Abdullah Gül University

Department of Computer Engineering

Embedded Systems

Smart Watch Project Progress Report

Instructor

Abdulkadir Gülşen

09.05.2025

Tolgahan Keleş

Muhammed Çağrı Akkuş

Umut Kaya

1. Contents

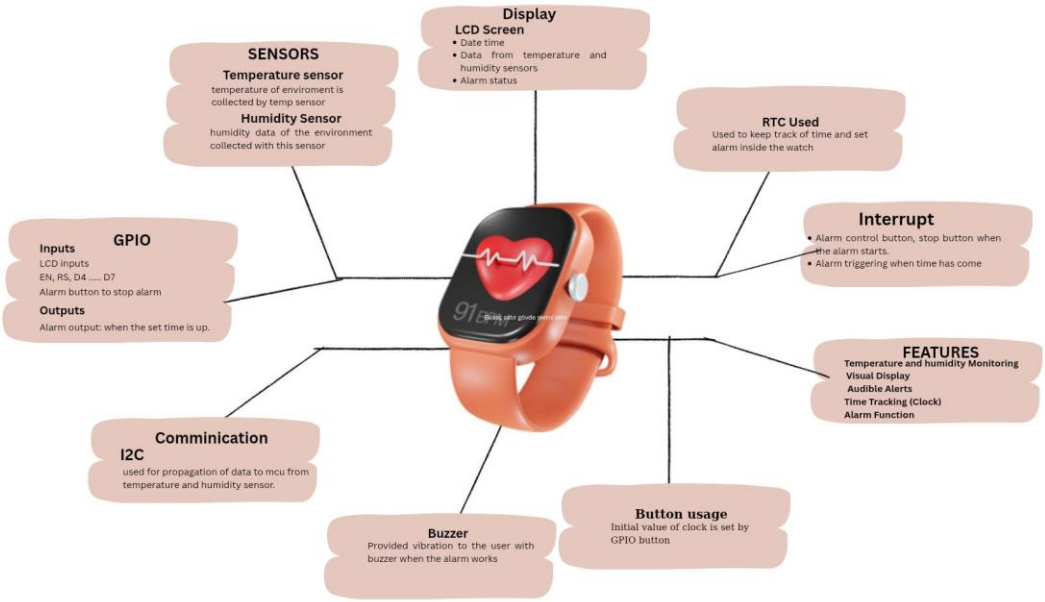
2. Objective

The primary goal of this project is to design and implement a smart watch system using STM32F401VETx. The system offers a real-time clock (RTC), a chronometer (stopwatch) feature, an LCD for displaying time-related data, and an emergency alert feature. Upon triggering an emergency, the system plays a warning tone via a speaker driven by PWM signals to simulate a nuclear siren-like effect. The entire logic is managed using CMSIS-RTOS2 (Keil RTX5), ensuring modular and concurrent thread execution. All hardware was simulated using Proteus 8.17 and firmware development carried out in Keil MDK with CubeMX configuration.

3. System Design Overview

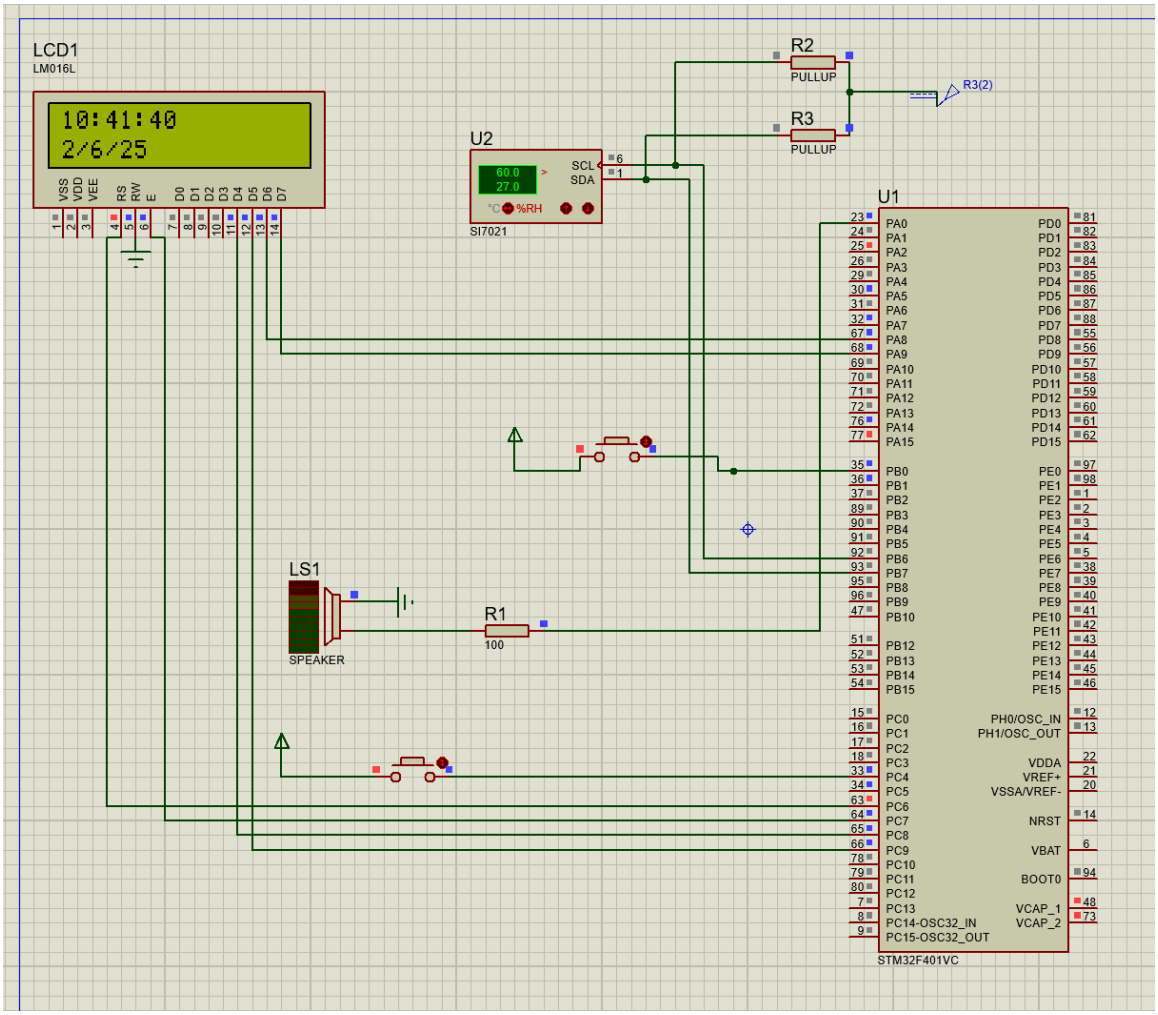
The smart watch architecture includes:

- STM32F401VETx microcontroller
- RTC peripheral with LSE oscillator
- 16x2 LCD (GPIO-controlled, 4-bit mode)
- Speaker connected to PWM output
- Emergency push-button with EXTI interrupt
- CMSIS-RTOS2 multithreaded environment (LED, LCD, and alarm threads)
- Chronometer system based on RTC timer and button inputs
- SI7021 Temperature & Humidity Sensor



4. Block Diagram and Circuit

Below is the Proteus simulation schematic of the implemented design:



5. Features of the Smart Watch

- Real-time **display of temperature, current time and date** using RTC.
- **LCD** shows regular time or emergency warnings based on system state.
- **Chronometer** mode enabled via button input, displays elapsed time in seconds
- **External interrupt (EXTI)** toggles emergency mode.
- PWM signal generates repeating **emergency alarm via speaker**.
- Entire structure uses **CMSIS-RTOS2** threads for concurrency.
- **Synchronization via RTOS**.

6. Price List of Components

Component	Description	Approximate Price (TL)
STM32F401VETx development board	Microcontoller Unit	₺1800
SI7021 Sensor	Temperature and humidity	₺295,19
16x2 LCD Display	Visual output	₺98,33
Speaker	Sound alert module	₺320
Button	State management	₺40

7. Used Modules and Features

Module/Feature	Some of the possible types	We used these types:
GPIO	● ⭐Digital Output ● ⭐Digital Input ● ⭐Other	● ⭐Digital Output ○ Push/Pull x9 ● ⭐Digital Input ○ External Interrupt x2
Communication	● UART ⭐, ● SPI ⭐ ⭐, ● I2C⭐ ⭐, ● CAN⭐ ⭐ ⭐, Others ● Using multiple devices at the same communication bus ⭐ ⭐ ⭐	● I2C ○ SI7021 ● UART ○ Debugging
Watchdog timer	⭐	
Interactivity (Leds, buttons, switches, touch etc.)	⭐ ⭐	● 16x2 Display ● Speaker ● Button ● SI7021
Using sensors	Single ⭐, few ⭐ ⭐, many or advanced one ⭐ ⭐ ⭐ ⭐	● Sensor x1 types ○ SI7021 temperature & humidity sensor
Actuators	● Motors, ● ..	● Speaker ● LCD

Timers	Systick ⭐, Advanced-basic Timers ⭐ ⭐, RTC alarm ⭐ ⭐	• TIM2
		  ○ PWM
		• RTC Alarm
		  ○ Used for alarms

Usage of polling	❌❌❌	
Usage of Interrupts	No interrupt ❌❌❌, Single ⭐, few ⭐ ⭐, many with different priorities ⭐ ⭐ ⭐	• To read button • For alarms
Error handling	No error handling ❌ , few ⭐ ⭐, full ⭐ ⭐ ⭐	• For I2C communcation
Analog-digital Converter	ADC ⭐ ⭐, DAC ⭐ ⭐	
Advanced Things that no code is provided during the course such as DAC, CAN etc.	extra ⭐ ⭐ ⭐	• RTC • Keil RTX5 • Library Usage • I2C communication
Power saving	Sleep - standby - wakeup ⭐ ⭐ ⭐	
DMA	⭐ ⭐ ⭐	
Ethernet-internet-wifi	⭐ ⭐ ⭐	
Writing own driver library for a peripheral	⭐ ⭐ ⭐ ⭐ ⭐	
bluetooth	⭐ ⭐ ⭐ ⭐	
PCB	External electronics design ⭐ ⭐, using a different board ⭐ ⭐ ⭐, using MCU unit on your own design without the development board	

	⭐⭐⭐⭐⭐	
Usage of advanced tools e.g., Matlab, CubeAI etc. (Matlab code should run on MCU)	⭐⭐⭐⭐⭐	
Real time OS	⭐⭐⭐⭐⭐	• Keil RTX5

8. Implementation

The implementation phase includes hardware simulation, microcontroller configuration, RTOS integration, PWM signal generation, and real-time clock synchronization through an LCD display. Below are the key steps:

A. Hardware Simulation Setup

The system was designed and simulated in Proteus 8.17. Virtual components used include:

- STM32F401VETx microcontroller
- 16x2 LCD display (4-bit mode GPIO)
- Speaker (connected to TIM2 PWM output)
- Push button (connected to EXTI interrupt line)

B. STM32CubeMX Configuration

- GPIOs configured for LCD, LED, and push button
- TIM2 initialized for PWM output
- RTC enabled with LSE oscillator
- NVIC interrupt priority setup for EXTI
- Clock configuration and RCC settings adjusted

C. Firmware Development in Keil uVision (CMSIS-RTOS2)

- Implemented three CMSIS-RTOS2 threads for LED blinking, LCD updating, and emergency alarm control
- RTC time polled every second and displayed on LCD
- External interrupt toggles emergency mode (EXTI callback)
- PWM output used to flash LED and play tone on buzzer
- RTC alarm configured to trigger scheduled speaker activation
- osDelay used in threads instead of HAL_Delay for non-blocking execution

D. Chronometer Thread Implementation

A dedicated timer_thread was added to handle stopwatch functionality. It operates based on a state variable (e.g., isTimerStarted) toggled via a push button using EXTI. When active, a counter variable is incremented every 100 ms using osDelay(100) to simulate stopwatch behavior. The elapsed time is dynamically printed on the LCD screen when chronometer mode is active. This thread remains idle otherwise, saving CPU resources.

9. Feature Justification and Code Explanation

This section explains the major components and their implementation details in the smart watch system, including the justification for their usage and the underlying code logic.

A. GPIO – General Purpose I/O Configuration

GPIO pins are used to drive the LCD screen, onboard LED, and alarm output pin. Additionally, emergency and reset buttons are connected as external interrupt inputs. Configuration is handled in gpio.c. The emergency and reset buttons are assigned EXTI0 and EXTI4 lines respectively to enable interrupt-driven response without polling. All output pins are set to push-pull mode with no pull-up/down and low frequency, ensuring stable digital output.

B. External Interrupts – Emergency Button Detection

A physical button is used to control emergency state (PC4). These are configured as external interrupts with rising-edge trigger with pull-down resistors. When pressed, the HAL_GPIO_EXTI_Callback() function sets the isEmergency flag, which is then handled in the emergency thread. This ensures an immediate and interrupt-driven response.

C. PWM – Emergency Siren via Frequency Modulation

The project includes a siren sound generated through a speaker using PWM (TIM2, Channel 1). A custom set_pwm_freq(float freq) function updates the frequency in real time by adjusting the ARR (auto-reload) register of the timer. The emergency_thread performs a frequency sweep between 1000 Hz and 2800 Hz repeatedly, simulating a siren similar to emergency vehicles. This is achieved by gradually increasing and decreasing the frequency in short delays, creating a dynamic sound pattern.

D. RTC – Real-Time Clock and Alarm Functionality

The internal RTC is configured using the LSE oscillator for better accuracy. Time and date are initialized in rtc.c and updated dynamically in the LCD thread. Alarm A is enabled to

trigger a specific time-based event. When the alarm time matches, the RTC_Alarm_IRQHandler() sets the isAlarmOn flag and drives the alarm_output_Pin high, which can activate an external circuit or buzzer. This provides a basic scheduling mechanism, simulating smart watch alarm functionality.

E. LCD – Time Display and Emergency Notification

A 16x2 LCD is connected via 4-bit mode to conserve GPIO pins. Initialization and display control are handled through a user-defined LCD.h library. The lcd_thread updates the screen every second to show current time and date using RTC data. If an emergency state is active, the display overrides with “!!! HELP !!!” warning messages, providing visual alert alongside the buzzer.

F. Chronometer – Stopwatch Feature Using CMSIS-RTOS2 and Systick Timer

The chronometer feature enables the smart watch to function as a stopwatch, tracking elapsed time in seconds. A new RTOS thread (**timer_thread**) was implemented for this purpose. When the chronometer mode is activated by a button (EXTI), the thread starts counting using the **osDelay()** function at 100 ms intervals, providing a fine resolution. The counter is displayed live on the LCD using the second row. This feature demonstrates advanced thread usage and user-interactive input within an RTOS environment. It also showcases integration of timer logic with the display system, using thread synchronization to avoid conflicts with the regular clock display.

G. CMSIS-RTOS2 (Keil RTX5) – Thread-Based Architecture

The system utilizes CMSIS-RTOS2 (Keil RTX5) for structured multi-threading, enabling concurrent and modular control of various features in the smart watch system. Each functional task runs in its own thread, ensuring real-time responsiveness without blocking other operations. The following threads are created and initialized inside the app_main() function:

- **lcd_thread**: Continuously updates the 16x2 LCD screen. It displays the current time and date during normal operation. If the emergency mode is triggered, it shows a visual alert message ("!!! HELP !!!"). During chronometer mode, it displays the elapsed time. If the alarm is active, it shows an "!!! ALARM !!!" message.

- **emergency_thread**: Simulates a nuclear-style siren by dynamically modulating PWM frequency through TIM2. It performs a frequency sweep between 1000 Hz and 2800 Hz to produce a realistic alert sound and stops automatically after five cycles.

- **alarm_thread**: Monitors the isAlarmOn flag and sets the alarm output GPIO pin high when triggered. This pin can be used to drive additional hardware like external buzzers or LEDs.

- **timer_thread**: Implements chronometer (stopwatch) functionality. When the isTimerStarted flag is active, it calculates the elapsed time using the RTC and stores it in a global counter (elapsedSeconds) for display.
- **uart_thread**: Sends a periodic dummy message ("as\r\n") through UART2 every second. This thread can be adapted for debugging or data logging in future extensions.

This architecture ensures each module operates independently while remaining synchronized via global flags and shared variables. By avoiding blocking calls and using osDelay() instead of HAL_Delay(), the system maintains real-time performance. This modular approach enhances scalability, allowing new features such as Bluetooth communication, step counters, or additional sensors to be integrated with minimal impact on the existing structure.

H. RTC Alarm Integration with GPIO Output

The RTC alarm system is not only a passive clock but also triggers events. When alarm A is triggered, the system sets isAlarmOn = true and writes a high logic to the alarm_output_Pin. This output can be used to activate an additional alert system. The code for this is implemented inside the RTC_Alarm_IRQHandler() function and verified using Proteus simulation.

I. Justification of Emergency Siren Design

Instead of a basic on/off buzzer, a **variable frequency PWM signal** creates a more realistic and attention-grabbing emergency siren sound. This feature simulates critical alerts (like radiation leaks or nuclear plant warnings) in real-life scenarios. The implementation mimics real-world signal patterns used in emergency systems, increasing both functionality and realism of the smart watch system.

10.Challenges and Solutions

- **Proteus did not support FreeRTOS well → switched to Keil RTX5**.

The initial plan was to use FreeRTOS; however, Proteus lacked compatibility with it during simulation. As a result, we transitioned to Keil RTX5 (CMSIS-RTOS2), which provided better simulation stability and integration with Keil MDK.

- **Keil MDK Lite linker limit → trimmed unused modules.**

The 32KB code size limit of Keil MDK Lite forced us to remove certain peripheral drivers and unused features to fit within the allowed memory. This constraint limited how many features we could actively run in the system.

- **Buzzer PWM required fine-tuning in simulation (Proteus timing issues).**

The PWM output to the buzzer had noticeable timing discrepancies in Proteus. We had to empirically adjust the frequency sweep delay and duty cycle settings to achieve a siren effect that resembled real-world emergency alarms.

- **I2C sensors failed in Proteus → temperature sensing discarded.**

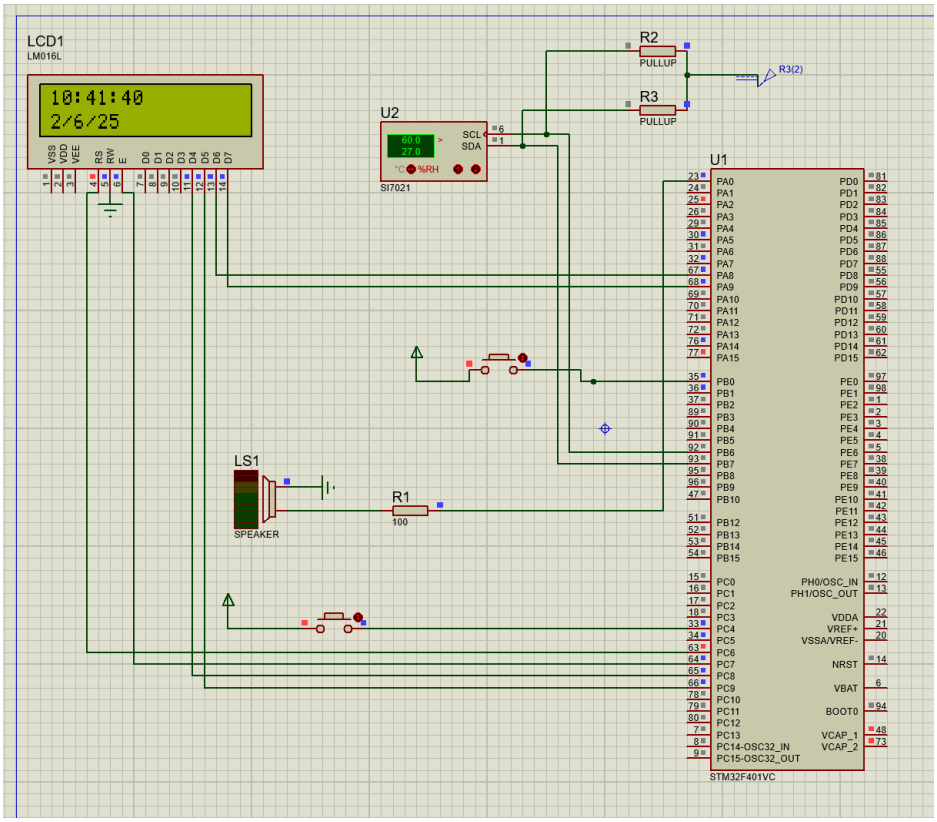
While we included SI7021 for temperature sensing in our code, Proteus did not emulate I2C communication correctly, causing the sensor to fail. As a result, temperature sensing had to be removed from the final simulation output, although the code remains in place for future physical implementation.

- **RTC alarm functionality partially failed due to code size limits caused by I2C modules.**

The inclusion of I2C-related drivers and sensor logic significantly contributed to hitting the Keil Lite linker limit. This overflow interfered with proper operation of other time-sensitive features like the RTC alarm interrupt. Due to restricted space, parts of the alarm activation logic had to be simplified or commented out, which led to incomplete functionality during simulation.

11.Testing and Screenshot

The following image shows the system simulation running in Proteus with LED, LCD, and emergency alarm:



12.Team Contribution

- Tolgahan Keleş (40%): RTOS & PWM implementation, debugging

- Muhammed Çağrı Akkuş (30%): LCD display and RTC integration

- Umut Kaya (30%): Proteus design, EXTI logic, documentation

13.Conclusion

This project demonstrates the design, development, and simulation of a wearable embedded system—namely, a smart watch—featuring real-time monitoring, user interactivity, and emergency alert capabilities. The system brings together essential embedded components such as an RTC module for accurate timekeeping, a PWM-controlled buzzer for dynamic alarm generation, external interrupt-driven emergency buttons for user-triggered actions, and a 16x2 character LCD for real-time data visualization. The inclusion of a chronometer (stopwatch) further enhances the utility of the device, aligning it more closely with real-world wearable technology.

The smart watch not only displays the current time and date via the RTC, but also handles critical events such as emergency alerts triggered by user input or predefined alarms. Both visual and auditory feedback are provided to ensure user awareness in critical situations. These features are carefully managed through a modular, multi-threaded architecture based on CMSIS-RTOS2 (Keil RTX5), allowing each subsystem to operate concurrently without blocking the others. This thread-based structure ensures real-time responsiveness and clean task separation, which is critical in embedded safety-oriented applications.

Furthermore, the system was implemented using industry-standard tools such as STM32CubeMX for peripheral configuration and Keil MDK for firmware development. It was fully simulated in Proteus 8.17, providing a complete virtual test environment. Despite challenges such as linker size limitations and I2C simulation issues in Proteus, the system was successfully adapted and optimized for functional demonstration. The project's siren system, implemented with variable PWM frequencies, realistically mimics emergency warning signals used in industrial and public safety domains—showcasing not just technical know-how, but thoughtful design inspired by real-world applications.

Overall, this project is a strong representation of the embedded systems design principles taught in EE304. It reinforces key concepts including peripheral driver integration, interrupt handling, real-time operating systems, analog/digital signal management, and system simulation. By overcoming practical development challenges and implementing a robust, modular design, students have demonstrated both engineering proficiency and an

understanding of how embedded systems contribute to intelligent, safety-critical products in modern technology landscapes.