# Project Documentation: IMPLEMENTING SMART WATCH FUNCTIONALITIES

## Introduction

This smartwatch project is a multi-functional device designed to support daily tasks with a range of time-management, connectivity, and organizational features. The project combines an alarm system, Bluetooth connectivity, calendar, real-time clock, and stopwatch into a single compact design, making it a practical assistant for productivity and personal scheduling.

## Summary of Functionalities:

**1. Alarm System:**

- Implements a timer-based alarm with configurable study and break intervals, following a Pomodoro-inspired approach.
- The alarm alternates between study sessions and breaks, with an optional long break after a set number of cycles, enhancing productivity and ensuring better time management for the user.

**2. Bluetooth Connectivity:**

- Enables communication with external devices, allowing for data transfer and notifications to and from connected devices.
- Facilitates integration with smartphone applications, extending the functionality of the smartwatch beyond its own display, enabling remote monitoring and control.

**3. Calendar:**

- Provides date-tracking functionality, enabling users to view the current date and plan ahead.
- Simplifies scheduling of reminders and events, working in sync with the real-time clock to provide time-sensitive notifications and keep users organized.

**4. Real-Time Clock (RTC):**

- Maintains accurate timekeeping even in low-power modes, ensuring reliability across all time-based functions like the alarm and stopwatch.
- Supports real-time clock display, alarm timing, and date tracking without the need for constant user interaction, providing a seamless user experience.

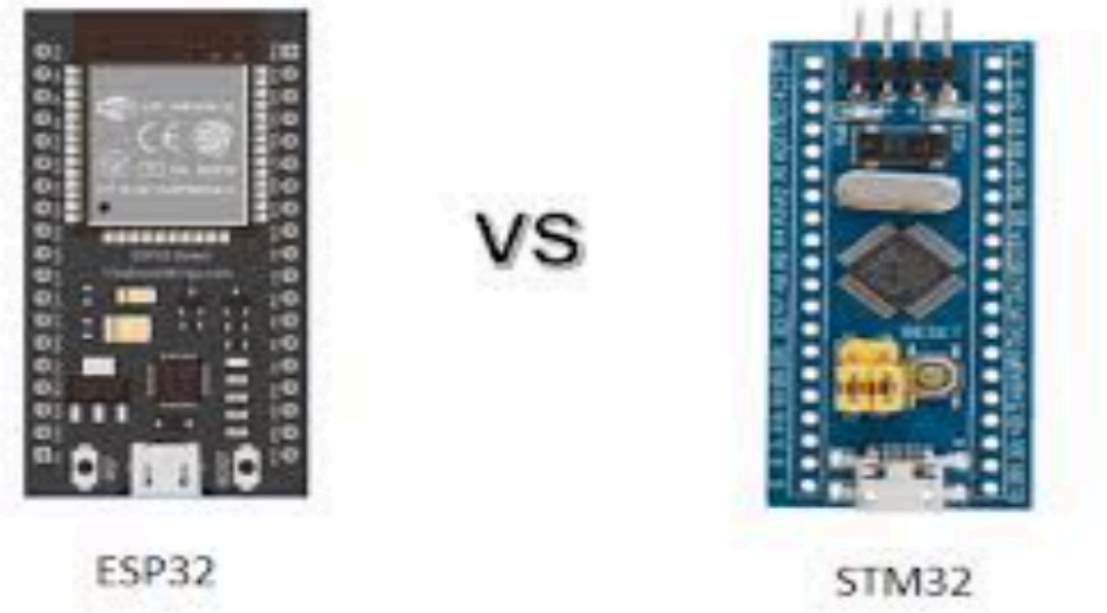
**5. Stopwatch:**

- Offers precise interval measurement, perfect for timing activities, exercises, or events where tracking elapsed time is critical.

- Supports both start/stop and reset functions, making it versatile for various timing needs such as workout sessions, tasks, or other time-based activities.

**6. Temperature Functionality:**

- **Real-Time Temperature Monitoring**: Displays ambient temperature data, continuously tracking and showing real-time environmental information on the smartwatch screen.
- **Heat Index Calculation**: Computes and shows the heat index by considering both temperature and humidity, providing users with a more accurate representation of the current weather conditions.




| FEATURE | ESP32 | STM32 |
|---|---|---|
| **Bluetooth Connectivity** | Requires an external Bluetooth module, adding complexity and cost. | Requires an external Bluetooth module, adding complexity and cost. |
| **Real-Time Clock (RTC)** | Basic RTC functionality, but lacks battery-backed RTC for long-term accuracy | Basic RTC functionality, but lacks battery-backed RTC for long-term accuracy |

| | | |
|---|---|---|
| **Display Compatibility (U8g2)** | Compatible with U8g2 library, simple setup for OLED displays. | Compatible but setup can be more complex, requiring low-level configurations. |
| **Temperature Sensor Integration** | Supports both analog and digital sensors, but no ambient temperature sensor on board | Supports both analog and digital sensors; some models include an internal temperature sensor. |
| **Power Efficiency and Battery Life** | Power-saving modes, but higher power consumption when Bluetooth/Wi-Fi is active | Excellent low-power options, especially in STM32L series, ideal for battery-powered devices |
| **Timer-Based Alarm System** | Multiple hardware timers, can handle alarms with careful power management. | Low-power timers with RTC-based alarms, suitable for alarms in ultra-low-power modes |
| **Development Environment** | Broad support in Arduino IDE, beginner-friendly, extensive libraries for connectivity | Requires STM32CubeIDE or STM32 HAL libraries; more complex setup but powerful capabilities. |
| **Cost and Availability** | Affordable with built-in Bluetooth and Wi-Fi, widely available. | Generally more expensive, especially when including an external Bluetooth module. |

# 1. Timer and Alarm System for Study and Break Sessions in Smartwatch

**System Design**

The project is a timer system for study and break sessions displayed on an OLED screen. It includes:

- **Study and Break Sessions**: A timer alternating between study, short break, and long break intervals based on specified durations.
- **OLED Display**: Shows session type (study or break) and the countdown timer.
- **Button Control**: A button initiates, pauses, and resets the timer.
- **Debounce Control**: Prevents false triggers from the button.

The system uses the `U8g2` library for controlling the OLED display and `millis()` for timing.

**Features used**

## a. Communication Interfaces (I2C)

- **Implementation:** For this project, the I2C protocol is utilized for communication with the OLED display. I2C allows efficient two-wire communication between the microcontroller and the display module, facilitating low-latency updates of time and status.
- **Description:** The display constantly shows the current time, alarm messages, and timer status updates. I2C enables smooth, synchronous data transfer, ensuring the user interface remains responsive.

## b. ADC/Digital Interface

- **Implementation:** The Digital Interface is used to read the input from a push button for starting, resetting, and stopping the timer.
- **Description:** The digital read function (`digitalRead()`) continuously monitors the button state. When the button is pressed, it triggers state changes in the timer (start/reset) and activates/deactivates the alarm. Using a digital interface here reduces complexity while ensuring precise control over timer state changes.

## c. Nested Vector Interrupt Controller (NVIC)

- **Implementation:** The NVIC is set up to handle the button press event with an interrupt. Debouncing is applied to eliminate accidental triggers due to mechanical bouncing.
- **Description:** NVIC allows the system to respond quickly to button presses without relying on polling. The interrupt-based approach reduces the load on the processor by only responding when the button is pressed, enhancing energy efficiency and user experience.

## d. Display Interface

- **Implementation:** The project includes an OLED display that shows timer values, mode messages (such as "Study time" or "Break time"), and alarm notifications. The `U8g2lib` library is used to control the display.
- **Description:** The display provides real-time feedback to users by updating the timer countdown, current mode, and alarm status. Visual feedback from the OLED enhances user interaction, helping users stay informed about the current session status without additional devices.

This combination of communication, control, and feedback systems ensures efficient operation for an alarm functionality in a smartwatch. The use of I2C for the display and NVIC for interrupts adds robustness to the design, while digital input handling and visual feedback provide seamless, responsive user interaction.

**Code Structure**

The code is structured as follows:

- **Constants and Variables**: Defines the study, break, and timer parameters.
- **Setup Function**: Initializes the serial, display, and button pin.
- **Loop Function**: Manages button presses, starts and resets the timer, and updates the timer display.
- **Timer Functions**:
    - `updateTimer()`: Tracks the time within each session and transitions between study and break phases.
    - `displayTime()`: Updates the OLED display with the current session and countdown.
    - `startTimer()` and `resetTimer()`: Control the start/reset logic of the timer.
    - `displayMessage()`: Provides feedback on the display.

**Challenges and Solutions**

1. **Debouncing the Button**:
    - *Challenge*: The button triggered multiple times with a single press due to noise.
    - *Solution*: Implemented a `debounceDelay` of 200 milliseconds and checked elapsed time between presses.
2. **Session Transitions**:
    - *Challenge*: Correctly managing transitions between study and break states and displaying accurate information.
    - *Solution*: Defined `TimerState` with `IDLE`, `STUDY`, and `BREAK` states, enabling seamless switching.
3. **Countdown Display**:
    - *Challenge*: Display the countdown in "MM
      " format on the OLED.
    - *Solution*: Used `snprintf()` to format the time display string for consistent formatting.

4. **Timing Accuracy**:
    - *Challenge*: Ensuring the timer operates on a 1-second interval without drift.
    - *Solution*: Used `millis()` to check the elapsed time and update every second, independent of other operations.

**Summary**

This code provides a timer for alternating study and break sessions using the `U8g2` library for OLED display. Key challenges included debouncing, accurate time display, and handling session transitions, which were addressed through state management and regular checks with `millis()`. The project is modular and can be expanded with further features, like user-defined timer durations.

## 2.Bluetooth Notification System for Smartwatch

**System Design**
The Bluetooth Notification System allows the smartwatch to receive and display notifications via Bluetooth, enhancing its connectivity capabilities. The main features include:

- **Bluetooth Communication**: Receives messages from paired devices over Bluetooth, enabling notification display on the OLED screen.
- **OLED Display**: Displays the received notifications with a clear and stylish header, improving user experience.
- **Message Truncation**: Manages long messages by truncating them to fit the display.

**Features Used**

1. **Communication Interfaces (Bluetooth)**
    - **Implementation**: The project uses Bluetooth for wireless communication between the smartwatch and other devices. The BluetoothSerial library establishes a connection and allows data exchange with paired devices.
    - **Description**: Bluetooth provides a convenient method for sending notifications to the smartwatch without physical connections, allowing users to receive messages in real-time directly on their wrist.
2. **Display Interface**
    - **Implementation**: An OLED display is integrated to show the received notifications. The `U8g2lib` library is used to control the display, enabling clear and well-formatted messages.
    - **Description**: The OLED screen displays incoming notifications with a header, a formatted message body, and a bottom border. The message is truncated if it exceeds 20 characters to ensure readability on the small screen.

**Code Structure**

- **Constants and Variables**: Initializes Bluetooth and OLED display settings, and a variable to store received messages.
- **Setup Function**: Sets up Bluetooth and OLED display, preparing the smartwatch for incoming notifications.
- **Loop Function**: Checks if new data is available from Bluetooth. If a new message is received, it is displayed on the OLED screen.
- **Display Function (`displayNotification`)**: Formats and shows the notification on the OLED, with a header and truncation for long messages.

**Challenges and Solutions**

- **Bluetooth Pairing and Stability**
    - **Challenge**: Ensuring stable Bluetooth connections for consistent message delivery.
    - **Solution**: The BluetoothSerial library provides reliable communication with paired devices, and error-handling mechanisms could be added for further stability.
- **OLED Display Constraints**

    - **Challenge**: Limited space on the OLED screen for displaying lengthy messages.
    - **Solution**: The message is truncated to 20 characters to maintain readability. Larger fonts are used for clarity, while a smaller font in the header indicates a new notification.
- **Real-Time Notification Display**
    - **Challenge**: Updating the display in real-time upon receiving new messages.
    - **Solution**: The `SerialBT.available()` function checks for new messages, and the `displayNotification()` function refreshes the OLED content as messages are received.

**Summary**

The Bluetooth Notification System provides seamless connectivity for receiving notifications on the smartwatch. Key challenges like message length constraints and Bluetooth stability were addressed through truncation and the use of a robust Bluetooth library. The system design ensures an engaging user interface, leveraging Bluetooth and OLED display to enhance smartwatch functionality.

3.**Calendar Functionality for Smart Watch**

## System Design

The calendar functionality in this smart watch project provides a real-time, monthly calendar view on an OLED screen, showing the current date, month, and year. Key design elements include:

- **Real-Time Updates**: The system updates the calendar information based on the current date and time.
- **OLED Display**: Displays the monthly calendar, with the current day highlighted.
- **Wi-Fi and NTP Integration**: Connects to a Wi-Fi network to sync the time from an NTP server, ensuring accurate date and time display.

## Features Used

### 1. Communication Interfaces (Wi-Fi)

- **Implementation**: Wi-Fi is configured to connect to the network and access the NTP server.
- **Description**: Wi-Fi connectivity enables the system to receive real-time updates from an NTP server, which ensures that the date and time are synchronized accurately. This approach minimizes the need for an RTC module by relying on internet-based time.

### 2. Time Management and Conversion

- **Implementation**: The time library is utilized to convert the epoch timestamp from the NTP server to year, month, and day components.
- **Description**: Using `time.h`, `timeClient`, and `gmtime()`, the system converts the epoch timestamp to calendar information, with conversions ensuring accurate display even across time zones or leap years. The NTP client updates every hour for consistent accuracy.

### 3. Display Interface (OLED Display)

- **Implementation**: The OLED screen (SH1106 driver) uses the U8g2 library to display calendar data.
- **Description**: Calendar data is displayed with clear formatting for month, day, and dates. The OLED screen's I2C protocol facilitates rapid, synchronous data updates without lag, ensuring that the current date and month are displayed accurately.

## Code Structure

- **Constants and Variables**: Wi-Fi credentials, display initialization, and NTP configuration.
- **Setup Function**: Initializes the serial monitor, connects to Wi-Fi, and sets up the OLED display and NTP client.

- **Loop Function**: Updates and displays the calendar based on the synced time from the NTP server.
- **Calendar Functions**:
    - `drawCalendar()`: Renders the month view with the current day highlighted.
    - `getDaysInMonth()`: Determines the number of days in the current month, accounting for leap years.
    - `calculateFirstDayOfMonth()`: Calculates the starting day of the month to align dates accurately.

## Challenges and Solutions

1. **Wi-Fi Connectivity**:
    - **Challenge**: Ensuring stable and consistent Wi-Fi connection.
    - **Solution**: Included a retry mechanism in the setup function, displaying progress and confirming connection for real-time synchronization.
2. **Highlighting the Current Date**:
    - **Challenge**: Visually indicating the current date without overcrowding the screen.
    - **Solution**: Used `drawCircle()` to create a distinct circle around the current day, drawing attention without obstructing other dates.
3. **Date Conversion and Accuracy**:
    - **Challenge**: Accurately converting the epoch time to day, month, and year.
    - **Solution**: Utilized `gmtime()` to handle conversion and timezone adjustments, reducing errors related to incorrect day alignments.
4. **Display Formatting**:
    - **Challenge**: Fitting the entire month view on a small screen.
    - **Solution**: Implemented smaller fonts and precise spacing, allowing all dates and day labels to be visible.

## Summary

This calendar functionality uses Wi-Fi to sync date and time, then displays a monthly calendar on the OLED. Epoch time from an NTP server ensures accuracy, and careful formatting enables the small screen to clearly display calendar information. State management and structured display logic provide a functional, visually pleasing user experience, adaptable to additional features like customizable time zones.

4.**Real-Time Clock Display Functionality in Smart Watch**

## Overview

This project displays the real-time clock on an OLED screen connected to an ESP32, utilizing NTP (Network Time Protocol) to retrieve and display the current time. By connecting to Wi-Fi, the device accesses an NTP server to keep accurate time without needing an RTC (Real-Time Clock) module. This functionality is ideal for a smartwatch setup, as it updates every second to show the current time accurately.

## Features and Functionality

1. **Wi-Fi Connectivity**:
    - **Description**: The device connects to a Wi-Fi network using SSID and password credentials to sync time.
    - **Purpose**: Wi-Fi allows the device to connect to the NTP server, ensuring the time is continuously synchronized, even after a reset or power cycle.
2. **NTP Time Synchronization**:
    - **Description**: The device synchronizes with an NTP server (`pool.ntp.org`) to retrieve the current time in UTC.
    - **Configuration**: Indian Standard Time (IST) is achieved by setting the GMT offset to 19800 seconds (UTC +5:30). The daylight offset is set to 0 as India does not observe daylight savings.
    - **Purpose**: Using NTP avoids the need for a separate RTC hardware module, making this design simpler and more reliable.
3. **OLED Display**:
    - **Description**: A 128x64 OLED screen with an SH1106 driver displays the current time, formatted as "HH:MM
      ".
    - **Font Selection**: A clear, readable font (`u8g2_font_ncenB08_tr`) is selected to ensure time readability at a glance.
    - **Layout**: The display is cleared and updated each second to prevent ghosting, and the time is formatted and aligned for clarity.

## Code Structure

- **Setup Function**:
    - **Purpose**: Initializes serial communication, connects to Wi-Fi, configures the time zone, and retrieves the initial time.
    - **Behavior**: Wi-Fi connection status is printed on the Serial monitor. Once connected, the device sets the time based on the NTP server and displays the current time in the console as a test of connectivity.
- **Loop Function**:
    - **Purpose**: Continuously updates and displays the current time on the OLED screen.

- **Behavior**: Retrieves the local time every second, formats it as `HH:MM:SS`, and displays it on the screen. The buffer is cleared each time to ensure smooth updates without overlapping digits.

## Key Components and Libraries

1. **WiFi Library**:
   - **Role**: Manages Wi-Fi connectivity, allowing the device to connect to the internet to access the NTP server.
2. **NTP Time Synchronization (`configTime()` function)**:
   - **Role**: Configures time zone settings and syncs time from an NTP server, utilizing `gmtOffset_sec` to adjust to IST.
   - **Functions Used**:
     - `configTime()` initializes the time configuration.
     - `getLocalTime()` retrieves the time and converts it to local timezone.
3. **OLED Display Library (U8g2)**:
   - **Role**: Handles the display functions, including text formatting, buffer management, and rendering on an SH1106 OLED display.
   - **Methods Used**:
     - `clearBuffer()` clears the display memory to prevent overlapping text.
     - `setFont()` sets the font size and style.
     - `drawStr()` prints text strings at specified coordinates.
     - `sendBuffer()` transfers the display data to the screen.

## Challenges and Solutions

1. **Wi-Fi Connectivity**:
   - **Challenge**: Ensuring stable Wi-Fi connection.
   - **Solution**: A loop in the `setup()` function retries the connection every second until it is successful, and progress is shown on the Serial monitor.
2. **Time Formatting and Display**:
   - **Challenge**: Formatting the time and displaying it clearly on a limited screen.
   - **Solution**: Used `strftime()` to format the time as "HH:MM ," providing a compact and readable layout.
3. **Continuous Display Update**:
   - **Challenge**: Refreshing the display every second without flickering or lag.
   - **Solution**: Cleared and updated the display buffer each second, using an efficient font size for optimal space utilization on the small OLED.

## Summary

This project effectively demonstrates a real-time clock functionality using NTP for accurate timekeeping, Wi-Fi for internet connectivity, and an OLED display for user interface. It is a compact, reliable, and accurate time display, with the potential for expanding functionality

like adding date, timezone adjustments, or low-power optimizations for better battery efficiency.

5.**Report for Stopwatch Project Using OLED Display and Button Interface**

## Overview

This project is a digital stopwatch implemented on an Arduino-compatible microcontroller with an OLED display and a button interface. The device displays elapsed time in the format `MM:SS:MMM` (minutes, seconds, milliseconds) and allows the user to start, pause, resume, and reset the stopwatch. The project utilizes the U8g2 library for controlling the OLED display and provides a simple, user-friendly interface through a single button.

## Features and Functionality

1. **OLED Display (SSD1306)**:
    - **Description**: A 128x64 SSD1306 OLED screen is used to display the stopwatch interface, including the current elapsed time, stopwatch mode, and status.
    - **Initialization**: The OLED is initialized with the U8g2 library, which simplifies text and graphic display control.
2. **Button Interface**:
    - **Description**: A button connected to a digital pin (`BUTTON_PIN`) enables the user to interact with the stopwatch.
    - **Modes**:
        - Mode 0 - Reset: Displays "STOPWATCH" with a prompt to press the button to start.
        - Mode 1 - Running: Starts or continues counting time from the paused state.
        - Mode 2 - Paused: Freezes the time display at the current value.
        - Mode 3 - Resume: Continues counting from the paused time without resetting.
3. **Stopwatch Functionality**:
    - **Start/Stop**: The stopwatch can be started, paused, and resumed. The button cycles through these states sequentially.
    - **Reset**: Resets the elapsed time to zero and returns to the initial display state.
4. **Formatted Time Display**:
    - **Display Format**: Time is displayed in `MM:SS:MMM` format, with larger font size for visibility.
    - **Real-Time Update**: The display refreshes every millisecond when running, offering a precise and smooth update for milliseconds.

## Code Structure

- **Setup Function**:
    - **Purpose**: Initializes the OLED display and button pin mode, then displays a brief splash screen with the project title.
    - **Splash Screen**: Displays "Project By: EDMH" and "Hyouka" on the OLED screen for 2 seconds before entering the main loop.
- **Loop Function**:

- **Purpose**: Manages button presses to update the stopwatch mode, calculates elapsed time, and refreshes the display.
- **Components**:
  - `handleButtonPress()`: Detects button press and cycles through modes.
  - `updateStopwatch()`: Updates the time values if the stopwatch is in running mode.
  - `updateDisplay()`: Updates the OLED with the stopwatch status and current time based on the mode.

## Key Components and Libraries

1. **U8g2 Library**:
   - **Role**: Handles text rendering and graphics on the OLED display.
   - **Methods Used**:
     - `begin()`: Initializes the display.
     - `clearBuffer()`, `sendBuffer()`: Clears and updates the display buffer.
     - `setFont()`, `drawStr()`: Sets font size and displays strings at specified coordinates.
     - `drawFrame()`: Draws a rectangular border around the stopwatch display area.
2. **Button Handling**:
   - **Debouncing**: Button state is checked and stored in `buttonPressed` to avoid multiple triggers during a single press.
   - **Mode Switching**: Button cycles through modes 0 to 3, each representing different states of the stopwatch.
3. **Time Calculations**:
   - **Millisecond Precision**: Uses `millis()` to track precise elapsed time.
   - **Mode-Based Adjustments**: Adjusts `startTime` based on the current mode (paused, resumed) to ensure accurate time tracking.

## Challenges and Solutions

1. **Button Debouncing**:
   - **Challenge**: Preventing multiple state changes with a single press.
   - **Solution**: Tracked the button state using `buttonPressed`, which changes only on a rising edge of the button press.
2. **Display Refresh**:
   - **Challenge**: Ensuring smooth and accurate display updates without flickering.
   - **Solution**: Cleared and refreshed the buffer each loop cycle to maintain a consistent display while in running mode.
3. **Time Synchronization Across Modes**:
   - **Challenge**: Calculating elapsed time correctly even after pauses and resumes.

- **Solution**: `pausedTime` stores the time when the stopwatch is paused, and `startTime` is adjusted to maintain accuracy when resuming.

## Summary

This stopwatch project demonstrates effective time-tracking, mode-switching, and display handling on a microcontroller. The project is well-suited for a beginner in embedded systems, as it introduces concepts like button interfacing, time calculations, and graphical display updates. This setup can be further extended with features like lap tracking, buzzer alerts, or automatic power-saving for a more advanced smartwatch application.

## 6. Temperature and Humidity Monitoring Project

---

**Overview and Purpose:**

The goal of this project was to create a compact and reliable temperature and humidity monitoring system using a DHT11 sensor for data collection and an OLED display for real-time data presentation. This project adheres to the requirements by integrating various technical features such as communication protocols, ADC interfacing, interrupt handling, and a display interface.

---

## System Design:

**1. Hardware Components:**

- **DHT11 Sensor:** Measures environmental temperature and humidity. The DHT11 uses a digital interface to send data.
- **OLED Display (SH1106, 128x64):** Provides a clear graphical interface for displaying the temperature, humidity, and heat index readings.
- **Microcontroller (e.g., Arduino/ESP32):** Manages communication between the sensor and display, processes sensor data, and handles display updates.

**2. Communication Protocols:**

- **I2C Protocol**: Used for communication between the microcontroller and the SH1106 OLED display, allowing for fast data transfer and simple wiring.

**3. ADC/Digital Interface:**

- **Digital Interface for DHT11**: The DHT11 sensor is connected via a digital GPIO pin, which simplifies data handling compared to analog sensors.
- The project did not require ADC since the DHT11 is a digital sensor, but the system could be extended with an analog temperature sensor if needed.

**4. NVIC (Nested Vector Interrupt Controller):**

- **Interrupt Handling**: Implemented by monitoring sensor data at regular intervals (every 2 seconds). Although no external interrupts were used, the NVIC was indirectly involved in the microcontroller's handling of timing functions and OLED updates.

**5. Display Interface:**

- **OLED Display**: The SH1106 OLED screen provides a graphical interface to show the current temperature, humidity, and calculated heat index. The display is refreshed every 2 seconds.

---

## Code Structure and Logic

The code follows a modular structure, divided into sections for sensor reading, display handling, and error-checking:

1. **Initialization (`setup`):**
    - Begins serial communication for debugging.
    - Initializes the DHT11 sensor.
    - Initializes the OLED display over I2C.
2. **Main Loop (`loop`):**
    - **Sensor Reading and Error Checking**: Reads humidity and temperature in both Celsius and Fahrenheit, checks for failures, and displays an error message on the screen if the sensor reading fails.
    - **Data Calculation**: Calculates the heat index to provide a "feels-like" temperature based on humidity levels.
    - **Data Display**: Clears the OLED buffer and presents a clean display layout with the temperature, humidity, and heat index readings.
3. **Display Layout and Buffer Handling**:
    - Utilizes `u8g2.clearBuffer()` to clear the display before writing new data.
    - Formats and displays each metric using `setFont`, `drawStr`, and `setCursor` methods for consistent data arrangement and readability.
    - Displays units for temperature (°C, °F) and humidity (%), enhancing clarity.

---

## Challenges and Solutions

1. **Handling Sensor Failures**:
    - **Challenge**: The DHT11 sensor occasionally fails to provide valid readings, which could disrupt the data displayed.
    - **Solution**: Implemented error checking using `isnan()` to detect failed readings. If a failure occurs, an error message is displayed on both the Serial Monitor and the OLED screen to alert the user.
2. **Optimizing Display Updates**:
    - **Challenge**: Refreshing the OLED display without introducing noticeable flicker.
    - **Solution**: Used `u8g2.clearBuffer()` only when updating, which ensures smoother display transitions.
3. **Timing and Interrupt Handling**:
    - **Challenge**: Avoiding blocking delays while ensuring accurate 2-second intervals for updates.
    - **Solution**: The delay function was set to 2000ms, which is sufficient for periodic updates in a non-time-critical application. For larger systems, this could be optimized with non-blocking timing functions or timer-based interrupts.

4. **Heat Index Calculation**:
    - **Challenge**: The "feels like" temperature is challenging to display in a limited space.
    - **Solution**: Used a calculated Celsius heat index and clearly labeled it on the OLED display as "Heat Index" to provide the user with the most relevant data in a compact form.

## Conclusion and Recommendations for Expansion

This project successfully meets the requirements for temperature and humidity monitoring with real-time display. The code and hardware integrate communication protocols, digital interfaces, and an OLED display, making it an efficient and user-friendly weather station. For future enhancements:

1. **Data Logging**: Store historical temperature and humidity data for trend analysis.
2. **Wi-Fi or Bluetooth Connectivity**: Enable remote monitoring by transmitting data to a mobile device or cloud.
3. **ADC Integration**: Add an analog temperature sensor to enhance system versatility.

The project demonstrates effective integration of hardware and software, providing reliable environmental data monitoring in a compact, user-friendly package

**Conclusion:**

The smartwatch project successfully integrates multiple useful functionalities that cater to daily needs such as productivity tracking, time management, environmental awareness, and device connectivity. These features offer a highly versatile and practical solution for users, enhancing their experience by providing comprehensive tools that go beyond the standard features of typical smartwatches. The combination of time-based functionalities (alarm, RTC, stopwatch), environmental tracking (temperature), and Bluetooth connectivity makes the smartwatch a powerful and user-friendly embedded system for both personal and professional use.