ECEN 5803

Mastering Embedded System Architecture

(Fall-2023)

PROJECT-1

Vortex Flowmeter Embedded System

Proof of Concept

Date

10/24/2023

Guided By

Prof. Timothy Scherr

Submitted By

Ajay Kandagal

Kshitija Dhondage

Contents

Executive Summary

This executive summary outlines our journey in developing a Proof of Concept (POC) for "**Sierra's Model 240V In-line Volumetric Vortex Meter**" while adhering to Sierra Instrumentation's project requirements. Our focus involved evaluating the STM32F401RE MCU using the Nucleo F01RE evaluation board. To facilitate this assessment, the project was divided into separate modules, each dedicated to the thorough evaluation of the STM32F401RE MCU within the mbed studio development environment using bare metal programming.

In the first module, the successful evaluation of processor computing power was carried out. A basic understanding of the ARM Cortex M4 instruction set (for assembly) and the memory model was achieved. The MCU was evaluated based on its arithmetic operations support which was essential for calculating the volumetric flow rate. In the following module, the evaluation of the MCU's support for external hardware input/output peripherals and its interaction with communication protocols such as I2C for connecting with the DS1631 temperature sensor, SPI for the 16x2 LCD, ADC for POT, and UART for a serial debug monitor was conducted.

Furthermore, an evaluation of the MCU's performance within a real-time environment (RTOS) was carried out by implementing the CMSIS RTOS in Keil using the RTX library. In the next module, we got familiar with STMcube IDE, and how to auto-configure the clocks (system, ADC).

Then we moved to, implement the bare metal O.S embedded system, and evaluate the MCU timers and interrupt performance and how to use that to allocate the CPU to different resources. In the next module, we familiarized with the STMcube IDE, learning how to auto-configure system and ADC clocks. Subsequently, we progressed to implementing a bare-metal embedded operating system, where we assessed the performance of the MCU's timers and interrupts, as well as the allocation of CPU resources to different tasks. Afterward, we evaluated the MCU's performance by employing the Dhrystone benchmark algorithm to determine the DIMPS (Dhrystone MIPS) of the processor and ensure that it met the prescribed DIMPS requirements.

In the final module, we designed a frequency simulation in MATLAB. Following that, we developed a frequency detection algorithm utilizing timer interrupts. We configured the system to read data from the board's internal temperature sensor using the ADC, which allowed us to calculate the volumetric flow rate of the fluid. Lastly, we conducted an analysis of the MCU's power consumption in both full run mode and low power mode.

After a comprehensive evaluation of both the hardware and software capabilities of the ST STM32F401RE Microcontroller, we can confidently conclude that this MCU not only meets all the performance criteria required for the vortex flowmeter but also aligns seamlessly with the cost and size specifications. This means that the STM32F401RE MCU is a good fit for the Sierra's vortex flowmeter.

Problem Statement and Objectives

Problem Statement:

The task was to design and implement a Proof of Concept (POC) for Sierra's Volumetric Vortex Flow Meter, following the **Schedule, Cost, and Budget** requirements provided by Sierra Instrumentation.

Objectives:

The objective was to identify potential solutions for the embedded system implementation of the product (Volumetric Vortex Flow Meter) by evaluating the hardware and software capabilities of the ST STM32F401RE Microcontroller Unit (MCU) using the ST Nucleo 401 board.
This goal is achieved by completing the analysis and development step by step, as outlined in the following six modules:

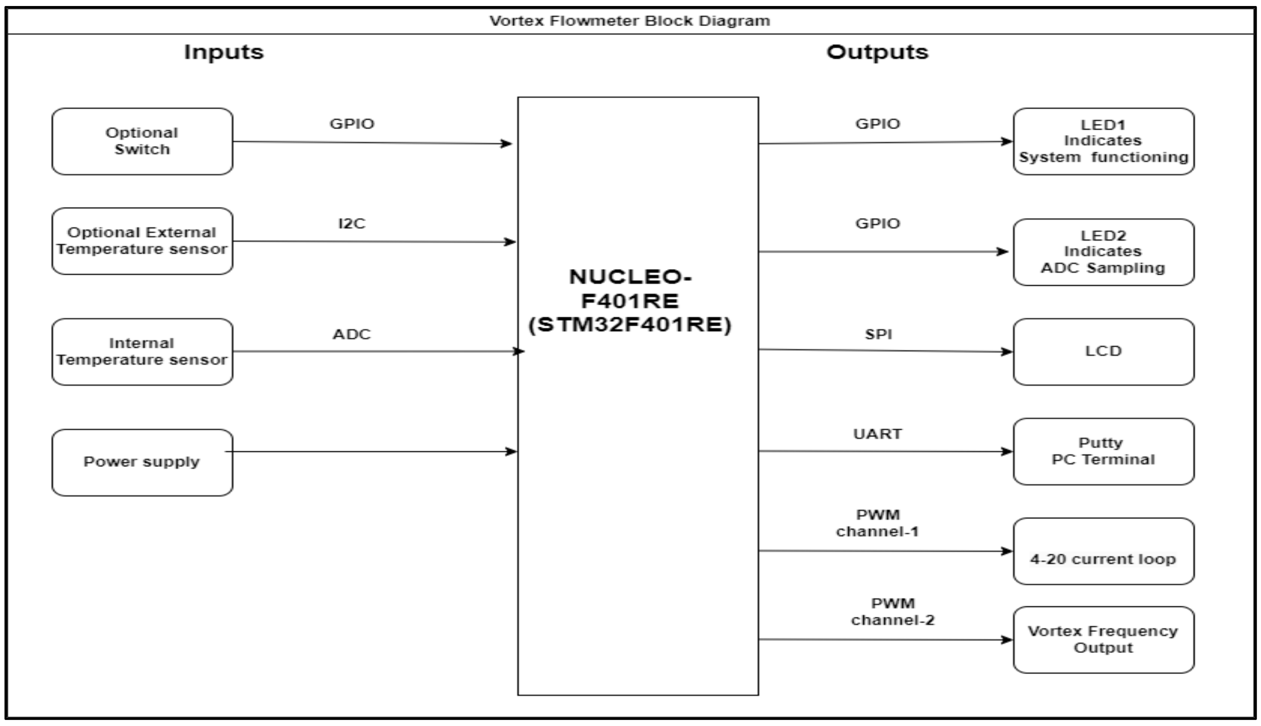
- **Module 1:** Module 1 objective was, to understand Cortex M4 instructions and implement an assembly code subroutine for approximating the square root of an argument using the bisection method. Additionally, to conduct an analysis of the Cortex M4 memory architecture and examine how memory consumption differs when utilizing C and assembly programming languages.
- **Module 2:** The objective for module 2 was, to accomplish the interfacing of external hardware with the MCU using communication protocols. This includes interfacing the DS1631 temperature sensor via I2C, connecting a 16x2 LCD through SPI, and generating audio waves using PWM. Additionally, the task involves displaying the temperature value on the UART terminal and LCD.
- **Module 3:** The objective of module 3 was, to implement CMSIS RTOS threads for efficient hardware utilization for the same code developed in module 2.
- **Module 4:** Module 4 objective was, to Create a start-up code using the STMCUBE configuration software to auto-configure the system and ADC clock.
- **Module 5:** The objectives for module 5 were, to construct a bare-metal embedded operating system featuring a super loop and a timer interrupt. Using these timer interrupts, control the allocation of the CPU resources to various tasks. Furthermore, a serial debug monitor was designed and developed, offering user commands for displaying register and memory dumps, as well as stack information. The Dhrystone benchmark was executed to calculate the amount of VAX DMIPS.
- **Module 6:** The objective for the last module was, to implement the bare-metal programming module for the flowmeter and start by designing the Simulink module for frequency. Proceed with the implementation of the frequency detection algorithm to identify the input frequency. After that, configure the internal temperature sensor of the ST Nucleo40RE board to provide temperature readings. Use the obtained temperature value to calculate density and viscosity, as they contribute to the volumetric flow rate. Additionally, generate the 4-20 mA and pulse outputs through PWM channels. Display the resulting flow rate data on an LCD screen, and showcase the flow, temperature, and frequency data on the UART terminal. Lastly, analyze the power consumption in both full run mode and low power mode.

Approach and methodology of evaluation

Approach:

To implement the Vortex Flowmeter, our first task was to make a block diagram with all the requirements given by Sierra Instrumentation. Then, we began evaluating the **STM32F401RE** MCU. In the beginning, we looked at the hardware and other needs. We made a table for hardware input/output evaluation, and we checked the MCU's hardware datasheet to make sure it had what the flowmeter needed. After that, we evaluate the MCU for additional requirements like budget, size, and performance.

a] Block diagram:


Vortex Flowmeter Block Diagram
Inputs
Outputs
Optional Switch
GPIO
Optional External Temperature sensor
I2C
Internal Temperature sensor
ADC
Power supply
NUCLEO-F401RE (STM32F401RE)
GPIO
LED1 Indicates System functioning
GPIO
LED2 Indicates ADC Sampling
SPI
LCD
UART
Putty PC Terminal
PWM channel-1
4-20 current loop
PWM channel-2
Vortex Frequency Output

b] Hardware evaluation:

Hardware requirements to implement the vortex flowmeter are as follows:

Input Interface		
	Requirements	**Availability on STM32F401RE MCU**
1	Vortex Frequency input (Analog) sampled at 10 kSps	ADC channels
2	2 Temperature Sensors	one on board temperature sensor and via I2c we can connect the external sensor
3	Touch Keypad	
4	12 bit ADC required at 10 kSps	has 12 bit ADC
Output Interface		
1	Pulse output (Hi-voltage Digital) for totalizer	PWM channel
2	3 LEDs (Power, flow detected)	1 on-board led and can connect externally via GPIO pins
3	LCD Display in response to touch input	Externally connected via SPI interface
4	4-20 current loop	PWM channel
5	Local Serial Port (for monitoring and data dumps)	Putty terminal via UART interface

c] Additional evaluations:

Additional evaluation is carried out for the following requirements:

1. Processing Requirement - 40 DMIPs:

In module -5 we have performed the Dhrystone benchmark and the calculated VAX DMIPS = 93716 / 1757 = 53.34. Thus, this requirement is satisfied.

2. Power - Average Power consumption must be less than 100 mW:

In Module 6 we checked the power consumption in full run mode and low power mode. So we can say that STM32F401RE MCU satisfies the power requirement.

3. Environmental: 1. Commercial Temp, 95% RH 2. Conforms to FCC EMC requirements 3. UL, CSA, and FM Certification 4. Compliant with ROHS requirements 5. 20-year Longevity (MTBF) 6. 20-year Availability

Looking at the datasheet of the MCU we can say that all the above-mentioned environmental requirements are satisfied with MCU.

4. Budget: Budget < $200 in production. This includes the PCB and connectors. Cost for the MCU <= $20.

Looking at BOM report we can say that, our implementation costs less than 200$, and STM32F401RE MCU costs < 20$.

Note: Please find the attached BOM report Excel sheet Here

So, after performing the above-mentioned hardware and additional evaluation, we concluded that the STM32F401RE MCU is the best choice for Flowmeter considering the given requirements.

Module Test Results

Module-1:

In Module 1, we successfully completed the Mbed and Nucleo board setup. Following that, we implemented the string copy and string capitalization functions in C and analyzed the memory usage through assembly and C code. Our conclusion is that the C language code consumes more memory than the assembly language code for the same functionality. Please find the technical report for map file analysis and test outputs explained on pages 2 to 6.
An explanation of the memory model of ARM Cortex-M4 with respect to the code memory, data memory, IRQ handlers, and peripherals with the help of a diagram is provided in the technical report on page 7. Next, we developed an assembly code subroutine to approximate the square root of an argument using the bisection method and tested the code with all provided inputs. The test results are documented in the Module 1 technical report on page 9.

The estimation of the number of CPU cycles used for this calculation is that 161 CPU cycles are used when approximating the square root for the input of 2. The detailed calculation and approach are explained the in module 1 technical report on page 11.

Technical report for module-1 : Here
The code development for module-1 and Doxygen report : Here

Module-2:

In Module 2, the objective was to establish the hardware interface with the Nucleo F01RE board using various communication protocols and GPIO. It was reported that the following interfacing tasks were successfully completed:
1] A button (switch) interface via GPIO pins, 2] An LCD interface through the SPI interface, 3] An external LED interface via GPIO pins, 4] An external temperature sensor interface through the I2C interface, 5]A buzzer interface via a PWM channel, 6] A Putty terminal interface via the UART interface, 7] A potentiometer interface via an ADC channel.

After the successful completion of interfacing each component, the code was merged to display the temperature sensor data on both the LCD and the Putty terminal. Additionally, the sound volume of the buzzer was adjusted based on the analog input from the potentiometer via the PWM channel. A sawtooth waveform was observed on the oscilloscope.

Wiring diagram for each interface is provided. Here

Exercise 2_8: Try to issue an interrupt on different signal edges (rising edge or falling edge). What changes?
Ans: For a rising edge interrupt, the LED state changes when I release the button. For a falling edge interrupt, the LED state changes when I press the button. (before release). When configured for a rising edge interrupt, the interrupt will be triggered when the signal transitions from low (0) to high (1). In the context of a button press, this means that the interrupt will occur when we release the button (transition from low to high). When configured for a falling edge interrupt, the interrupt will be triggered when the signal transitions from high (1) to low (0). In the context of a button press, this means that the interrupt will occur when we press the button (transition from high to low).
Exercise 2_9: What changes when you adjust the amount by which variable i is incremented/decremented?
Ans: The variable i is used to adjust the duty cycle. The amount by which i is incremented/decremented controls how fast the duty cycle reaches the target value. As a result, it affects the slope of the saw-tooth waveform. If we measure the period of the saw-tooth waveform generated, then the period value decreases as we decrease the amount by which i is changed.
Find the sawtooth waveform observed across the buzzer when connected to an RC circuit for averaging out the duty cycle. Period: 124 us, Pulse Width: 11 us and Duty Cycle: 0.089 Here
Exercise 2_11: What temperature is displayed on your PC terminal window? What is displayed on the LCD?
Ans: The temperature sensor sends temperature data via I2C, which is displayed on the PC terminal through UART and simultaneously on an LCD screen using SPI. So, the same

temperature value is displayed on the LCD and PC terminal. Find the hardware setup and test output Here
Technical report with detailed explanation for module-2 : Here
The code development for module-2 and Doxygen report: Here

Module-3:

In module 3 , we successfully implemented the CMSIS RTOS threads and convert the module-2 code in RTOS using RTX library.
Que] What temperature is displayed on the LCD?
Ans: The temperature sensor sends temperature data via I2C, which is displayed on the PC terminal through UART and simultaneously on an LCD screen using SPI. So, the same temperature value is displayed on the LCD and PC terminal. Find the hardware setup and test output Here
Technical report with detailed explanation for module-3 : Here
The code development for module-3 and Doxygen report: Here

Module-4:

In module -4 we have successfully created the start-up code using the STMCUBE configuration software and set the system clock to 84 MHz and the ADC sample clock to 100 kHz.
Configuration test output screenshots are provided Here.

Que] Compare this to the startup code you used in the other modules so far. How is it different? Provide a memory map for each.
Ans: In the STM32CubeMx generated code, the heap base is at 0x20000148 and in the audio code of Module 2, it is at 0x20000428. In CubeMx code, the heap size is mentioned as 512 bytes, and if we do the calculation for the module 2 code by subtracting the __heap_base with __heap_limit we get 0x400 which is 1024 bytes. Comparing the map files generated for the two codes, it is evident that the Module 2 code uses more code memory (around 15.6 kB), whereas the code generated by CubeMx takes only 6.2 kB. This might be because the Audio code example uses more Hal modules for PWM, serial and Analog because of which the library takes additional space. The library in Module 2 example takes up 10.3kB and in CubeMx, it is 880 bytes. However, the RAM usage in the CubeMx code is higher even with very few user variables used.
For detailed explication and required map files output please find the pages 4 to 6 in module-4 technical report. Here
Technical report with detailed explanation for module-4 : Here
The code development for module-4 Here

Module-5:

In Module 5, it was reported that improvements were made to the appearance of the debug monitor display, adding a personal touch. Additionally, commands were implemented to the monitor, enabling the display of registers R0-R15, sections of memory, and the stack.
Please find the screenshots of the debug monitor for all the tests Here .
Technical report with detailed explanation for module-5 : Here
The code development for module-5 and Doxygen report: Here

Que 1] What is the count shown in timer0 if you let it run for 30 seconds? Explain why it is this.
Ans: After running the timer0 function for 30 seconds, the count is 93E0 = 37856. Timer0 counter is a 16-bit unsigned value hence the maximum value is 65535. Each counter corresponds to 100 us. Timer0 will overflow four times within 30 seconds hence, on the 5th iteration, the counter will have a value of
(30 seconds in 100us ticks) – (100us ticks in 4 overflows of Timer0) = 300,000 – (65536 * 4) = 37856 = **0x93E0**

Que 2] How much time does the code spend in the main loop versus Interrupt Service Routines?
Ans: We did a timing analysis for 100ms. In 100 ms, the main loop runs around 26415 times, and the total time observed is 73840 us. The ISR runs around 1406 times, and the total time is 2834 us. If we take the average time for each iteration, the **main runs for around 2.8 us (73840 us / 26415),** and the **IRQ takes 2 us (2834 us / 1406).**

Que 3] Test each of the commands in the Debug Monitor and record the results. Explain anything you see that you did not expect. Are you able to display all the registers?
Ans: When switched from Quiet mode to Normal mode, we observed random values printed on the display. This might be because the tx_buffer keeps filling in the circular buffer as a result, all the characters stored in the buffer previously get printed.
Que 4] What is the new command you added to the debug menu, and what does it do? Capture a screenshot of the new monitor window.
Ans: We implemented the following three new commands:
1. REG: To display Registers R0-R15. This uses assembly instruction store multiple to copy the contents from R0 to R12 to an array.
2. STA: To display the Stack memory. Again, we use assembly instruction __ASM("sp") to access the stack memory and print out the first 16 words.
3. MEM: To display the section of memory. We are displaying 32 words in hex format from 0x20000000 to 0x2000007F.
Que 5] A GPIO pin is driven high at the beginning of the Timer ISR, and low at the end. What purpose could this serve?
Ans: Driving a GPIO pin high at the start of a Timer ISR and low at completion is a technique for precise timing analysis and debugging. It ensures that the ISR triggers at the expected time and measures its execution time. To achieve this, we can use a logic analyzer. The analyzer employs markers to capture the timing data by detecting the transitions of the GPIO pin from high to low. This allows for an accurate assessment of the ISR execution time, aiding in performance optimization and troubleshooting, ensuring that the system operates as expected.
Que 6] Estimate the % of CPU cycles used for the main background process, assuming a 100-millisecond operating cycle.
Ans: From our previous analysis in Q2, we estimated the execution times of the main and ISR. The main runs for 73840 us. The CPU runs at 84MHz, so each cycle takes around 11.9 ns.
Number of CPU cycles used by Main in 100ms = 73840 us / 11.9 ns = 6.2 million cycles.
% CPU cycles used = 6.2 / 8.4 = 73.8 %
Que 7] What is your DMIPS estimate for the ST STM32F401RE MCU?
Ans: Nucelo-STM32F401RE scores 93716 Dhrystones per second. VAX DMIPS = 93716 / 1757 = **53.34**

Module-6:

Simulink Model for Frequency Detection: We developed the above model for frequency detection. The model first tries to predict the mean amplitude value by taking the maximum and minimum amplitudes. The mean amplitude adjusts to an accurate value as more samples are obtained. After obtaining the mean amplitude, the model counts the maximum number of samples above and below the average amplitude value. These values are added together to get the sample count for a period. Since we know the sample time as 100us, we can calculate the frequency by taking the reciprocal. Find the simulation diagram Here.

Split Report: Here
Que1] What is the frequency estimate from your provided sample ADC data?
Ans: We found a frequency estimate for samples around 40 Hz using our frequency estimation algorithm. We got around 250 samples per period.
Que 2] What is the calculated flow you see from your input?
Ans: As shown in the screenshot above, we get around 1537.43 gallons/second flow.
Que 3] What is the range of temperatures you measured with your embedded system?
Ans: The minimum temperature reading is around **38.02** when the system is powered up. After running the system for some time, the temperature varies between **40.09 C and 41.13 C which gives a variation range of 1.04 C.** The total temperature variation from system start is **3.11 C**.
Que 4] How much time does the code spend in the main loop versus in Interrupt Service Routines?
Ans: Time spent in main is **7.98 us**. Time spent in ISR is **2.04 us**
Que 5] Estimate the % of CPU cycles used for the main foreground process, assuming a 100-millisecond operating cycle.
Ans: % of CPU cycles used for main =($8.269 * 10^6$ / $8.403 * 10^6$) = 98.4

The detailed calculation of all the questions is given in module-6 report Here.

Que 6] Calculate the power consumption for your complete system (including proposed hardware additions) when in full run mode and again in low power mode.
Ans: We have assumed that all the other peripherals are put to sleep and with that low power consumption is 7.92uW. When running at full speed and all the peripherals are enabled system consumes 734.74mW. If we leave the essential peripheral ADC on for taking samples and then wake up the CPU after 1000 samples, then in low power mode the system will consume 5.28mW. If we keep the LCD on always and ADC takes samples, then the system will consume 655.28mW with the CPU in deep sleep.

Please find the detailed table of power consumption Here.

List of project deliverables

The following are the project deliverables:
1] Hardware design files and documentation:

- Picture of Hardware test circuits used for modules: - These are provided in each module folder.
- Test Data and Oscilloscope Screenshots: - The terminal test data outputs for each module and oscilloscope screenshots for module 2 and module 6 are provided in their respective module folder and in technical reports.

2] Software design files and documentation:

- Simulation Software model to test the input frequency: - the .slx file is provided in the module 6 folder.
- Source code: - The Source code of each module is provided in the respective module folder.
- Doxygen reports: - The doxygen report of each module is provided in their modules folder.
- Splint report: - The Splint report for module 6 is provided in the respective module folder.

3] Detailed technical report: The technical report is written for each module and provided in the respective module folder.

Recommendations: GO/No GO

After thoroughly evaluating the STM32F401RE MCU in all the modules as detailed in the evaluation section above, it's evident that this MCU meets all the requirements set by Sierra Instrumentation for Flowmeter implementation. The MCU not only fits well within the budget and schedule but also delivers the required level of performance. With its cost-effectiveness, compact size, and ample computing power that fulfills the input requirements, it is the good choice for implementing the vortex flow meter.

References

1. Project 1 Guide by Prof. Timothy Scherr.
2. STM32F401RE Datasheet and Reference Manual.
3. ST Nucleo 401 User Guide.
4. Part of the Source codes given by Prof. Timothy Scherr for each module.

Project Staffing

Instructor:
Prof. Timothy Scherr
Teaching Associate Professor
University of Colorado Boulder
Email: Timothy.Scherr@colorado.edu

Students:

Ajay Kandagal
Grad Student
University of Colorado Boulder
Email: ajka9503@colorado.edu

Kshitija Dhondage
Grad Student
University of Colorado Boulder
Email: ksdh4214@colorado.edu