ECEN 5803 Mastering Embedded System Architecture

Project 1
"Vortex Flow Meter"

Authors
Kiran Jojare
Viraj Patel

Professor
Timothy Scherr

Table of Contents

1. Executive Summary

The Vortex Flow Meter, anchored on the STM32F401RE, underwent a comprehensive six-phase evaluation of its capabilities. Initially, assembly functions, particularly those related to string operations, were converted into C code, followed by a meticulous analysis of the memory model of the arm cortex M4. As the evaluation progressed, hardware functionalities such as GPIO, ADC, PWM, UART, and I2C were explored. Leveraging the mbed API, FreeRTOS was seamlessly integrated for concurrent task execution, along with SPI interfaced LCD. Later, the STM Cube MX was employed for auto-generation of application-specific code, which was then compared against the Keil start-up code. Furthermore, a Debug Monitor was realized, providing functionalities like register, stack, and memory dumps. The final phase delved into a simulation environment, utilizing Simulink for frequency detection algorithms and examining the STM32F401RE's proficiency in frequency and temperature-based flow meter calculations.

Throughout the evaluation, numerous tests underscored the STM32F401RE's adaptability and efficiency. Initial assessments emphasized that both Keil and mbed studio were suitable for STM32F401RE software development, highlighting only minor size variations between assembly and C code. A detailed examination of its capabilities revealed that applications using GPIO, ADC, UART, and PWM were notably robust, especially when paired with the mbed API. The introduction of RTOS emphasized the importance of static memory allocation for individual tasks due to the memory constraints of the device. The utility of STM Cube manifested as a streamlined method for initializing custom hardware components, revealing no differences in start-up codes compared to Keil. The integration of timer 0 with the debug monitor showcased the capability for simultaneous task management, registering a performance rate of 110 DMIPS. Simulink simulations for frequency detection proved consistent, even in varied noise environments. Furthermore, the seamless integration of bare metal code into an existing mbed application was noted, and a CPP checker was applied for the static code analysis of the project files.

Given the evaluations, it's evident that the STM32F401RE offers substantial promise for Vortex Flow Meter applications. For organizations aiming for efficient, robust, and adaptable solutions, this platform presents a compelling choice. Recommendations include delving deeper into optimizing memory allocations, as the static approach was crucial in the evaluation. Utilizing STM Cube for rapid customizations is also recommended. In conclusion, the STM32F401RE, with its diverse capabilities and impressive performance metrics, stands as a prime candidate for future Vortex Flow Meter deployments.

2. Problem Statement and Objectives

In fluid dynamics, specifically within the realm of the Vortex Flowmeter, the STM32F401RE microcontroller's capabilities need thorough exploration. The primary objective is to rigorously evaluate its hardware and software dimensions, entailing a series of tests to ascertain its aptitude for Vortex.

Flowmeter calculations. The software's performance will be assessed across multiple programming environments, with a keen focus on simulating flow measurements using the prototype and ensuring seamless interfacing with critical peripherals and sensors. A pivotal part of this assessment also involves understanding its power dynamics in different operational settings. The culmination of this exploration aims to provide a definitive recommendation regarding the STM32F401RE's viability in such applications.

3. Approach and Methodology for Evaluation

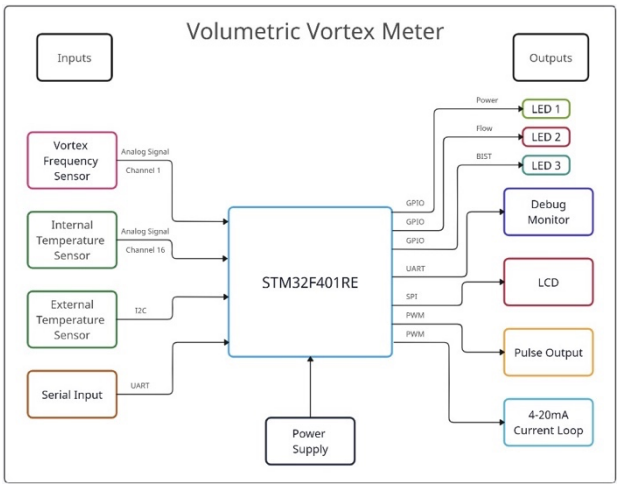


Figure 1 : Block Diagram Of Vortex Flowmeter

3.1. Module 1 : To C or not to C

The team began with the "M1String" Keil project, contrasting the memory utilization between functions written in C and assembly. By leveraging the memory model of the ARM Cortex-M4, the primary task was to approximate square roots via the bisection method in assembly, all executed on the STM32F401RE MCU. Tests were conducted rigorously, with an additional exploration into fixed-point representation using the Q16.16 format. Comprehensive documentation was generated using Doxygen.

3.2. Module 2 : Button Read, ADC read, LED PWM & UART

The team undertook a methodical approach, leveraging the mbed API to seamlessly interface with the designated hardware kit. Emphasis was placed on the development of precise wiring diagrams for each sub-exercise to ensure accurate hardware interactions. Systematic tests were then executed, which involved adjusting signal edges and tweaking variable increments to ascertain their effects. Observations were carefully noted from the readings displayed on both the PC terminals and LCD, allowing for a robust evaluation of the solution’s performance and reliability.

3.3. Module 3 : RTOS Threads

In this module, a systematic approach was adopted to realize a multitasking environment using RTOS threads. Key breadboard connections were established, incorporating additional LED and Potentiometer components. This module demonstrated the power of concurrent task execution, evident from the simultaneous display of counts and temperature on an LCD, as well as the real-time brightness adjustment of the display using a potentiometer. Evaluations from the serial terminal output validated the successful integration and functioning of the tasks. The observed temperature reading from the breadboard setup was 22 degrees Celsius. Throughout the exercise, the team relied heavily on the mbed RTOS to facilitate and manage the multitasking environment.

3.4. Module 4 : Autoconfigure with STM Cube MX

The module cantered on the use of the STMCUBE configuration software for the generation of start-up code. Key parameters were meticulously set, with the system clock at 84 MHz and the ADC sample clock at 100 kHz. An in-depth comparison between the start-up code from STMCUBE and that from Keil modules was conducted. Surprisingly, the evaluation revealed that the start-up codes were identical, challenging the assumption of potential differences arising from varied sources. The creation of a detailed memory map for each version underscored this uniformity. Leveraging the STMCUBE software showcased its efficiency, but it also emphasized the consistent nature of start-up codes across different platforms.

3.5. Module 5 : Debug Monitor

In Module 5, participants construct a debug monitor using the Keil development environment based on an mbed project. The setup integrates a timer with 100us intervals, GPIO controls, and essential source files to craft an enhanced debug monitor display with additional features. Participants proceed to benchmark using Dhrystone 2.1 , with a notable achievement of 110 DMIPS in the Dhrystone test, better than datasheet specifications of 105 DMIPS. Documentation is generated through Doxygen. The evaluation focuses on the nuances of Timer0's operation, the runtime comparison between the main loop and Interrupt Service Routines, and the efficiency of Debug Monitor commands. Further scrutiny assesses the utility of newly added commands and the strategic use of a GPIO pin within the Timer ISR, all set against the backdrop of assessing the STM32F401RE MCU's performance.

3.6. Module 6 : Bare-Metal Flow Meter Simulation

In Module 6, participants delve into the simulation and design of a bare metal flowmeter system using MATLAB's Simulink. They leverage the physics of water flow, especially the Strouhal number, to derive formulas for fluid velocity and volumetric flow rate. After testing their algorithm amidst noisy signals in Simulink, they transpose this logic to C code for a microcontroller, integrating components like ADC, PWM outputs, and SPI port LCD displays. Additionally, they focus on power optimization, using low-power bare-metal ADC modes, and ensure robust software through documentation and code quality checks, resulting in an efficient simulated flow meter system.

4. Module Test Results

4.1. Module 1 : To C or not to C

For Project 1 Module 1, the assembly code demonstrated a slight memory efficiency over the C code by using 16 bytes less in the code section. Testing of the approximate square root function yielded results in the R0 register for given inputs, both with and without Q16.16 format code. A detailed breakdown of memory usage, as well as the square root function results, can be found in the appendix along with relevant screenshots.

4.2. Module 2 : Button Read, ADC read, LED PWM & UART

The GPIO was programmed using the mbed API. For the given setup, the breadboard connections incorporated an additional switch due to the kit's limitation. The system's functionality is outlined as follows: one button activates the internal LED LD2, another deactivates it, while the other two control an external LED's operation. Module results are included in appendix.

4.3. Module 3 : RTOS Threads

In Module 3, the team utilized the mbed FreeRTOS thread APIs to concurrently execute several tasks. These tasks encompassed displaying the temperature and a counter on an LCD using SPI, presenting temperature brightness and a counter on a Serial terminal via UART, modulating LED brightness via PWM and a potentiometer using ADC, and blinking an LED. Comprehensive details and results of these tasks are provided in the appendix.

4.4. Module 4 : Autoconfigure with STM Cube MX

In Module 4, the team achieved several tasks. Firstly, the system clock was configured to 84 MHz Secondly, the ADC clock configuration was adjusted using a prescaler, setting it to 97.2 kHz, ensuring it stayed within the target frequency range of 100kHz with a margin of ±5%. An intriguing observation was made during the comparison of the start-up code between Keil and STMCube MX, where both were found to be identical. Detailed screenshots supporting these findings can be found in the appendix.

4.5. Module 5 : Debug Monitor

In Module 5, the team adeptly executed the Debug Monitor implementation, meeting a spectrum of crucial requirements. This entailed the activation of Timer0 for accurate timing measures, smooth integration of pivotal mbed elements, and the successful import of essential source code files. The Debug Monitor was brought to life, introducing user-centric commands such as Normal, Debug, Version, and Quite. Moreover, it boasted advanced functionalities through new commands for register, memory, and stack dumps. Further refinements included aesthetic enhancements and the introduction of flags for LED control. Benchmarking was accomplished and came out to be 110 DMIPS, and a thorough documentation was curated using Doxygen. For a more detailed view, the appendix contains pertinent screenshots.

4.6. Module 6 : Bare-Metal Flow Meter Simulation

In Module 6, a Simulink-based frequency detection algorithm was developed to precisely calculate flow. Tested against an AWGN channel with a 20dB SNR, the model proved consistent in frequency detection amidst noise. This algorithm was then implemented in C, meshing seamlessly with the monitor program of Module 5. It displayed flow, temperature, and frequency data over UART and an LCD. Additionally, an bare-metal ADC channel was set up for internal temperature readings, and two PWM channels were introduced: one for an external LED and another for a 4-20 mA current loop. Further specifics and visuals are available in the appendix.

5. List of Project Deliverables

The submitted ZIP file encompasses essential project deliverables such as module-specific project files, comprehensive Doxygen documentation, detailed test results, individual module reports, and a structured block diagram. For a more interactive experience and broader access, all these details are also hosted on the GitHub repository.

6. Recommendations

When designing Sierra Instrumentation's Vortex Flowmeter, it's essential to account for factors such as speed, cost, power efficiency, and overall budget. Based on comprehensive evaluations, the STM32F401RE emerges as the preferred microcontroller for this endeavour. Not only does it offer

compatibility with prominent open-source software like the mbed compiler, but it also boasts seamless debugging experiences through platforms like Keil uVision. Extensive documentation further simplifies the development process. Importantly, the Bill of Materials (BOM) for the STM32F401RE totals $49.648, excluding the PCB. This figure comfortably fits within the project's budgetary constraint of $200. Given its robust features and its alignment with budget requirements, it's recommended to proceed with the STM32F401RE for Sierra's Vortex Flowmeter project. Hence, a **'GO'** decision can be affirmed for its utilization

7. Appendix : References

[1]. Project1Guide.pdf
[2]. ST STM32F401RE Datasheet.
[3]. ST STM32F401RE Reference Manual.
[4]. ST Nucleo 401 Product Brief
[5]. ST Nucleo 401 User's Guide.

8. Appendix : Test Results

8.1. Module 1

- **Memory Usage for Assembly Code :**

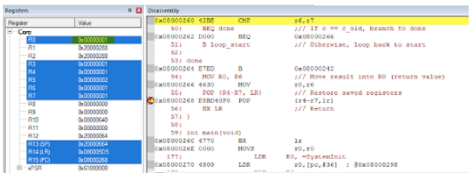
Section	Size (Bytes)
Code	1392
RO-data	436
RW-data	40
ZI-data	1632

- **Memory Usage for C Code :**

Section	Size (Bytes)
Code	1408
RO-data	436
RW-data	40
ZI-data	1632

Figure 2 : Memory Usage of C and Assembly Code

- Testing input: 2
  Result as seen in R0 register = 0x00000001



- Testing input: 2.0
  Result as seen in R0 register = 0x00016A00(1.41 in Q16.16 format)

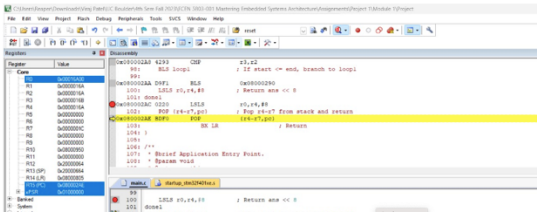


Figure 3 : Square root using integer and fixed point for input 2

8.2. Module 2

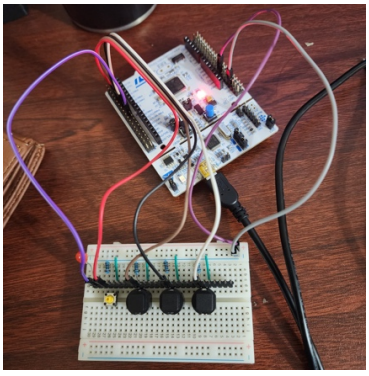


Figure 4 : Hardware Connections for Module 2

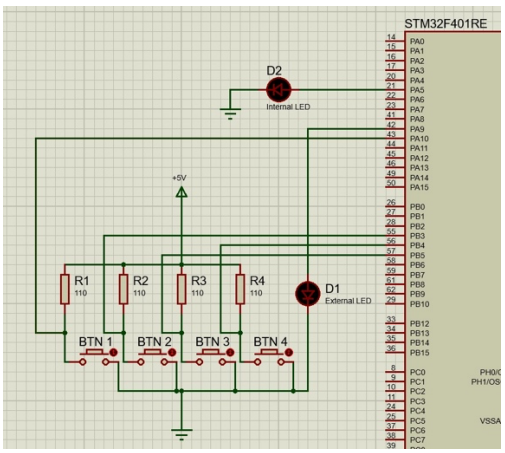


Figure 5 : Module 2 Schematic

8.3. Module 3

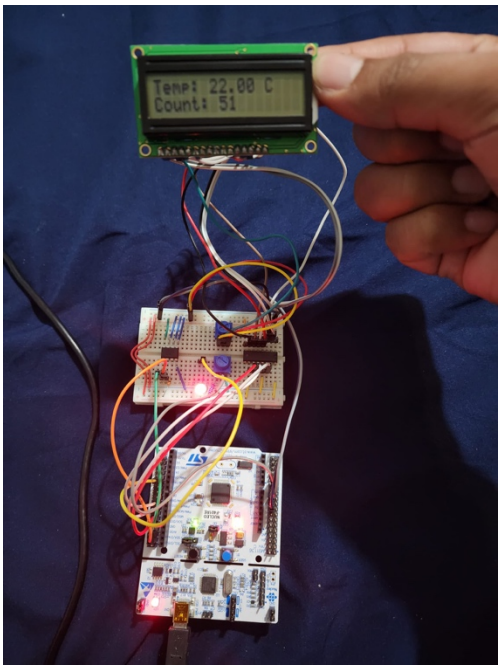


Figure 6 : Hardware Connection for Module 3

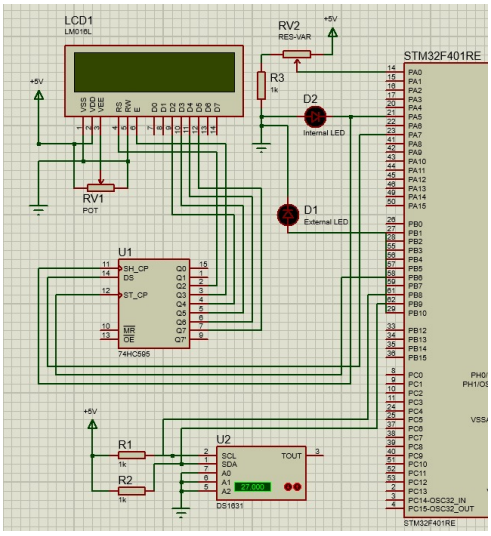


Figure 7 : Module 3 Schematic

```
8. COM4 (STMicroelectronics STLink
Temp: 24.06 C
Brightness: 0.75
Count: 98
Brightness: 0.61
Count: 99
Temp: 24.19 C
Brightness: 0.62
Count: 100
Temp: 24.19 C
Brightness: 0.54
Count: 101
Brightness: 0.54
Count: 102
Temp: 23.94 C
Brightness: 0.54
Count: 103
```

Figure 8 : Serial Terminal Output for Module 3

8.4 : Module 4

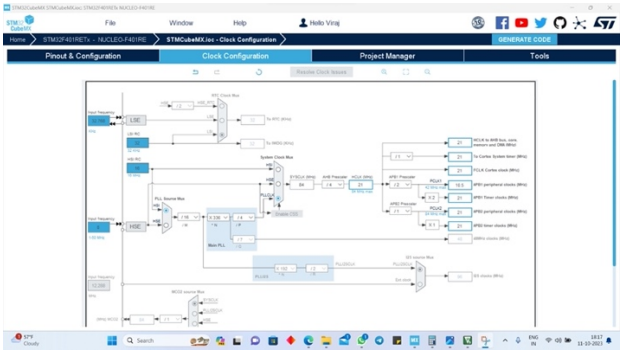


Figure 9 : SYSCLOCK Configuration of 84 MHz

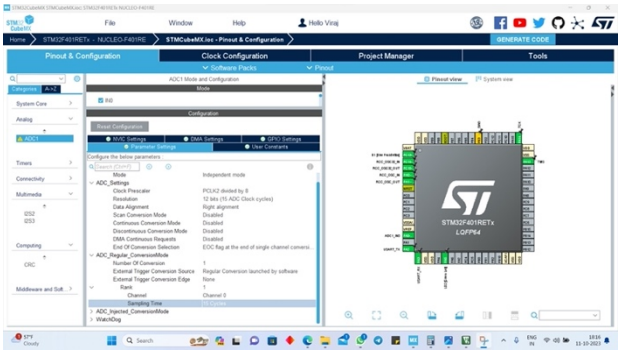


Figure 10 : ADC Configuration for 100 kHz

8.5 : Module 5

```
13. COM3 (STMicroelectronics STLin
Hello World!


***** Project 1 Module 5 - Viraj_Kiran *****


System Reset
Code ver. 2.0 2016/09/29
Copyright (c) University of Colorado

+------------------------+
|  Select Mode           |
|------------------------|
|  NOR - Normal          |
|  QUI - Quiet           |
|  DEB - Debug           |
|  REG - Registers Dump  |
|  MEM - Memory Dump     |
|  STK - Stack Dump      |
|  V   - Version#        |
+------------------------+
Select:
```

Figure 11 : Debug Monitor Entry Commands

```
Select:  V->
2.0 2016/09/29
Select:  NOR->
Mode=NORMAL

NORMAL  Flow:  Temp:  Freq:
NORMAL  Flow:  Temp:  Freq:
NORMAL  Flow:  Temp:  Freq:
NORMAL  Flow:  Temp:  Freq: D
NORMAL  Flow:  Temp:  Freq: EB->
Mode=DEBUG

DEBUG  Flow:  Temp:  Freq:
DEBUG  Flow:  Temp:  Freq:
DEBUG  Flow:  Temp:  Freq:
DEBUG  Flow:  Temp:  Freq: QUI->
Mode=QUIET
```

Figure 12 : Debug monitor - Version Info, Normal and Quite

```
Select:  NOR->
Mode=NORMAL

NORMAL  Flow:  Temp:  Freq:
NORMAL  Flow:  Temp:  Freq:
NORMAL  Flow:  Temp:  Freq: R
NORMAL  Flow:  Temp:  Freq: E
NORMAL  Flow:  Temp:  Freq: G->
Mode=Registers Dump

Printing Register Values
R0: 0x0000001A
R1: 0x0000000A
R2: 0x2000038C
R3: 0x0800150A
R4: 0x0800150A
R5: 0x08000BD5
R6: 0x00000064
R7: 0x0415DCE1
R8: 0x00000000
R9: 0x00000000
R10: 0x00000000
R11: 0x00000000
R12: 0x40004400
SP: 0x20017FF0
LR: 0x08001225
PC: 0x080001F8
```

Figure 13 : Debug monitor - Normal & Register Dump

```
MEM->
Mode=Memory Dump

Dumping memory at Location 0X20010000 64 Bytes:
0x20010000: C0 DC 05 46 CD DA FB 71 CF B1 9A 1C 67 FE F6 15  ...F...q....g...
0x20010010: B7 10 D5 05 55 0C 6F C5 0D F6 3F BD 94 81 0A 55  ....U.o...?....U
0x20010020: 3F 88 2A 3D 99 8C DD 81 0A DD F5 71 55 80 A9 40  ?.*=.......qU..@
0x20010030: B2 FE D9 04 8F B3 45 60 74 BB EF 4D 91 B4 EE 45  ......E`t..M...E
STK->
Mode=Stack Dump

Print Stack
Current Stack Pointer SP: 0x20017FE8
Word 16: 0x20017FE8 : 0800150A
Word 15: 0x20017FE4 : 20017FE8
Word 14: 0x20017FE0 : 0000000F
Word 13: 0x20017FDC : 0800153C
Word 12: 0x20017FD8 : 200002D8
Word 11: 0x20017FD4 : 0800138F
Word 10: 0x20017FD0 : 00000064
Word 9: 0x20017FCC : 20017FE8
Word 8: 0x20017FC8 : 00000007
Word 7: 0x20017FC4 : 080034B1
Word 6: 0x20017FC0 : 00000000
Word 5: 0x20017FBC : 00000000
Word 4: 0x20017FB8 : 00000000
Word 3: 0x20017FB4 : 00000000
Word 2: 0x20017FB0 : 0439301F
Word 1: 0x20017FAC : 00000064
```

Figure 14: Debug Monitor - Memory & Stack Dump

8.6 : Module 6

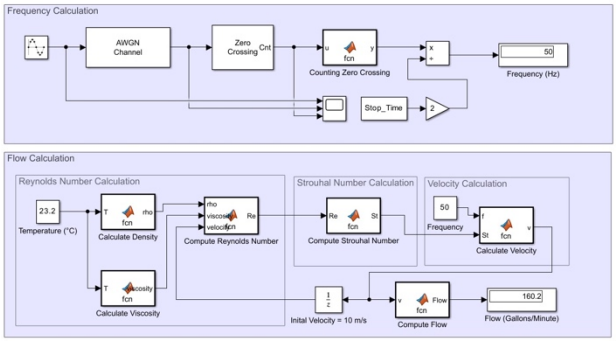


Figure 15 : Simulink Model for Frequency and Flow Calculation

```
Select:  NOR->
Mode=NORMAL

NORMAL  Flow: 160.16 Temp: 21.42 Freq: 50.00
NORMAL  Flow: 160.16 Temp: 21.75 Freq: 50.00
NORMAL  Flow: 160.16 Temp: 21.42 Freq: 50.00
NORMAL  Flow: 160.17 Temp: 21.10 Freq: 50.00 DEB->
Mode=DEBUG

DEBUG  Flow: 160.17 Temp: 21.10 Freq: 50.00
DEBUG  Flow: 160.16 Temp: 21.75 Freq: 50.00
DEBUG  Flow: 160.17 Temp: 21.10 Freq: 50.00
DEBUG  Flow: 160.17 Temp: 21.10 Freq: 50.00
DEBUG  Flow: 160.16 Temp: 21.42 Freq: 50.00
```

Figure 16 : Monitor Code Displaying Flow, Temperature and Frequency

9. Appendix : Frequently Asked Questions

9.1. Module 1

9.1.1. Testing Approximate square root with bisection method without Q16.16 format code with these inputs: 2, 4, 22, and 121.

Results seen in R0 as 0x00000001(2), 0x00000002(4), 0x00000004(22) and 0x0000000B(121) respectively.

9.1.2. Estimate the number of CPU cycles used for this calculation

After going through the code computing CPU cycles for each instruction the final CPU cycle time can be computed as follows. Assuming that branching is not taken and I being number of iterations.

Total Cycles = 11 (Setup) + [8 + (2 (for either Update A or B)) + 5]*i (i Iterations) + 8 (Clean-up).

9.1.3. Auto-generate documentation using Doxygen. Provide either an HTML directory or PDF file documenting your codebase.

Autogenerated documentation available at Module1/Doxygen Documentation.

9.1.4. Testing Approximate square root with bisection method with Q16.16 format code with these inputs: 2.0, 4.0, 22.0, and 121.0.

Results seen in R0 as 0x00016A00(1.41), 0x00020000(2.0), 0x0004B000(4.68) and 0x000B0000(11.0) respectively.

9.1.5. Estimate the number of CPU cycles used for this calculation and also the size of the code in memory.

After going through the code computing CPU cycles for each instruction the final CPU cycle time can be computed as follows. Assuming that branching is not taken and I being number of iterations.

Total Cycles = 12 (Setup) + [9 + 1 (cond_less_equal if branched) + 3]*i (i Iterations) + 6 (Clean-up)

9.2. Module 2:

9.2.1. Try to issue an interrupt on different signal edges (rising edge or falling edge). What Changes?

We configured interrupts for both rising and falling edges using the mbed API. However, there were no discernible differences in LED behaviour or response times between the two interrupt configurations to naked eye.

9.2.2. What changes when you adjust the amount by which variable i is incremented/decremented?

Adjusting the increment/decrement value of variable "i" alters waveform density: increasing makes it rougher, decreasing makes it smoother, affecting auditory perception. Our ability to discern changes relies on waveform resolution and human auditory perception.

9.2.3. What temperature is displayed on your PC terminal window? What is displayed on the LCD ?

The temperature displayed on UART & LCD was 27 Celsius.

9.3 : Module 3

9.3.1 : What temperature is displayed on the LCD?

The temperature displayed on LCD was 22 degrees Celsius.

9.4. Module 5

9.4.1. What is the count shown in timer0 if you let it run for 30 seconds? Explain why it is this?

The count shown in timer 0 is 37856. The variable timer0_count is constructed with 16 bits. Every 6.5 seconds, this 16-bit variable undergoes a reset due to the reloading of its count value. Within 30 seconds, this resent occurs 4 times (6.5 multiplied by 4 equals 26 seconds). In the leftover 4 seconds, it increases to a count of 37856.

9.4.2. How much-me does the code spend in the main loop versus in Interrupt Service Routines?

Time take by Main in seconds : 0.6468 sec.

Time taken by ISR in seconds : 2 usec.

9.4.3. 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?

We tested all Debug Monitor commands, documented results with screenshots in the Debug Monitor Serial Terminal section, and noted unexpected newline characters during initial version display. Additionally, we successfully displayed all registers, including R0 to R13, PC, SP, and LR.

9.4.4. What is the new command you added to the debug menu, and what does it do? Capture a screenshot of the new monitor window

Three new commands named “Register Dump”, “Memory Dump” and “Stack Dump” has been added in debug monitor code

9.4.5. Estimate the % of CPU cycles used for the main background process, assuming a 100 millisecond operating cycle.

Percentage of CPU cycles used for the main background process = (Time taken by Main / Total time) * 100 = (0.6468/0.6488) * 100 = 99.69%

9.4.6. What is your DMIPS estimate for the ST STM32F401RE MCU?

Each run yielded a DMIPS value of 110.352674 and DMIPS/MHz measure is seen as 5.9650 DMIPS/MHz

9.5. Module 6

9.5.1. What is the frequency estimate from your provided sample ADC data?

For the provided sample ADC data we obtained a frequency of 50 Hz.

9.5.2. What is the calculated flow you see from your input?

For the computed frequency as 50 Hz and room temperature observed at 21.75 Celsius, we obtained the flow of 160.17 gallons per minute

9.5.3. What is the range of temperatures you measured with your embedded system?

We measured the temperatures in the range of 20 to 29 degree Celsius.

9.5.4. How much time does the code spend in the main loop versus in Interrupt Service Routines?

Time take by Main in seconds : 0.6468 sec.
Time taken by ISR in seconds : 2 usec.

9.5.5. Estimate the % of CPU cycles used for the main foreground process, assuming a 100 millisecond operating cycle.

Percentage of CPU cycles used for the main background process = (Time taken by Main / Total time)
* 100 = (0.6468/0.6488) * 100 = 99.69%.

9.5.6. Calculate the power consumption for your complete system (including proposed hardware additions) when in full run mode, and again in low power mode.

- Maximum Power Consumption STM32F401RE: 3.3V * 8.2mA = 27.06mW
- Maximum Power Consumption(LCD): 3.3V * 2.5mA = 8.25mW
- Power Consumption (Shift Register) - SN74HC595 At 3.3V Current Value - 0.066mW

Total Maximum Power Consumption (Full Run Mode): 27.06mW + 8.25mW + 0.066mW = 35.376mW

Total Power Consumption in Low Power ADC Mode: 3.3V * 8.19702mA= 27.05mW (Slightly Greater Than This)

10. Appendix : Bill Of Material (BOM)

The Bill Of Material (BOM) called BOM.xlsx is available in submitted folder at Final Report.

Item No.	Description	Quantity	Unit Cost ($)	Total Cost ($)	Vendor	Vendor Part Number
1	Microcontroller	1	7.628	7.628	STMicroelectronics	STM32F401RET6
2	LCD	1	13	13	Newhaven Display Intl	NHD-0216HZ-FSW-FBW-33V3C
3	Shift Register	1	0.33	0.33	Texas Instruments	SN74HC595
4	LED	3	0.34	1.02	DigiKey	1497-XCMDK12D-ND
5	Temperature Sensor	1	8.58	8.58	DigiKey	DS1631-ND
6	Pin Headers	4	0.26	1.04	DigiKey	2057-PH1-15-UA-ND
7	Wire	2	4.9	9.8	DigiKey	BKWK-3-ND
8	Jumpers	3	2.75	8.25	DigiKey	PRT-14284
			Total = 49.648			

Figure 16 : Bill Of Material (BOM)

11. Appendix : Project Staffing

Kiran Jojare
Graduate Student
University of Colorado, Boulder.
Email : kijo7257@colorado.edu
Phone : +1 (720) 645 – 6212

Viraj Patel
Graduate Student
University of Colorado, Boulder.
Email : vipa5773@colorado.edu
Phone : +1 (720) 561 – 1864