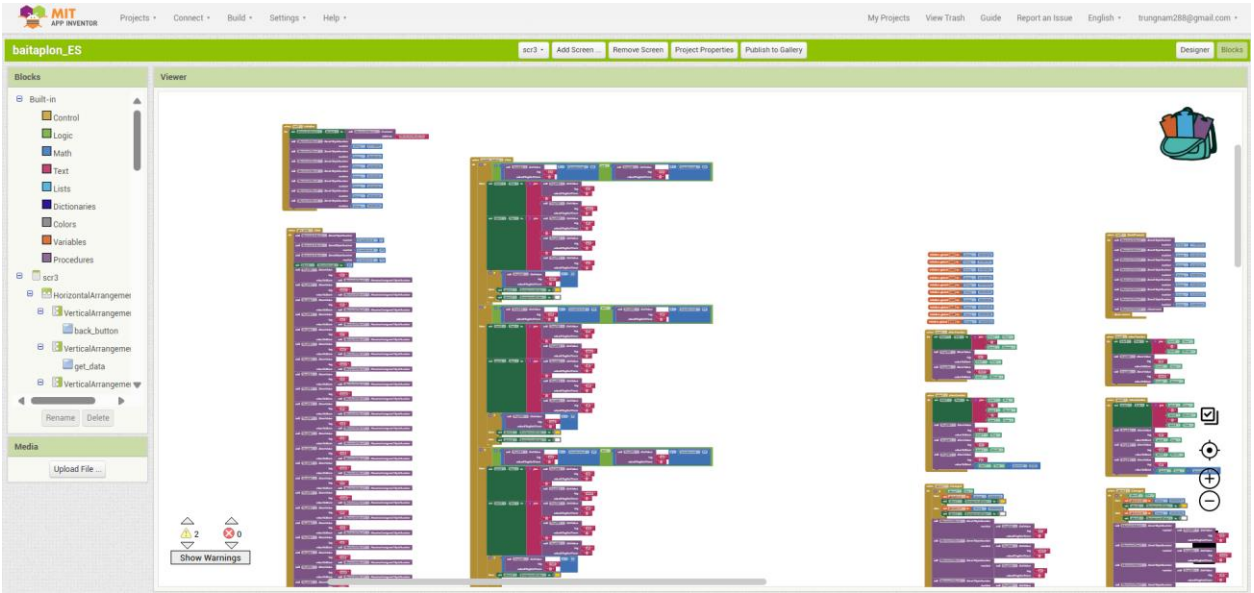



**Figure 12.** Blocks mode of MIT App Inventor

**Screen 1: Menu**

Screen 1 is the first screen and the default screen that will automatically appear when we open the app. In this screen, we are free to choose what to do with our clock through 3 buttons:

- "Update Time and Date"
- "Update Alarms in Week"

- "Update Alarms in Month"

In Designer mode, we will see 3 buttons appear on the screen like the figure below:

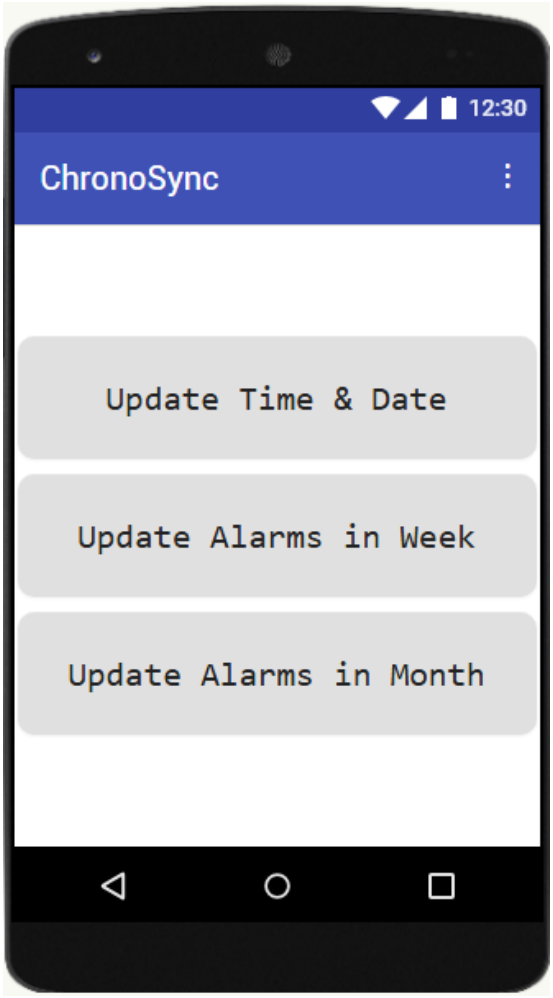



**Figure 13.** Screen 1

In the Blocks mode, We can observe the function of each buttons:

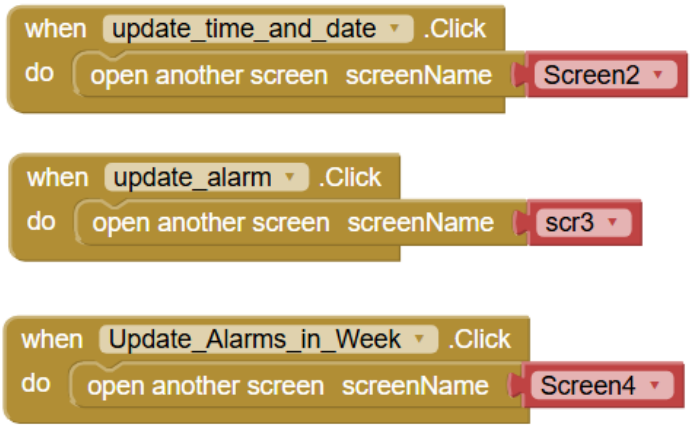



**Figure 14.** Blocks functions of Screen 1

For each buttons, they will lead us to others screen according to their functions.

In this screen, the app is staying in Mode 0 (Idle Mode): Do nothing.

### Screen 2: Update Time and Date

After pressing the first button (Update Time and Date), we get to screen 2 that allow us to change the time of the clock.
The system is now in Mode 1 (Time Adjustment Mode).

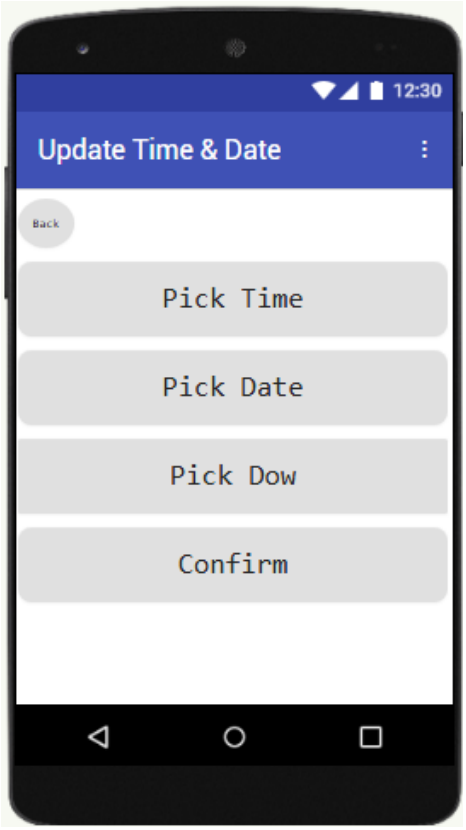



**Figure 15.** Screen 2

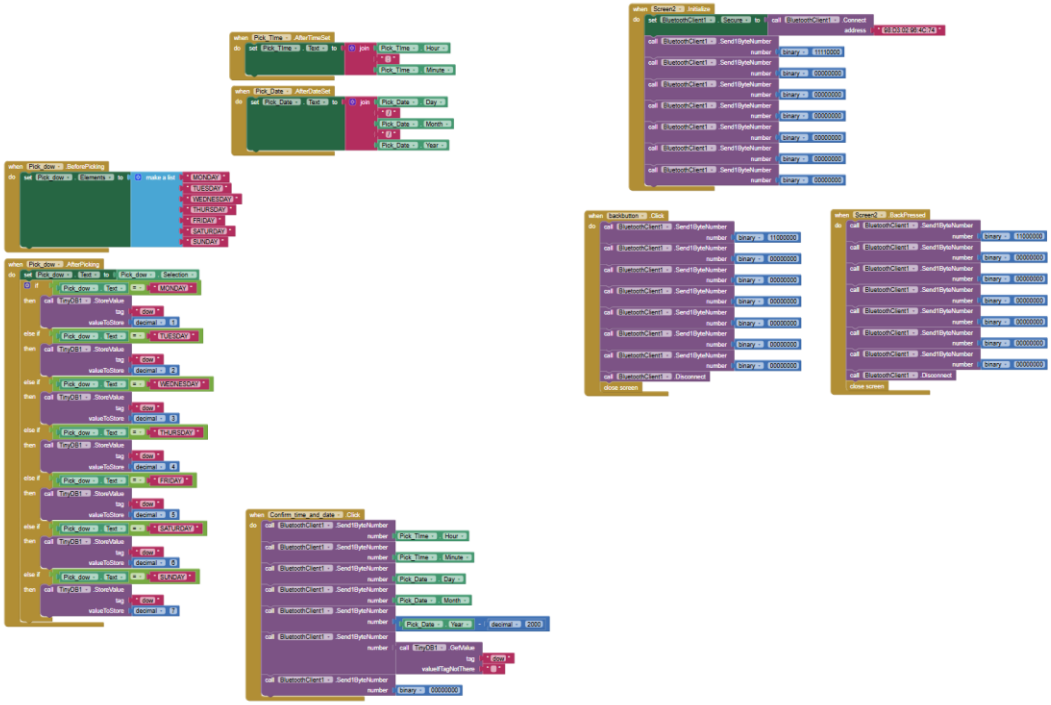


**Figure 16.** Block function of Screen 2

In this mode, we can change time, date, day of the week and then press the confirm button to admit the adjustment to the clock.

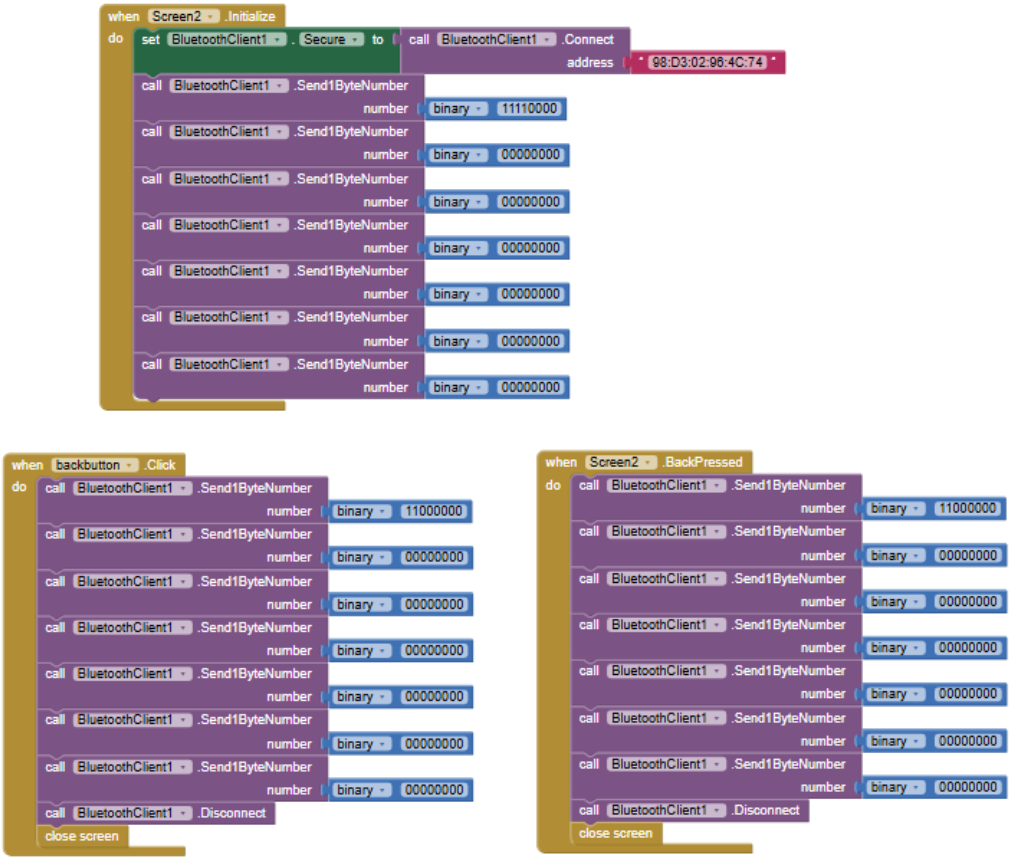



**Figure 17.** Initializing block and ending blocks of Screen 2

When the screen 2 initialize, the app will automatically connect to HC05 through address **98:D3:02:96:4C:74**. After that, it will send 7 byte data (**F0-00-00-00-00-00-00**) to STM32 to announce that we're in mode 1. The first byte is not only for hour data but for mode data too. Only when we press the back button or back press, STM32 will receive data (**C0-00-00-00-00-00-00**) that let it escape mode 1 to turn back to mode 0.

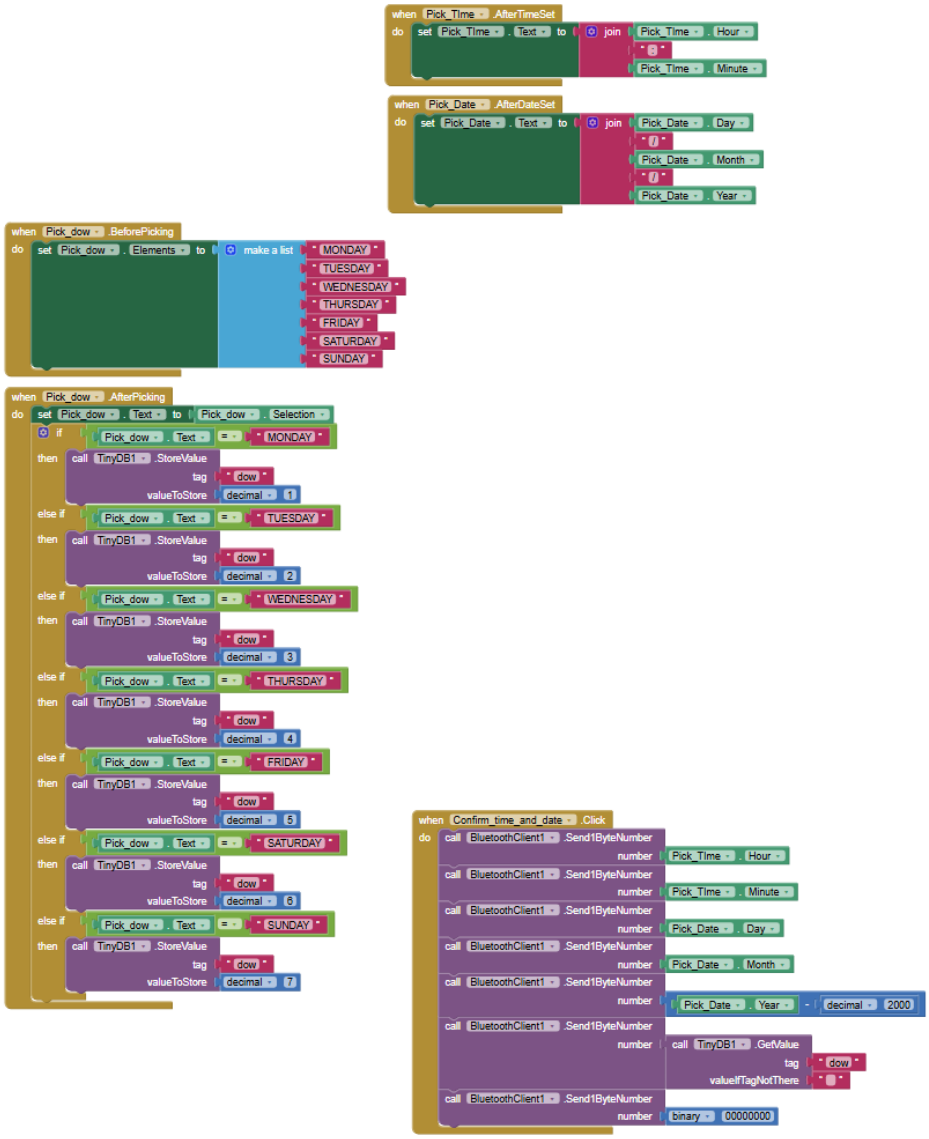



**Figure 18.** Function blocks of Screen 2

After set the time and date button, they will display the value that we chose.

For day of the week, we will choose from a list of day (Monday to Sunday). After picking, it will show our option on the screen and also save the value of **dow** value as it's equivalent number. (1-Monday, 2-Tuesday, etc).

When we press the confirm button, all the data of time and date will be sent to STM32 via Bluetooth.

Note that data of the year have to minus for 2000 as it's too big to fit in 1 byte.

**Screen 3: Update Alarms in Month**

After pressing the second button (Update Alarms in Month), we get to screen 3 that allow us to change the alarms of the clock in date mode.

The system is now in Mode 2 (Alarms Adjustment Mode).

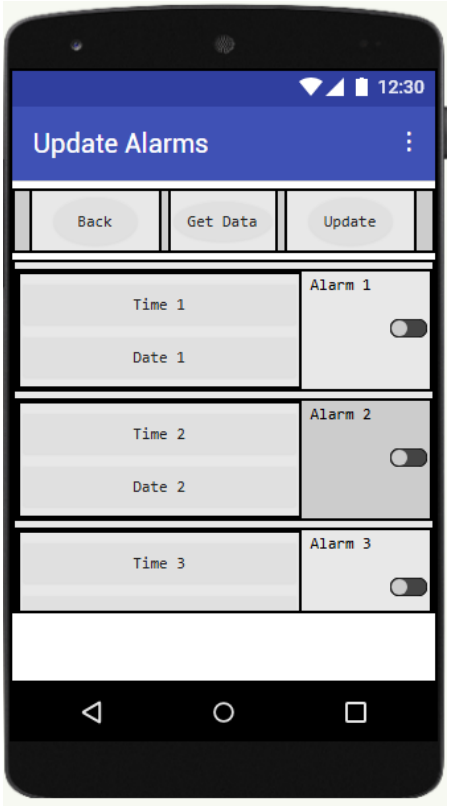



**Figure 19.** Screen 3

**Figure 20.** Blocks of Screen 3

In this mode we can adjust and set the alarms of the clock in detail of time and date.

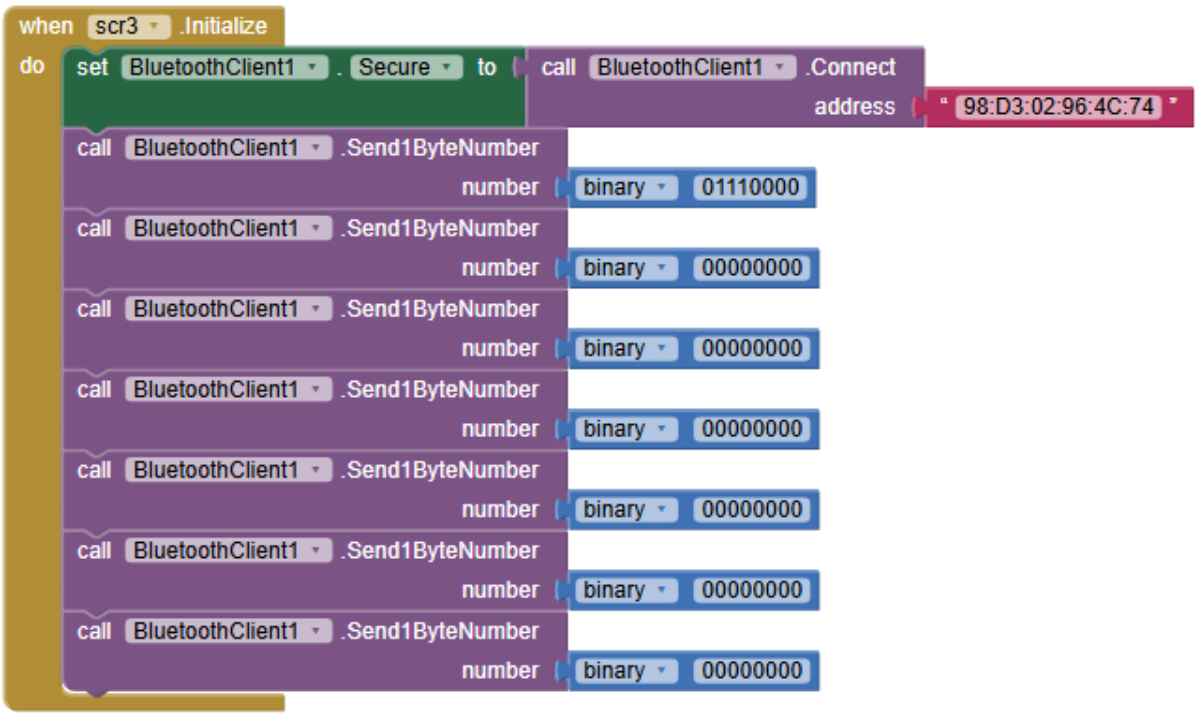
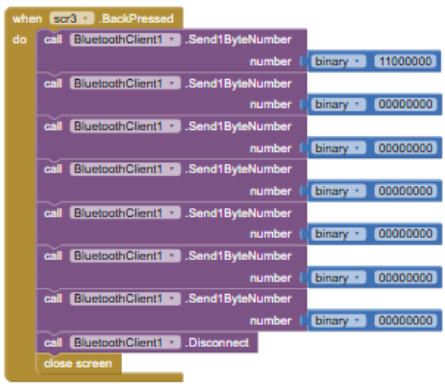



**Figure 21.** Initializing blocks of Screen 3

When screen 3 initialize, the app will automatically connect to HC05 and then send 7-byte data to STM32 to announce mode 2.


**Figure 22.** Enable variables for alarms in Screen 3

Besides, it will also initialize 10 global variables for alarm enable data. These objects are used to check if the alarms are turn on or off.



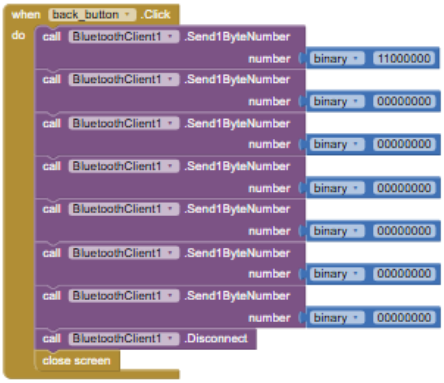



**Figure 23.** Outing blocks of Screen 3

Same as mode 1, when we press back button or back press, it will send data to let system turn back to mode 0 and then close the current screen.

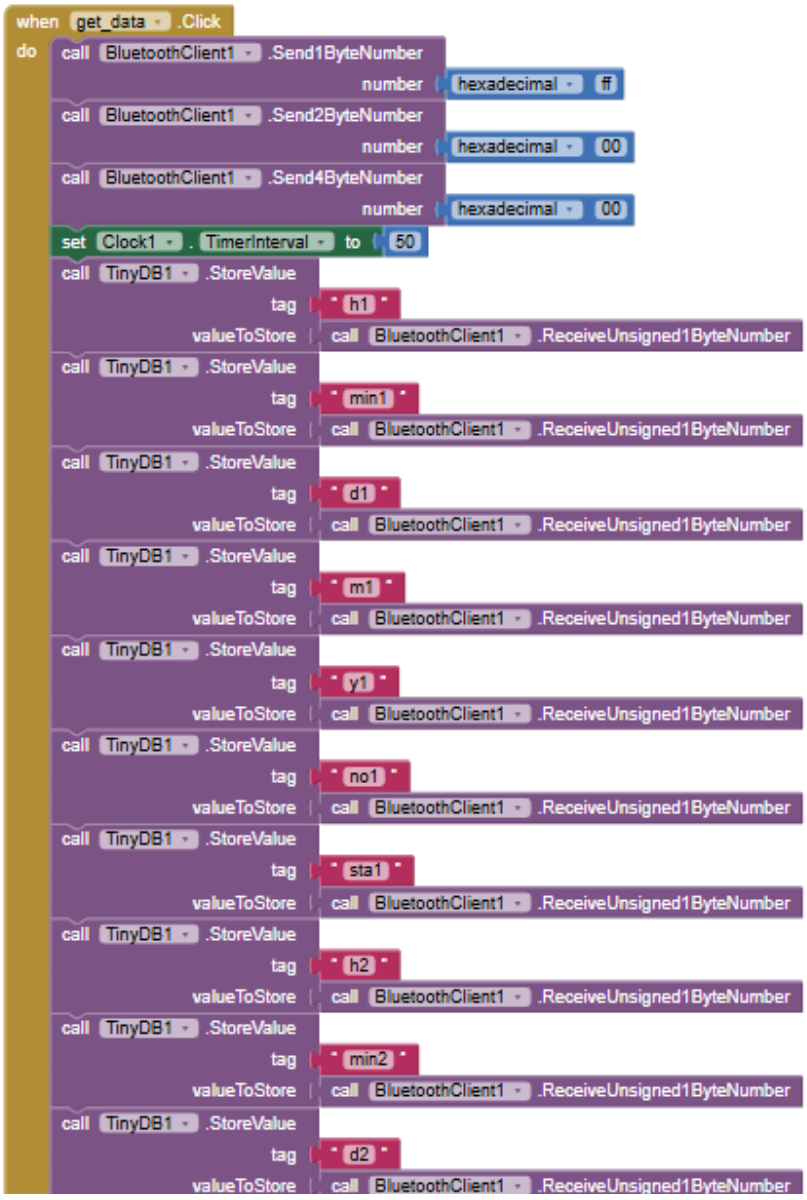



**Figure 24.** get_data blocks of Screen 3

This get data button is used to receive all 10 alarms data in STM32 and store them all in those tag. For each alarm, there will be 7 tags: **h (hour), min (minute), d (day), m (month), y (year), no (# of the alarm), sta (status of the alarm)**.

Before we do all that, the app will send data to request STM32 to send data and then wait 50ms to stabilize the system.


**Figure 25.** update_button blocks of Screen 3

After we got all the alarms data stored, we need to update them to the app.

As there are 2 alarm adjustment modes: one for month and one for week, we will consider 2 byte "month" and "year" of the alarms. If both of them are **FF** we know that this is for week mode and vice versa.

With those data store in tags, we update the interface of the app.

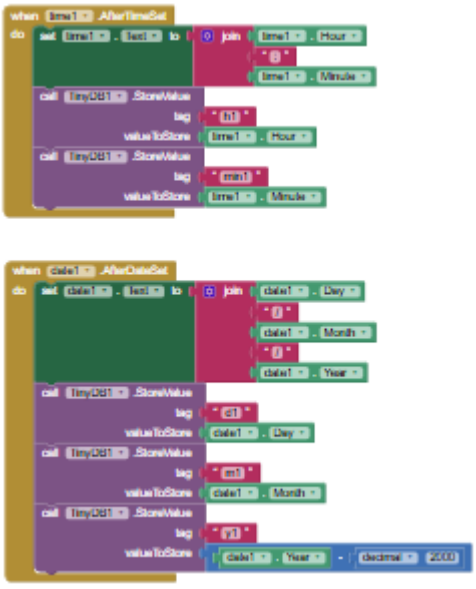



**Figure 26.** Setting time and date blocks in Screen 3

These two blocks will help us change and update the interface of the app after we set value.

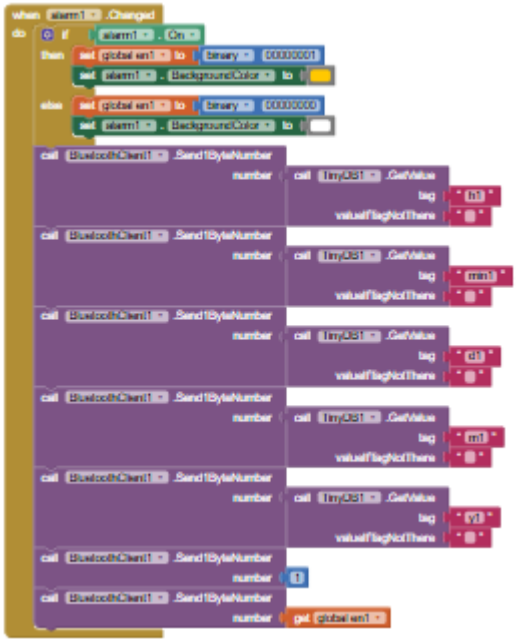


**Figure 27.** Sending data in Screen 3

To send data to the STM32, we toggle the switch called **alarm** to send all the data.

**Screen 4: Update Alarms in Week**

After pressing the third button (Update Alarms in Week), we get to screen 4 that allow us to change the alarms of the clock in week mode.

The system is now in Mode 2 (Alarms Adjustment Mode).


**Figure 28.** Screen 4

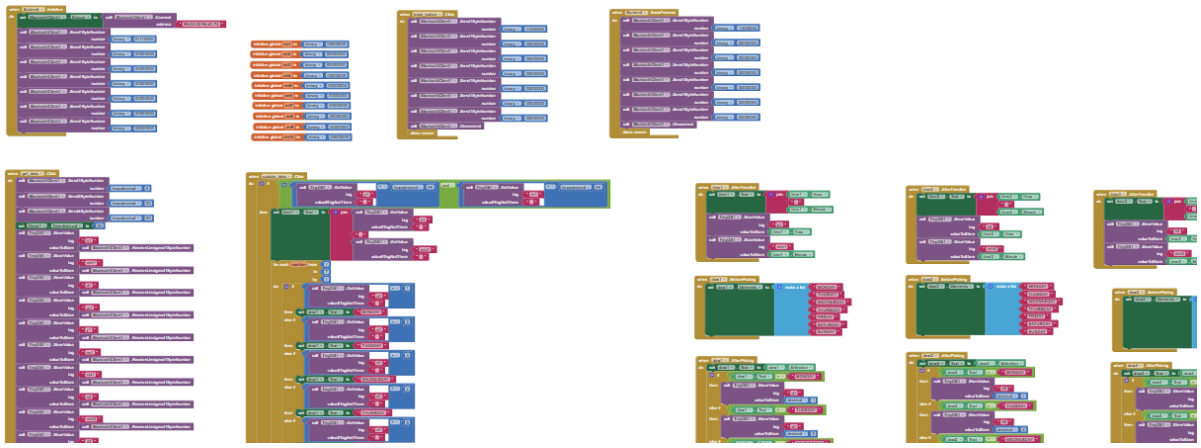


**Figure 29.** Blocks of Screen 4

In this mode we can adjust and set the alarms of the clock in detail of time and day in the week.

**Figure 30.** Initializing and outing blocks of Screen 3

Initialization and escaping is the same as screen 3.

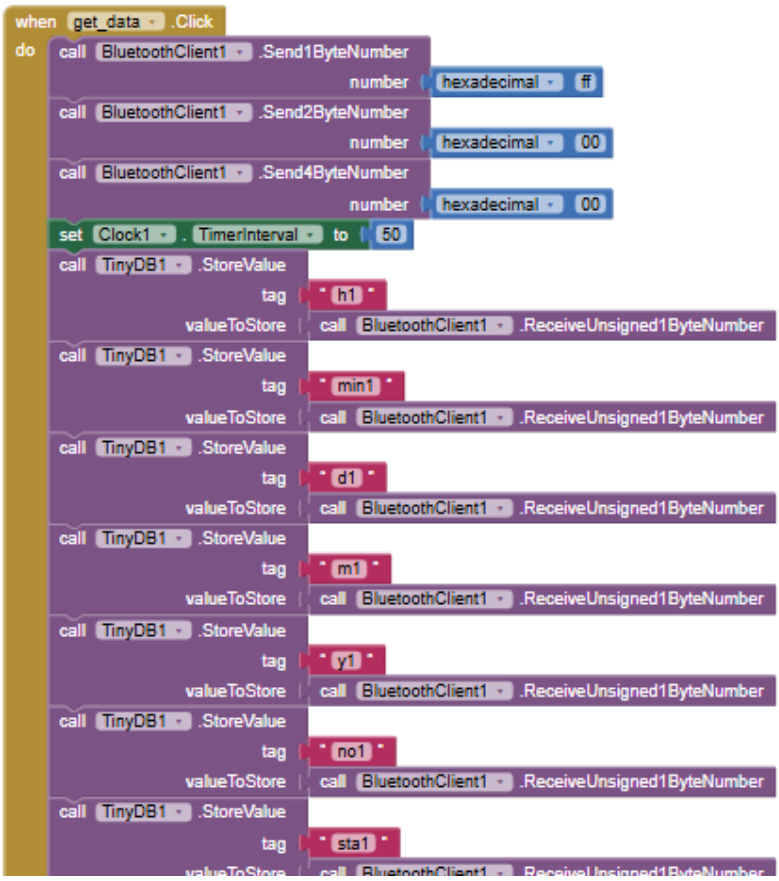



**Figure 31.** get_data blocks of Screen 4

Get data function is also same as the previous screen.

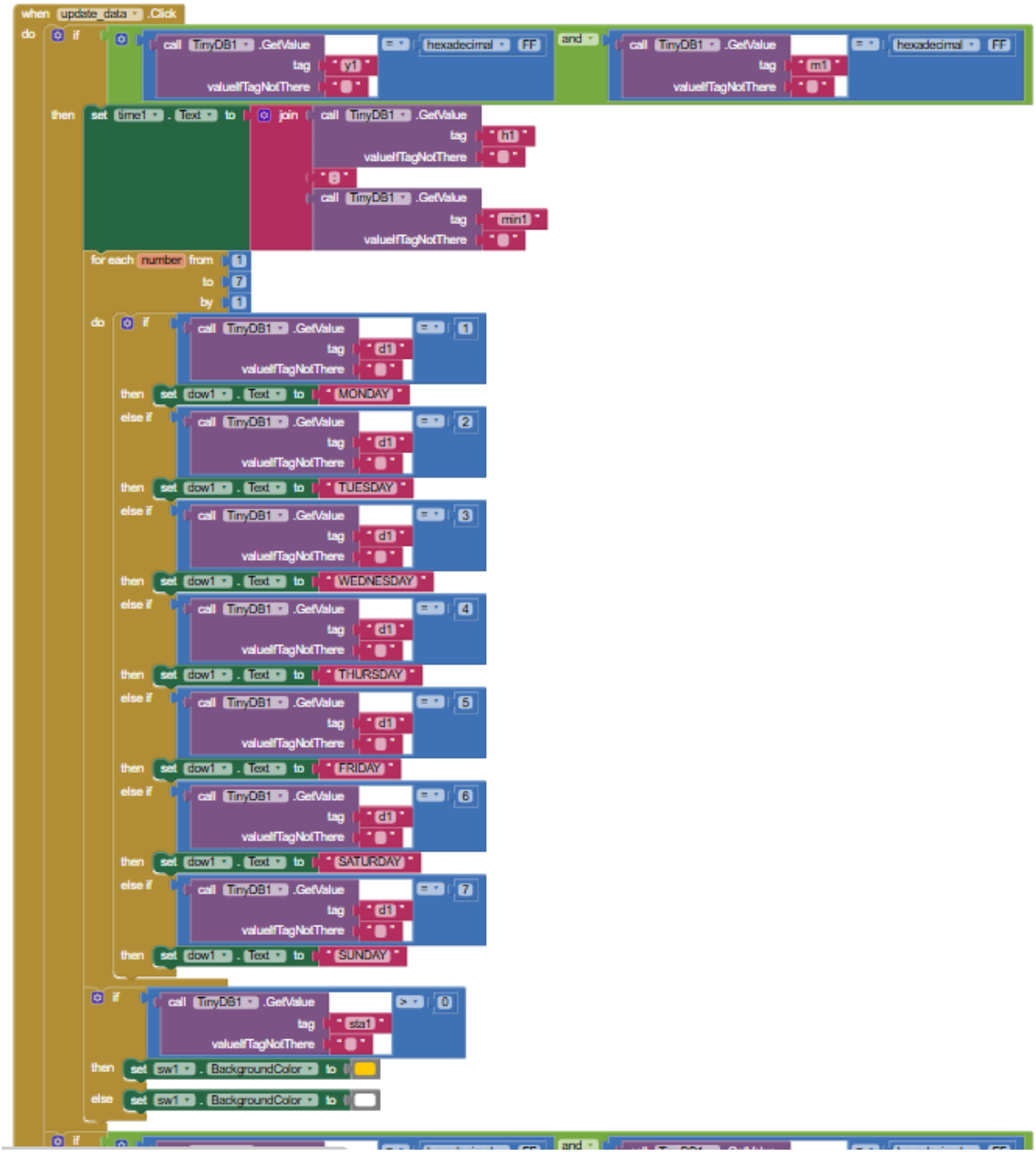



**Figure 31.** update_data blocks of Screen 4

For updating data, we will check for the **month** and **year** data if they're equal to FF or not. If yes, we will update the interface of the app in the format of time and day in week (Monday, Tuesday, Wednesday, etc). The **day** data will be used to display the day in week data under the form of number from 1 to 7 (Monday to Sunday). With each day, **d** will get it's suitable value.

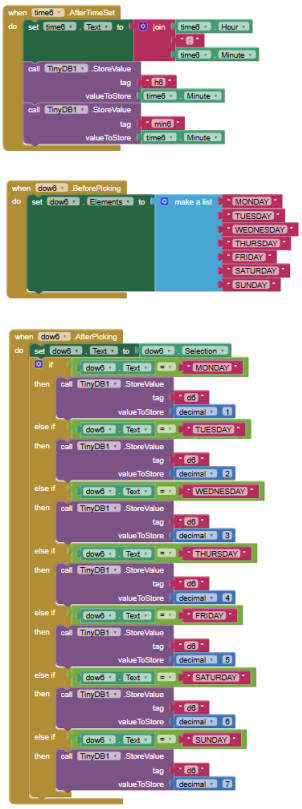



**Figure 31.** Alarms setting blocks of Screen 4

For setting time, it will be the same as previous one.

For setting day in the week, we will choose form a list of day include Monday to Sunday. After picking, the assigned number for each day will be stored into the **d** tag.

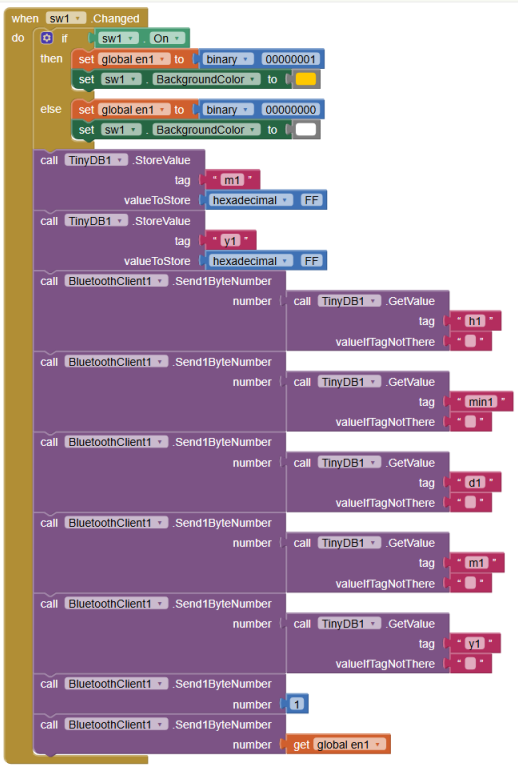



**Figure 31.** Sending data blocks of Screen 4

Sending data to STM32 is also the same when we want to adjust alarms in months.

## D. RESULTS AND DISCUSSION

### 1. Implemented Features

The system has successfully implemented a majority of the specified functional requirements as follows:

- **FR-1: Time Display Functionality**
  - The system retrieves and displays real-time data from the DS3231 RTC module.
  - Time is updated every second using an external interrupt generated by a 1 Hz square wave from the RTC.
  - The display format includes hour, minute, day of the week, date, month, and year.
- **FR-2: Alarm Management**
  - Users can configure up to 10 alarm entries. Each alarm includes fields for hour, minute, selection of day-of-week or date-of-month mode, the specific day or date value, and an ON/OFF flag.
  - A hierarchical matching algorithm checks alarms against the real-time clock every second, starting from the ON/OFF status, followed by day/date, hour, minute, and finally second.
  - If a match is found, the system activates a buzzer with a pre-defined melody pattern and displays alarm status.
- **FR-3: Time and Alarm Setting via Physical Buttons**
  - Five physical buttons enable users to navigate through setup interfaces and adjust time or alarm parameters.
  - The buttons support incrementing/decrementing values and navigating between fields.
- **FR-4: Alarm Review and Editing**
  - A dedicated alarm review mode allows users to browse through saved alarm slots.
  - While reviewing, users can toggle the ON/OFF status of an alarm or enter editing mode to adjust its configuration.
- **FR-5: Time and Alarm Configuration via Mobile Application**
  - The system communicates with a mobile app through Bluetooth using an HC-05 module.
  - Users can wirelessly send updated time and alarm configuration data, which is then decoded and applied to the system's internal state.
- **FR-6: System Utility Features**
  - A system options mode is available in the user interface, allowing users to perform selected utility functions.
  - Currently, the feature to clear all alarms is implemented and functional.
- **FR-7: Physical Device Requirements**

    - The system is built on a compact PCB with all components placed for minimal size and optimal routing.
    - Although the final enclosure is handmade from carton for prototyping, the layout complies with dimensions suitable for future ABS/Mica casing.
- **FR-8: Power Efficiency and Display Technology**
    - The device uses a 2.9-inch E-Ink display to ensure ultra-low power consumption during idle states, as it only consumes energy during screen refresh.
    - The power system is designed around a TPS54231 buck converter and AMS1117 linear regulator to support efficient voltage regulation.

## 2. Testing and Debugging Process

During system integration, several functional conflicts and implementation challenges were encountered and resolved collaboratively:

- **Button Debounce and Press Type Detection**: Short and long button presses were distinguished using debounced input readings and timing thresholds. Without debouncing, erratic behavior was observed during mode transitions.
- **System Mode Handling vs. E-Ink Display Timing**: The E-Ink display requires a physical delay to refresh. Mode-switching logic and button handlers were designed to avoid interrupting display operations by introducing non-blocking synchronization.
- **RTC and EEPROM $I^2C$ Communication**: Conflicts arose when simultaneous $I^2C$ operations were attempted. The $I^2C$ routines were made atomic and managed using state flags to prevent bus contention.
- **RTC Square Wave Output to STM32 Interrupt**: The DS3231 was configured to generate a 1 Hz square wave signal. This was connected to an external interrupt on the STM32 to ensure precise timekeeping without relying on software timers.
- **Time and Alarm Data Encoding/ Decoding**: Time and alarm structures were encoded compactly for storage in EEPROM. Bit-level packing was used to reduce memory footprint while preserving data integrity.
- **Efficient Alarm Matching Algorithm**: Alarm match logic was optimized to reduce computation time by applying early exits: if the alarm is disabled, or if the day/date does not match, the handler skips remaining checks to save power and processing time.
- **Alarm Ring Melody Management**: When an alarm triggers, a melody is played using the buzzer. Timed sequences were designed to avoid interference with main logic and user interactions.
- **System State Management Across Modes**: Transitions between system modes were carefully managed to preserve relevant state variables and reset temporary fields only when necessary (e.g., upon exit).

- **Context-Aware Button Handling**: Button behavior was made context-sensitive based on the current and previous modes. This ensures intuitive navigation and prevents incorrect action triggering across modes.

## 3. Challenges and Solutions

While the project has successfully achieved the majority of its functional requirements, several features and performance constraints remain partially completed or untested due to time limitations, hardware constraints, and integration challenges. The tables below outlines these issues, the underlying causes, and proposed solutions for future improvement.

### *3.1. Functional Limitations*

| Challenge | Description | Proposed Solution |
|---|---|---|
| **Button long press: continuous increment/decrement** | Although long press detection is implemented, continuous value increment/decrement during time and alarm setup has not been realized. This limitation is due to the blocking nature of E-Ink display refresh cycles, which interrupt the periodic update required for repeated input changes. | Refactor the button handler to operate asynchronously from the display update process. Use a timer-based interrupt or non-blocking polling mechanism to trigger value changes independently of the display state. Delay or queue display refreshes to avoid conflicting with active input. |
| **Debounce delay conflicts with E-Ink display refresh** | The debounce logic includes a short delay that conflicts with the slow physical refresh time of the E-Ink display, causing visual artifacts or latency during frequent user input. | Separate the display update logic into a non-blocking task or conditional refresh trigger to minimize impact on button handling routines. |
| **Mobile app functionality incomplete** | The mobile app does not yet support real-time clock synchronization display, alarm list editing, or battery percentage reporting. | Extend the BLE communication protocol to support two-way synchronization for time and alarm slots. Add new UUIDs for battery status and system flags. Use periodic polling or notification callbacks from the embedded system. |
| **Display interface not fully designed** | The E-Ink display currently shows raw system values without a user-friendly graphical layout or labels. | Design a cleaner UI using bitmap icons, improved font layout, and logical screen partitioning. Consider integrating a lightweight UI library compatible with E-Ink displays. |
| **System option functions unimplemented** | The functions for factory reset and display test patterns in System Options Mode are defined in UI but not yet programmed. | Add handler functions for each feature. Factory reset can clear EEPROM and reset system state. Display test can toggle through predefined patterns or contrast checks. |

| | | |
|---|---|---|
| **No finalized enclosure (Mica/ABS)** | The prototype is housed in a temporary carton-based casing, lacking robustness or long-term usability. | Design a 3D-printable case using Mica or ABS plastic. Ensure proper cutouts for display, buttons, and ventilation. Future versions can consider injection molding for product-scale development. |
| **Battery holder not integrated into PCB** | Although footprint and schematic are prepared for battery power, no actual battery is installed. Power is currently supplied through DC adapter only. | Solder the lithium-ion battery holder, verify the power path, and enable real-time voltage measurement through ADC. Re-enable battery percentage display logic using pre-defined conversion formula from ADC input. |

### *3.2. Constraints and Testing Limitations*

| Constraint | Description | Proposed Solution |
|---|---|---|
| **C-1.2: Alarm matching time < 50 ms** | The alarm matching logic is designed for optimized performance, but its actual execution time has not been benchmarked. | Use HAL_GetTick() or STM32 cycle counters to measure time taken per loop. Optimize further by breaking out early from comparison steps if mismatches are found. |
| **C-2.3: EEPROM access time < 10 ms** | Read/write operations to EEPROM are fast in practice, but precise measurement is yet to be conducted. | Benchmark write and read durations using debug pins or live expression timing, and optimize by minimizing repeated EEPROM access during frequent tasks. |
| **C-4.3: Power consumption < 3 mW** | The system is designed for low power with an E-Ink display and efficient DC/DC converters, but actual power draw has not been measured. | Use a precision power analyzer to measure current draw in typical idle and active states. Confirm average consumption over time and identify optimization opportunities in idle states (e.g., reduce ADC/UART activity). |

## 4. Performance Evaluation

The system has been assessed against the majority of the defined performance constraints. The following summarizes the constraints that have been successfully met, along with their key performance indicators:

- **C-1.1: Timekeeping Accuracy – ±3 ppm at room temperature**: The system employs the DS3231 RTC module, which features an integrated temperature-compensated crystal oscillator (TCXO) with a typical accuracy of ±2 ppm at room temperature. During testing, the RTC maintained timekeeping precision with no observable drift over 24-hour periods, satisfying this constraint.
- **C-1.3: Time Setting Resolution – Configurable to seconds**: The time and alarm setup interfaces enable users to configure parameters including second, minute,

hour, and either day-of-week or date-of-month. The inclusion of second-level adjustment meets the requirement for fine time resolution.

- **C-2.1: EEPROM Data Persistence – Retain data after power loss**: The $I^2C$ EEPROM reliably stores user-configured alarm data. After system resets and power cycles, all alarm data is correctly retained and retrieved without corruption, meeting the persistence requirement.
- **C-2.2: EEPROM Endurance Handling – Minimize unnecessary writes**: To extend EEPROM lifespan (typically ≥ 1 million write/erase cycles), a conditional write strategy is employed. EEPROM is only updated when new data differs from the stored content, reducing redundant write operations and preserving endurance.
- **C-3.1: User Interface Responsiveness – Response latency < 100 ms**: All short press events are debounced and processed within approximately 30–50 ms. System mode transitions and screen updates respond promptly to user inputs, ensuring a smooth and responsive interface.
- **C-4.1: Low Power Sleep Mode Support – Idle periods enabled, low activity**: Although full sleep mode is not yet implemented, the system architecture supports efficient task scheduling. The processor remains idle between events, and peripheral access (e.g., $I^2C$, display updates) is event-driven. This foundation supports future enhancements for power-saving.
- **C-5.1: Safety and Electrical Tolerance – Stable at 3.3V ±5%**: All components operate stably under regulated 3.3V supply. No overheating, brown-out, or power fluctuation was observed under normal operation, confirming electrical stability.
- **C-6.1: Firmware Modularity and Maintainability – Layered architecture with naming convention**: The firmware follows a structured design with HAL, BSP, Middleware, and Application layers. All modules adopt a consistent and descriptive naming convention (as documented), enhancing readability, debugging, and maintainability.

In summary, the system fulfills its core performance metrics related to accuracy, memory reliability, input responsiveness, and modular software design. Remaining constraints that have not been verified are discussed in the next section.

## E. CONCLUSION

The ChronoSyn embedded digital clock system has been successfully developed to meet the core functional requirements of timekeeping, alarm scheduling, user configuration via physical buttons and Bluetooth application, and low-power operation using an E-Ink display. Through a structured development process, the team integrated hardware and firmware components efficiently, ensuring modularity, maintainability, and expandability.

Despite the project's achievements, several features remain partially implemented or unverified, including enhanced mobile app capabilities, certain system utility functions, battery integration, and performance validation against specific timing and power constraints. These limitations primarily arose due to hardware prototyping constraints and time limitations in the academic schedule.

Nonetheless, the ChronoSyn system has demonstrated solid design principles, reliable functionality, and responsive interaction under real-world operating conditions. Future improvements will focus on completing the remaining features, optimizing power efficiency, and enhancing user interface design, both in hardware form factor and graphical layout.

This project provided valuable hands-on experience in embedded systems design, peripheral interfacing, real-time programming, and system-level debugging – bridging theoretical concepts with practical implementation in a collaborative engineering context.

## F. REFERENCES


1. STMicroelectronics, *CD00161566 - STM32F103C8T6 Datasheet – ARM Cortex-M3 32-bit MCU*, Rev. 18, Jul. 2023. [Online]. Available: https://www.st.com/resource/en/datasheet/cd00161566.pdf
2. Microchip Technology Inc., *AT24C64D - 64-Kbit Serial EEPROM I2C Compatible (Two-Wire Interface)*, DS20005937A, Feb. 2016. [Online]. Available: https://ww1.microchip.com/downloads/aemDocuments/documents/MPD/ProductDocuments/DataSheets/AT24C64D-64-Kbit-Serial-EEPROM-I2C-Compatible-Two-Wire-DS20005937.pdf
3. Analog Devices, *DS3231 - Extremely Accurate I²C-Integrated RTC with TCXO and Crystal*, Rev. 11, Jul. 2020. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ds3231.pdf
4. Shenzhen Ehong Technology Co., Ltd., *HC-05 Bluetooth Module Datasheet*. [Online]. Available: https://components101.com/sites/default/files/component_datasheet/HC-05%20Datasheet.pdf
5. Waveshare Electronics, *2.9inch E-Ink Display Module Datasheet*, [Online]. Available: https://www.waveshare.com/w/upload/e/e6/2.9inch_e-Paper_Datasheet.pdf