AIN SHAMS UNIVERSITY FACULTY OF ENGINEERING

CSE211s: Introduction to Embedded Systems

Spring 2025

Name	ID
Ammar Ahmed	2200262
Omar Mohamed	2201058
Tarek Hazem	2200680

Introduction:

This report describes the implementation of a project for the CSE211s Introduction to Embed ded Systems course, Spring 2025. The project utilizes a NUCLEO-F401RE board interfaced with an Arduino Multifunction Shield to implement a Real-Time Clock (RTC) and an analog voltage display. The system displays elapsed time in minutes and seconds on a 7segment dis play, with the ability to reset the clock using switch S1. When switch S3 is pressed, the display shows the voltage from an on-board potentiometer, measured via the onchip ADC. The report details the code structure, startup code, main function, and Interrupt Service Routines (ISRs) of the provided C program.

Code Structure:

The program is a single C source file developed using the mbed framework, which abstracts lowlevel hardware details for the NUCLEO-F401RE. The code is organized into sections for hardware configuration, helper functions, and the main program loop, ensuring modularity and clarity. The structure includes:

- **Header Inclusion:** The mbed.h header provides APIs for GPIO (DigitalOut, DigitalIn), analog input (AnalogIn), and timing functions (wait_us, get_ms_count)
- **Pin Assignments and Global Variables:**

– Shift Register Pins: shiftDataPin (D8), shiftClockPin (D7), and latchPin (D4) control the 7-segment display via a shift register.

– Input Pins: voltagePin (A0) reads the potentiometer voltage, resetButton (A1) is switch S1, and voltageButton (A3) is switch S3.

– Display Constants: segmentDigits[] defines 7-segment patterns for digits 0–9 (com mon anode, active LOW). digitSelectionMask[] specifies masks for selecting digits D0 D3.

Helper Functions:

– shiftDataToRegister(): Sends 16-bit data (8-bit segment pattern + 8-bit digit mask) to the shift register to update the display.

– showVoltage(): Displays the potentiometer voltage in centivolts (e.g., 3.45V as 345) with a decimal point.

– readStableVoltage(): Averages 50 ADC readings to provide a stable voltage measure ment (0–5V).

– showTime(): Displays time in MM:SS format (e.g., 12:34).

- **Main Function:** Implements the core logic for timekeeping, voltage measurement, and display switching based on button inputs.

The code follows a simple, layered architecture:

- **Hardware Layer**: Managed by mbed, handling GPIO, ADC, and timer operations.
- **Application Layer:** Implements RTC and voltage display logic, meeting the project re quirements

Startup Code(startup_stm32f401xe.s):

The startup_stm32f401xe.s file is an assembly language startup code for STM32F401xe microcontrollers, designed for the MDK-ARM toolchain. It performs essential initialization tasks to prepare the Cortex-M4 processor to execute the main application. Below is a breakdown of its structure and functionality.

1. Purpose of the Startup Code:

- Initializes the stack pointer (SP).
- Sets up the program counter (PC) to point to the Reset_Handler.
- Defines the vector table, mapping exception and interrupt service routines (ISRs).
- Transfers control to the C library's __main function, which eventually calls the user's main() function.
- Configures the processor in **Thread mode** with **Privileged priority** and the **Main Stack** after reset.

2. Key Directives and Sections

Directives

- PRESERVE8: Ensures 8-byte stack alignment, required by the ARM Cortex-M4 architecture.
- THUMB: Specifies that the code uses the Thumb instruction set, which is compact and efficient for Cortex-M4.

Sections

- AREA RESET, DATA, READONLY: Defines the vector table in a read-only data section named RESET.
- AREA |.text|, CODE, READONLY: Contains the executable code (e.g., Reset_Handler and exception handlers)

3. Vector Table (__Vectors)

The vector table is a critical data structure mapped to address 0x00000000 at reset. It contains:

- Stack Pointer Initialization: The first entry (DCD |Image$$ARM_LIB_STACK$$ZI$$Limit|) sets the initial stack pointer to the top of the stack, defined by the linker.

- Reset Handler: The second entry points to Reset_Handler, the first function executed after reset.
- Exception Handlers: Entries for Cortex-M4 exceptions (e.g., NMI, HardFault, MemManage, etc.).
- Interrupt Handlers: Entries for peripheral interrupts (e.g., WWDG, EXTI, DMA, etc.).
- Reserved Entries: Set to 0 for unused or reserved slots.

The vector table is exported as __Vectors, __Vectors_End, and __Vectors_Size for use by the linker and debugger.

Key Vector Table Entries

- **Exceptions**:
  - NMI_Handler: Non-Maskable Interrupt.
  - HardFault_Handler: Handles critical errors (e.g., invalid memory access).
  - MemManage_Handler, BusFault_Handler, UsageFault_Handler: Handle memory, bus, and usage faults.

    - SVC_Handler, PendSV_Handler, SysTick_Handler: System-level handlers for supervisor calls, context switching, and system tick interrupts.
- **Interrupts**:
    - Peripheral-specific ISRs (e.g., WWDG_IRQHandler, EXTI0_IRQHandler, DMA1_Stream0_IRQHandler).
    - Examples include handlers for timers (TIM1, TIM2), I2C, SPI, USART, and USB OTG.

4. Reset Handler (Reset_Handler)

The Reset_Handler is the entry point after a reset. It:

1. Calls SystemInit (a C function) to configure the microcontroller's system-level settings (e.g., clock, PLL).
2. Branches to __main (provided by the C library), which initializes the C runtime environment (e.g., stack, heap, global variables) and calls the user's main() function.

The Reset_Handler is marked as WEAK, allowing it to be overridden by user-defined code if needed.

5. Exception and Interrupt Handlers

- **Dummy Handlers**: All exception and interrupt handlers (e.g., NMI_Handler, HardFault_Handler, WWDG_IRQHandler) are implemented as infinite loops (B .) by default. These are placeholders that can be overridden by user-defined implementations.
- **Weak Symbols**: Handlers are exported with the WEAK attribute, allowing users to define custom handlers in their code without modifying the startup file.
- **Default Handler**: A single Default_Handler procedure serves as the entry point for all interrupt handlers. It loops indefinitely, ensuring the system halts if an unhandled interrupt occurs.

6. Memory and Stack Initialization

- The stack is initialized via the vector table's first entry, which points to |Image$$ARM_LIB_STACK$$ZI$$Limit|, a linker-defined symbol for the top of the stack.
- Heap initialization is handled by the C library's __main function, not directly in the startup code.

- The startup code does not explicitly initialize the heap or other memory sections, as this is typically managed by the linker script and C runtime.

Main Function:

The main() function serves as the entry point for the application logic, running in an infinite loop to meet the projects RTC and voltage display requirements. Its key tasks are:

- **Initialization:** Sets lastUpdateTime to the current system time using get_ms_count() to start the RTC from zero.
- **Main Loop:**

– **Timekeeping:**

∗ Reads the current time (currentTime) via get_ms_count().

∗ Increments secondsCounter every 1000 ms (1 second).

∗ Resets secondsCounter to 0 and increments minutesCounter when secondsCounter reaches 60.

– **Display Control:**

∗ If voltageButton (S3, A3) is pressed (active LOW, voltageButton == 0):

· Calls readStableVoltage() to measure the potentiometer voltage (0– 5V).
· Convertsthevoltage to centivolts (e.g., 3.45V to 345) and displays it using showVoltage().

∗ Otherwise, displays the RTC time (MM:SS) using showTime(minutesCounter, secondsCounter).

– **Reset Logic:** Resets minutesCounter and secondsCounter to 0 if:

∗ resetButton (S1, A1) is pressed (active LOW, resetButton == 0). ∗ minutesCounter reaches 100 (additional feature beyond project requirements)

Project Requirements:

- **RTC:** The RTC starts from 00:00 after reset and displays minutes (D3, D2) and seconds (D1, D0), as required.
- **S1 Reset:** Pressing S1 (resetButton) resets the RTC to 00:00 at any time.
- **Voltage Display**: Pressing S3 (voltageButton) displays the potentiometer voltage (0–5V, scaled from ADCs 0.0–1.0 range) in volts with a decimal point (e.g., 3.45V). Releasing S3 resumes RTC display without stopping the clock.
- **Potentiometer Voltage**: The on-board potentiometer provides 0V (minimum) to 5V (max imum), as the ADC reference is tied to the 5V supply

Interrupt Service Routines (ISRs):

The provided code does not define any userimplemented ISRs, as the projects functionality is achieved through polling. The mbed framework handles interrupts internally for:

- **Timer Interrupts:** The get_ms_count() function relies on a system timer (e.g., SysTick), with ISRs managed by mbed.

- **ADCInterrupts:** The AnalogIn object (voltagePin) may use ADC interrupts for conver sions, abstracted by mbed.

- **GPIO Interrupts:** Switches S1 (resetButton) and S3 (voltageButton) are polled in main(), not interrupt-driven.

Polling is sufficient for this project due to the low frequency of button presses and the sim plicity of the display updates. If interrupts were needed (e.g., for debouncing S1/S3), mbeds InterruptIn class could be used, with an ISR like:

```
void button_isr() {
    // Handle button press (e.g., set a flag)
}
```

Conclusion:

The project successfully implements an RTC and voltage display using the NUCLEO-F401RE and Arduino Multifunction Shield. The code is structured as a single C file with mbed framework dependencies, featuring:

- **Code Structure:** Clear organization with pin definitions, helper functions, and a main loop.
- **Startup Code:** Handled by mbed, initializing hardware and calling main().
- **Main Function:** Manages RTC, voltage measurement, and display switching via polling.
- **ISRs:** None user-defined**;** mbed handles timer and ADC interrupts.

Links:

- Github Repository

https://github.com/OmarMohammed299/Embedded-Project.git

- Video

https://drive.google.com/file/d/1xihj5X5wrXCdcJjtQfc0rjgoPRfWbtPO/view?usp=sharing

- Code

https://github.com/OmarMohammed299/Embedded-Project/blob/main/Code%20and%20Startup%20code/Code.txt

- Sartup Code

https://github.com/OmarMohammed299/Embedded-Project/blob/main/Code%20and%20Startup%20code/Startup%20code.txt