ECEN 5803-401, University of Colorado Boulder

Sierra 240 Vortex Flowmeter

By: Dayton Flores & Mark Sherman

Table of Contents

[Overview 2](#_Toc108377186)

[1. Executive Summary 2](#_Toc108377187)

[2. Problem Statement and Objectives 3](#_Toc108377188)

[3. Approach and Methodology for Evaluation 3](#_Toc108377189)

[4. Module Test Results 3](#_Toc108377190)

[5. List of Project Deliverables 5](#_Toc108377191)

[6. Recommendations 6](#_Toc108377192)

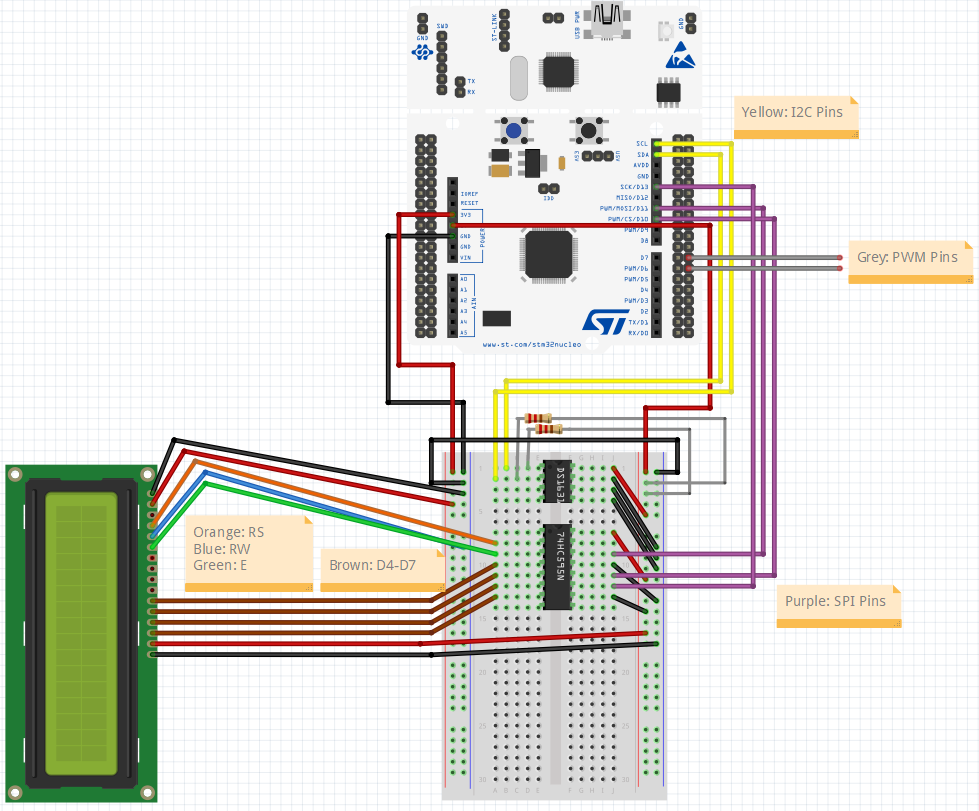
[7. Appendix A: References 6](#_Toc108377193)

[8. Appendix B: Project Team Staffing 7](#_Toc108377194)

# Overview

## Executive Summary

|  |  |
| --- | --- |
|  | The goal of this project was to assess the STM32F401RE MCU in support of Sierra Instrumentation’s proposed product development, the Sierra 240 Vortex Flowmeter, by using the Nucleo-64 board. The scope of this work includes evaluation and testing of the proposed embedded system platforms. A total of five modules were produced for testing, benchmarking, and analysis. The data used in testing was gathered in real time from sensor readings with simulated environmental noise that would closely match the product’s working environment.  Extensive research of hardware peripherals along with thorough consideration for theory of operation was taken with this project. We have examined power consumption of the prototype, capabilities and features, as well as budget. Based upon our evaluation, the recommendation for use of the STM32F401RE MCU in the Sierra 240 Vortex Flowmeter is a straight **GO**. |



## Problem Statement and Objectives

|  |  |
| --- | --- |
|  | We assessed the hardware and software capabilities of the STM32F401RE MCU using the Nucleo-64 board to determine its feasibility of being used in the Sierra 240 Vortex Flowmeter. The software environment used for development, testing, and loading was Keil uVision 5.  Required hardware peripherals (UART for monitoring and data dumps, I2C for sensors, SPI for LCD display, PWM output proportional to frequency and flow) and average power consumption of >100 mW were fulfilled. |

## Approach and Methodology for Evaluation

|  |  |
| --- | --- |
|  | The approach taken for evaluation was to produce five modules for various parts of the system. Once each module had been signed off, integration of each module was done whereafter a final system evaluation was done. |

1. **String Processing In Assembly and C** – Approximated square root with embedded C programming + in-line ARM assembly using the bisection method. Compared string copy and capitalization function memory usage between ARM assembly implementation and C implementation. Explained the memory model of the ARM Cortex-M4 with respect to code memory, data memory, IRQ handlers and peripherals.
2. **Button Read, ADC Read, LED PWM, and UART** – Tied user buttons to GPIO pins to control both on-board + external LEDs using ISRs. Read analog inputs to calculate PWM output to control pitch and volume of speaker. Read temperature sensor values via I2C and displayed the results to LCD via SPI + serial computer port via UART.
3. **RTOS Threads** – Read temperature sensor values via I2C and displayed the results to LCD via SPI + serial computer port via UART while also incorporating RT requirements with ISRs and threading with mutex locks.
4. **Debug Monitor** – Displayed values to and read user input from serial computer via UART. Operation modes included Normal with essential information, Debug with detailed information, and Quiet with no information displayed. Added new operation modes using in-line ARM assembly to display register values and data blocks in memory. Temperature sensor values read via I2C. Also ran a Dhrystone benchmark to calculate the number of VAX DMIPS.
5. **Bare Metal Flowmeter Simulation** – Implemented a cyclic executive using polling for debug monitor user input/output via UART and timer interrupts for on-board LED control. Temperature sensor values were read via I2C while calculated flow was displayed on LCD via SPI.

## Module Test Results

|  |  |
| --- | --- |
|  | Test results were taken and simulation was documented for each of the five modules. All questions were answered for each module. |

1. **String Processing In Assembly and C** (see Appendix A: References for all screenshots/pictures)
   * The memory usage for ***Code1Nucleo*** (assembly implementation) is 3.59 kB. The memory usage for ***Code1Nucleo*** (C implementation) is 3.60 kB. We can evaluate that C implementation uses 12 more bytes
   * The STM32F401RE system address map is:
     + Code memory (FLASH) block has a capacity of 512 MB. This is where all source code will be stored along with static data (constants) and the Interrupt Vector Table (IRQs)
     + Data memory (SRAM) has a capacity of 512 MB. This is the on-chip RAM where dynamic data will be stored (stack and heap)
     + Peripheral memory block has a capacity of 512 MB. This is the on-chip peripheral space
   * Results of testing square root with inputs:
     + See Appendix A: References
   * CPU cycles for each input was estimated by setting breakpoints before and after the ***my\_sqrt()*** function. Then the time difference was taken and multiplied by clock frequency.

|  |  |  |  |
| --- | --- | --- | --- |
| **Function Call** | **Time to Execute (μs)** | **Frequency (MHz)** | **CPU Cycles** |
| my\_sqrt(0) | 10.7 | 84 | 899 |
| my\_sqrt(25) | 25.0 | 84 | 2100 |
| my\_sqrt(133) | 27.6 | 84 | 2319 |
| my\_sqrt(2) | 27.3 | 84 | 2294 |
| my\_sqrt(4) | 23.0 | 84 | 1932 |
| my\_sqrt(22) | 29.3 | 84 | 2462 |
| my\_sqrt(121) | 24.9 | 84 | 2092 |

* + See ***..\Module 2\Code2\sqrt\_approx \Doxygen*** directory

1. **Button Read, ADC Read, LED PWM, and UART** (see Appendix A: References for all screenshots/pictures)
   * Wiring diagrams:
     + See Appendix A: References
   * Changing the interrupt on different button signal edges alter the point at which LEDs toggle (i.e. beginning of button push vs. letting go of button push)
   * Adjusting increment on I allows for finer tuning of each saw-tooth in the wave. For instance, the smaller the increment then the smoother the volume sounds going from minimum adjusted volume to the user’s adjusted volume
   * The temperature displayed in the terminal window was ~27 degrees C. Same result was recorded on the LCD
2. **RTOS Threads** (see Appendix A: References for all screenshots/pictures)
   * The temperature displayed on the LCD was ~23 degrees C
3. **Debug Monitor** (see Appendix A: References for all screenshots/pictures)
   * See ***..\Module 4\Code4\Doxygen*** directory
   * The count shown in timer0 when letting it run for ~30 seconds is 0x0075E703 = 7726851
   * The code spends time in the main loop versus in the ISRs
   * Yes I was able to display all 32-bits of data in all registers r0-r15. Something I did not expect before testing is upon doing a system reset, the input buffer is not displaying the keys I type. This only happens for the first selection after a system reset
   * The new commands I added to the debug menu are to display registers r0-r15 and data from a predefined memory block (0x00000000 – 0x00000100)
   * The GPIO pin driven high at the beginning of the ISR and then low at the end
   * The % of CPU cycles used for the main background process
   * DMIPS = Dhrystones Per Seconds / 1757 = 341887/1757 = 194.6 DMIPS
4. **Bare Metal Flowmeter Simulation** (see Appendix A: References for all screenshots/pictures)
   * Code5.zip downloaded and used as reference
   * Used a peak detection frequency algorithm. We subtract the current data point with the next data point and store the sign (positive or negative). Once we experience a sign change, we know we’ve hit a peak/trough. We repeat until we experience another sign change, where we know we’ve hit a trough/peak. Double the difference in the indexes and multiply by the sample period to approximate the frequency
     + The ADC data file provided by Professor Scherr gave us samples taken every 100 uS
   * Used Keil MDK for this project
   * Calculated frequency from the sample data = 1000 Hz. Calculated flow = 9974 gallons/min
   * Given a 100mS operating cycle, and the CPU operating frequency of 84 MHz, we estimate the number of CPU cycles to be 8.4 \* 10^6.
   * Power consumption in full power mode is:
     + Without LCD backlight: STM32F401RE + DS1631 + CD74HC595E + NHD\_0216HZ + 4-20 mA Transmitter + Hall Effect Sensor (Pulse) = 1.8uA(3.3V) + 1mA(5v) + 80uA(5V) + 1.5mA(3.3V) + 800uA(24V) + 6.9mA(5V) = 64.1mW
   * See ***..\Module 5\Code5\Doxygen*** directory
   * Used ***Cpplint*** on ***main.cpp*** to get “errors”. Upon review, almost all of them regard whitespace formatting suggestions. There are a few pointing out that we are using C-style casting and we should use static\_cast for readability. Other than that, no other issues being returned. Also ran ***Cpplint*** on other modified files such as ***Monitor.cpp*** and got similar results (almost all whitespace format suggestions)
     + See CPP-checker.xml found in ***..Code5/src*** directory.
   * The range of temperatures measured were 0x19 to 0x25 (25 degrees C to 37 degrees C)
     + See Appendix A: References
   * Screenshots of the product working in all operation modes (Version, Normal, Debug, Quiet, Register, Memory):
     + See Appendix A: References

## List of Project Deliverables

|  |  |
| --- | --- |
|  | We have compiled a list of product deliverables, along with the price and power consumption in the table below. Under this, we have compiled a list of the project deliverables. |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Component | Part Number | Cost | Operation Voltage | Operation Current | Power Consumed |
| MCU | STM32F401RE | $11.39 | 3.3 V | 1.8 uA | 0.00594 mW |
| Temperature Sensor | DS1631 | $1.20 | 5 V | 1 mA | 5 mW |
| Shift Register | CD74HC595E | $0.53 | 5 V | 80 uA | 0.4 mW |
| LCD Display | NHD-0216HZ-FSW-FBW-33V3C | $13.39 | 3.3 V | 1.5 mA | 4.95 mW |
| 4-20 mA Transmitter | XTR105P | $11.63 | 24 V | 800 uA | 19.2 mW |
| Hall Effect Sensor (Pulse) | A1326LUA-T | $2.09 | 5 V | 6.9 mA | 34.5 mW |
| Total | | $40.23 |  | | 64.1 mW |

|  |  |  |
| --- | --- | --- |
| **Module** | **Deliverables** | **Relative Path** |
| Module 1 | M1StringCode2 | *..\Module 1\M1String* ***..\Module 1\Code2*** |
| Module 2 | 2\_8\_1-digital-io 2\_8\_2-interrupt  2\_9-audio  2\_11-i2c  2\_11-spi  2\_11-uart  2\_11-integration | *..\Module 2\2\_8\_1-digital\_io\_uvision6\_nucleo\_f401re**..\Module 2\2\_8\_2-interrupt\_uvision6\_nucleo\_f401re**..\Module 2\2\_9-audio\_uvision6\_nucleo\_f401re**..\Module 2\2\_11-i2c\_uvision6\_nucleo\_f401re**..\Module 2\2\_11-spi\_uvision6\_nucleo\_f401re**..\Module 2\2\_11-uart\_uvision6\_nucleo\_f401re**..\Module 2\2\_11-integration\_uvision6\_nucleo\_f401re* |
| Module 3 | Code3\_3 Video of operation | *..\Module 3\Code3\_3\Code312**..\Module 3\Video* |
| Module 4 | Code4mBedDhrystone | *..\Module 4\Code4**..\Module 4\mBedDhrystone* |
| Module 5 | Code5SchematicSimulink | *..\Module 5\Code5**..\Module 5\Schematic**..\Module 5\Simulink* |

## Recommendations

|  |  |
| --- | --- |
|  | We would recommend to seek approval from Sierra to implement having the LCD’s backlight turned off during sleep mode. When the LCD’s backlight is turned on, the system will consume 113.6 mW which breaks the requirement of <100 mW. |

## Appendix A: References

|  |  |
| --- | --- |
|  | This appendix contains all detailed answers and reference screenshots related to **4. Module Test Results**. |

1. **String Processing In Assembly and C** 🡪 Double click here 🡪 ****
2. **Button Read, ADC Read, LED PWM, and UART** 🡪 Double click here 🡪 ****
3. **RTOS Threads** 🡪 Double click here 🡪 ****
4. **Debug Monitor** 🡪 Double click here 🡪 
5. **Bare Metal Flowmeter Simulation** 🡪 Double click here 🡪 

## Appendix B: Project Team Staffing

|  |  |
| --- | --- |
|  | We approve the project as described above, and authorize the team to proceed. |

|  |  |  |
| --- | --- | --- |
| Name | Title | Date |
| Dayton Flores | Graduate Student at University of Colorado Boulder | July 9, 2022 |
| Mark Sherman | Graduate Student at University of Colorado Boulder | July 9, 2022 |