Project 1 Module 6 Flow Meter

In this project, a robust simulation frequency detection algorithm was developed using Simulink. Our primary objective was to accurately calculate flow based on the detected frequency. To ascertain the effectiveness and reliability of our model, we incorporated an AWGN channel into our Simulink simulation, with a Signal-to-Noise Ratio (SNR) of 20dB. Through rigorous testing using white noise, we confirmed that the algorithm consistently yielded identical frequency calculations both with and without noise interference.

Transitioning from simulation to real-world application, the frequency detection algorithm was implemented in C. This C code integrated seamlessly with the monitoring program from Project 1, Module 5. It facilitated the display of flow, temperature, and frequency data over UART, while concurrently presenting flow readings on an LCD. Additionally, an ADC channel was set up to fetch internal temperature readings using bare metal C code. The project further showcased the versatility of PWM by establishing two distinct PWM channels; one drove an external LED, and the other controlled a 4-20 mA current loop, exemplifying the breadth of applications possible with our design.

Simulation Model Analysis of Frequency Detection and Flow Calculation Using Simulink

The primary goal of the model is to calculate frequency and subsequently compute flow in gallons per meter. The study was systematically divided into two parts, each delving into separate aspects of the simulation. Figure(1) shows the Simulink model implemented.

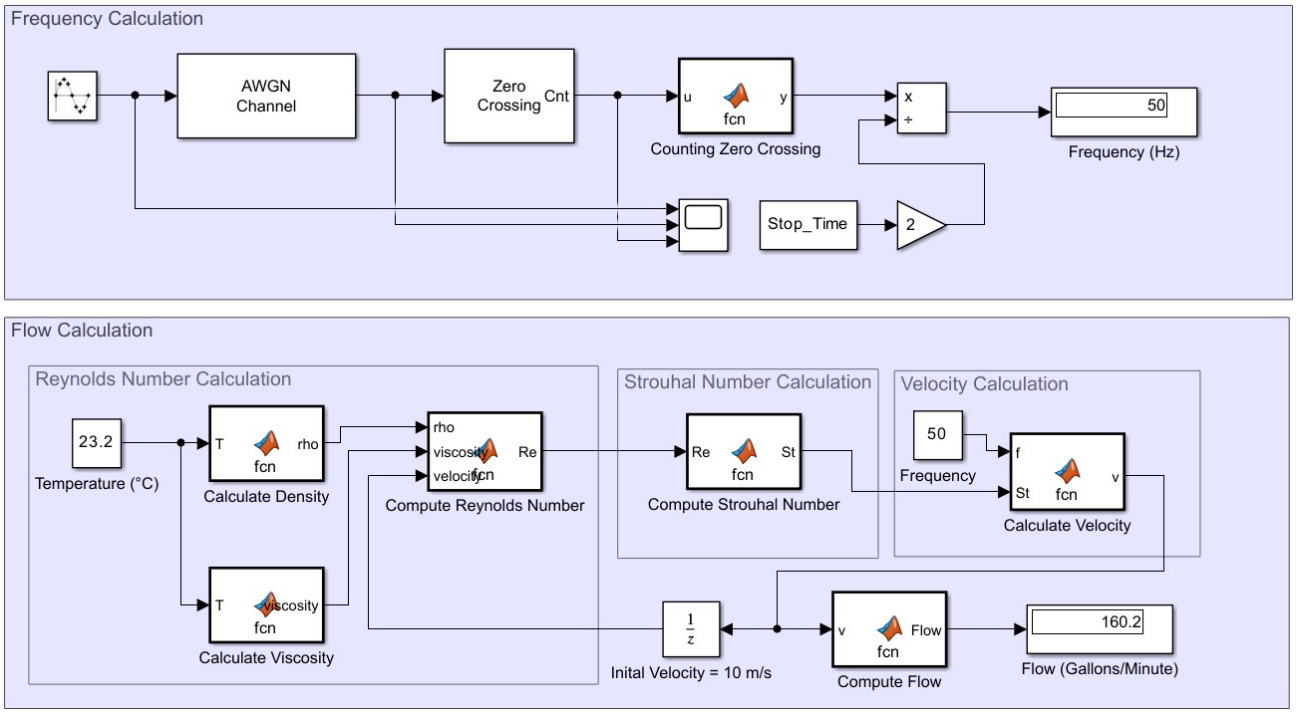

Figure 1 : Simulink Model for Frequency and Flow Calculations

Part 1: Frequency Detection Algorithm :

The frequency detection algorithm revolves around the sine wave generator block and Zero Crossing Detector Block in Simulink. Sine wave block, working on a sample-based methodology. A dedicated MATLAB script was employed to establish key parameters required for sine wave generator block which are sample time and samples per period.

Sampling and Frequency Parameters:

The MATLAB script requires two user inputs: one for frequency in Hz and another for stop time in seconds. The user-specified frequency is then converted into radians per second for subsequent computations. To achieve enhanced resolution, the samples per period are consistently set at 1024, a minimum of 256 is necessary. The sampling rate, on the other hand, is dynamically determined as follows.

$$Sample\ Time = \frac{2 * \pi}{Frequency * Samples\ Per\ Period}$$

Zero Crossing and Frequency Calculation:

Once the sine wave is generated, it is directed to a zero-crossing detector block. This counts the number of times the signal crosses zero. By using the stop time and the counts obtained from the zero-crossing detection, our model computes the final frequency. After the simulation runs until the defined stop time, a display block showcases the derived frequency as seen in figure(1).

Part 2: Flow Calculation

Post frequency determination, the model embarks on calculating the flow, measured in gallons per meter. The first calculation iteration uses a initial velocity of 10 m/s, as prescribed by TA Amey, using a unit delay block.

To validate the model's robustness, it was subjected to white-noise conditions using an AWGN channel block. AWGN Channel is using 20 dB Signal-to-Noise ratio (SNR).By simulating frequencies from 10 Hz to 2000 Hz, with and without noise, the model consistently demonstrated its resilience. The output for a 50 Hz frequency and temperature set to 23.2 Celsius corresponds to flow computed as 160.2 Gallons per minute, can be found in Figure (1).

"Code Section" in this report lists all MATLAB functions and scripts used for the calculation of flow and frequency used in the Simulink model.

Simulation of Frequency and Flow Calculation Using C Code

Part 1: Utilization of ADC Samples for Frequency Calculation

In the initial phase of this section, we utilized ADC samples that was provided by our professor in a TXT file format. For our computational purposes, we established a statically allocated array named '*unsigned short ADC_SAMPLES[1000]*'. This array was designed to mirror the exact values represented in the TXT file we received using a python script. With this data in place, we calculated the frequency.

Taking guidance from the professor's email, the sampling time for the ADC samples provided was 100 microseconds as mentioned by professor. An intriguing observation was made during analysis of data given; we found that the zero crossing corresponded to a hexadecimal value of 0x7FFF in the samples we received. An integral part of this process was to check the ADC samples for zero crossing. It was through these counts that we were able to derive the frequency using formula below. A single sinewave has 2 zero crossing per one period as also seen in the formula.

$$Frequency = \frac{Zero\ Crossing\ Counts}{No\ of\ Samples * Sampling\ Time(ADC) * 2}$$

Part 2: Temperature Reading with Custom ADC Configuration

For the second segment, we engaged the ADC to decipher the temperature. This was accomplished by utilizing the internal temperature sensor available on the STM32F401RE. As per the assignment guidelines, we used AnalogIn mbed API to use internal temperature sensor configured at ADC 1 Channel 16. Temperature sensor calculation also requires us to set VREFINT configured at ADC 1 Channel 17. The ADC 1 Channel 1, has been assigned to a virtual vortex frequency input.

ADC configuration was changed using a bare metal C code as requested in the assignment. Configuration selected us regular conversion, continuous conversion mode, scan mode, no analog watch dog, resolution 12 bit and other options are kept at default. Sampling rate is also set to 100kSps. This 100kSps translates to 100k samples per seconds which is 10 usec. Now based on this and using formula from data sheet in the section 11.5 Channel-wise programmable sampling time we computed as follows.

11.5 Channel-wise programmable sampling time

The ADC samples the input voltage for a number of ADCCLK cycles that can be modified using the SMP[2:0] bits in the ADC_SMPR1 and ADC_SMPR2 registers. Each channel can be sampled with a different sampling time.

The total conversion time is calculated as follows:

T_{conv} = Sampling time + 12 cycles

Example:

With ADCCLK = 30 MHz and sampling time = 3 cycles:

T_{conv} = 3 + 12 = 15 cycles = 0.5 µs with APB2 at 60 MHz

Figure 2 : Sampling Time Calculation for ADC

The internal clock for the ADC, denoted as APB2CLK, is pre-set to 16MHz by default. Referring to the formula presented in Figure(2), we determined that the SMPR1 register should be populated with the value SMPx[2:0]: 110 in bits, equivalent to a 144-cycle duration. When combined with the 12 cycles allocated for resolution, the total comes to 9.75 microseconds. This aligns closely with our target of 10 microseconds, which translates to a sampling rate of 100kSps.

We cross-referenced the readings from the internal temperature sensor with the ambient room temperature measurements. The room's temperature was recorded at 22.6°C, while the sensor reported a value of 21.75°C. This discrepancy is well within the +/-1.5°C margin mentioned in the data sheet, indicating the sensor's accuracy.

Part 3: Flow Calculation in Gallons Per Minute

In this step, we wanted to find out the flow rate in gallons per minute. We used the frequency from Part 1 and the temperature from Part 2 to do this. When we ran the C code with the samples given by our professor, we found the flow rate to be 160.17 gallons per minute. This was with a 50 Hz frequency and a room temperature of 21.75°C. This result matched with what we saw in our Simulink Simulation.

Part 4: Pulse Width Modulation (PWM)

We chose PC8 and PC9 for the PWM pulse output. PC8 is set to give a PWM output that is proportional to the flow rate in gallons per second, with a 50% duty cycle, which is 12 mA. Similarly, PC9 is set to give an output that proportional to the frequency in Hz, also with a 50% duty cycle , which is 12 mA.

Part 5: Integration with Monitor Code for Data Display

We added the last part of our project by connecting it with the Monitor Code from Project 1, Module 5. This let us send data in two ways: using UART with the monitor code and showing it on an LCD screen using SPI. When we enter either "NOR" or "DEB", we can see the temperature, frequency, and flow data on UART. The same flow data is also displayed on the LCD, as shown in figures(3) and (4).

```
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 3 : Monitor Code Displaying Flow, Temperature and Frequency

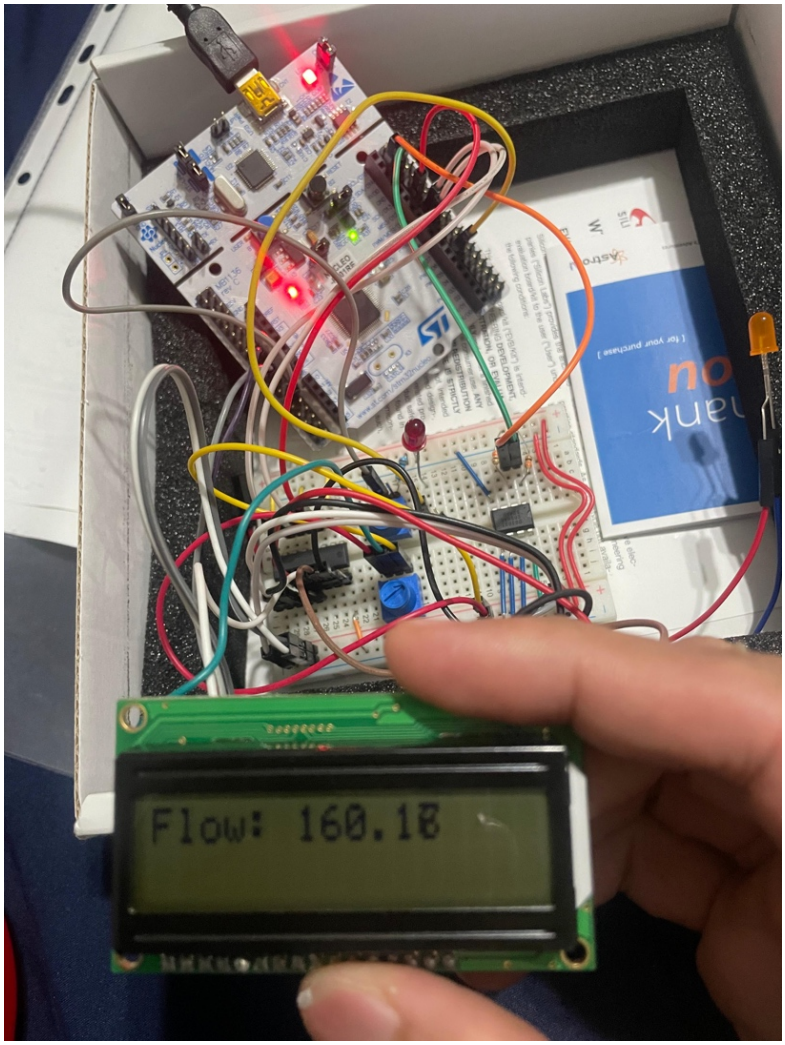

Figure 4 : LCD Displaying Flow in Gallons per minute

Code Analysis & Documentation

CPP Checker

We ran the CPP checker as seen in figure(5). After running the CPP checker, we discovered that the function read_serial_input() was never invoked. A review of the code revealed the function to be superfluous, leading us to comment it out. Another issue arose with the function tasked with reading the register address in memory.cpp. Considering our requirement for a C callable assembly function, we opted to keep this anomaly in alignment with our code's specific needs.

Further examination of the CPP checker results revealed an issue concerning the va_list 'v' being initiated but not terminated with va_end(). This was rectified by inserting a va_end() in the relevant location. Additionally, a suggestion was made to narrow the scope of the variable ‘j’. To address this, we confined the declaration of 'j' within a while loop. Originally, 'j' was declared once outside the while loop. While the fix involved redeclaring it inside the loop, this meant that 'j' would be redeclared every time the loop iterates, which might be considered redundant.

Please refer to Figure(6) for the final warnings and errors after addressing all identified issues. A detailed report can be found in the CppCheckReport.txt, which is housed in the CPPCheck folder.

```
root@Reaper:/mnt/c/Users/Reaper/Desktop/Flow Meter Simulink Comments/Flow Meter Simulink Comments/Code6# cppcheck --enable=all --max-ctu-depth=25 ../Code6 --output-f
ile=report.txt
Checking ../Code6/Code5/adc.cpp ...
1/10 files checked 15% done
Checking ../Code6/Code5/flow.cpp ...
2/10 files checked 17% done
Checking ../Code6/Code5/memory.cpp ...
3/10 files checked 29% done
Checking ../Code6/Code5/pwm.cpp ...
4/10 files checked 31% done
Checking ../Code6/Code5/src/DS1631.cpp ...
5/10 files checked 32% done
Checking ../Code6/Code5/src/Monitor.cpp ...
Checking ../Code6/Code5/src/Monitor.cpp: MAIN...
6/10 files checked 56% done
Checking ../Code6/Code5/src/NHD_0216HZ.cpp ...
7/10 files checked 59% done
Checking ../Code6/Code5/src/UART_poll.cpp ...
Checking ../Code6/Code5/src/UART_poll.cpp: MAIN...
8/10 files checked 75% done
Checking ../Code6/Code5/src/main.cpp ...
9/10 files checked 83% done
Checking ../Code6/Code5/src/timer0.cpp ...
Checking ../Code6/Code5/src/timer0.cpp: MAIN...
10/10 files checked 100% done
root@Reaper:/mnt/c/Users/Reaper/Desktop/Flow Meter Simulink Comments/Flow Meter Simulink Comments/Code6#
```

Figure 5 : CPP Checker Terminal Window

```
root@Reaper:/mnt/c/Users/Reaper/Desktop/Flow Meter Simulink Comments/Flow Meter Simulink Comments/CppCheck# cat CppCheckReport.txt
../Code6/Code5/memory.cpp:70:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R0
    ^
../Code6/Code5/memory.cpp:76:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R1
    ^
../Code6/Code5/memory.cpp:82:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R2
    ^
../Code6/Code5/memory.cpp:88:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R3
    ^
../Code6/Code5/memory.cpp:94:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R4
    ^
../Code6/Code5/memory.cpp:100:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R5
    ^
../Code6/Code5/memory.cpp:106:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R6
    ^
../Code6/Code5/memory.cpp:112:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R7
    ^
../Code6/Code5/memory.cpp:118:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R8
    ^
../Code6/Code5/memory.cpp:124:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R9
    ^
../Code6/Code5/memory.cpp:130:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R10
    ^
../Code6/Code5/memory.cpp:136:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R11
    ^
../Code6/Code5/memory.cpp:142:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, R12
    ^
../Code6/Code5/memory.cpp:149:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, SP
    ^
../Code6/Code5/memory.cpp:156:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, LR
    ^
../Code6/Code5/memory.cpp:163:5: error: Found a exit path from function with non-void return type that has missing return statement [missingReturn]
    MOV R0, PC
    ^
../Code6/Code5/src/NHD_0216HZ.cpp:49:10: warning: Redundant assignment of 'hi_n' to itself. [selfAssignment]
    hi_n = hi_n = (data & 0xF0);
         ^
../Code6/Code5/src/UART_poll.cpp:271:0: style: The function 'UART_direct_hex_put' is never used. [unusedFunction]

^
../Code6/Code5/src/UART_poll.cpp:260:0: style: The function 'UART_hex_put' is never used. [unusedFunction]

^
../Code6/Code5/src/UART_poll.cpp:248:0: style: The function 'asc_to_hex' is never used. [unusedFunction]

^
../Code6/Code5/src/Monitor.cpp:303:0: style: The function 'is_hex' is never used. [unusedFunction]

root@Reaper:/mnt/c/Users/Reaper/Desktop/Flow Meter Simulink Comments/Flow Meter Simulink Comments/CppCheck#
```

Figure 6 : CPP Check results after fixes

Doxygen

Doxygen documentation was generated and available inside folder called “Doxygen Documentation” in final submitted zip file.

Lab Questions :

Question 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.

Question 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.

Question 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.

Question 4 : How much time does the code spend in the main loop versus in Interrupt Service Routines?

Utilizing the Timer class from the mbed API, we initiated the timer using timer.start() at the beginning of both our main.cpp and ISR functions. By employing timer.reset(), timer.stop() at the conclusion of each function, we obtained specific timings for both the main function and the ISR, as illustrated in Figure(6) below.

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

Times calculated is before any command given to debug monitor. The reason being value could change for the different debug monitor commands, so we found the best way could be to test it before sending any command to debug monitor.

Question 5 : Estimate the % of CPU cycles used for the main foreground process, assuming a 100 millisecond operating cycle.

Time taken by Main() = 0.6468 seconds
Time taken by ISR = 2 usec = 0.000002 seconds

Since the Timer ISR is set at 100 microseconds and gets called 1000 times in 100 milliseconds, the total time for ISR in 100 milliseconds = 0.000002 seconds * 1000 = 0.002 seconds

Total time for both Main() and ISR in 100 milliseconds = 0.6468 + 0.002 = 0.6488 seconds

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

We estimate that, during a 100 millisecond operating cycle, the main background process uses approximately 99.69% of the CPU cycles, calculated as (0.6468 seconds / 0.6488 seconds) * 100.

Question 6 : Calculate the power consumption for your complete system (including proposed hardware additions) when in full run mode, and again in low power mode.

Power Consumption Analysis:

- **STM32F401RE Microcontroller**
  - Voltage (Vdd):
    - Minimum: -0.3V
    - Maximum: 4.0V
    - Maximum (at ambient temperature): 3.6V
    - Current Value: 3.3V
  - Current (Idd):
    - Maximum input: 100mA
    - Consumption (at 25°C, peripherals on, Vdd = 3.3V, Code Running From FLASH in run mode with data processing): 8.2mA
  - Oscillator Frequency: 16MHz

  Maximum Power Consumption: 3.3V * 16Mhz * 8.2mA/Mhz = 432.96mW
- **LCD - NHD-0216HZ**
  - Voltage (Vdd): 3.3V (maximum)
  - Current (Idd): 2.5mA (maximum)

  Maximum Power Consumption: 3.3V * 2.5mA = 8.25mW
- **Shift Register - SN74HC595**
  - Voltage (Vdd):
    - Minimum: -0.5V
    - Maximum: 7.0V
  - Current (Idd): ±20mA

  Power Consumption:
    - Min-Max: 0.01mW to 0.14mW
    - At 3.3V Current Value - 0.066mW

Total Maximum Power Consumption (Full Run Mode): 432.96mW + 8.25mW + 0.066mW = 441.276mW

ADC Power Consumption in Sleep Mode:

- Current (Idd):
  - ADC maximum usage: 2.98uA/MHz
  - In sleep mode: 8.2mA-2.98uA = 8.19702mA
- Voltage (Vdd): 3.3V

Note: Other peripherals, such as UART and PWM, remain active, so the ADC supply current is subtracted.

Total Power Consumption in Low Power ADC Mode: 3.3V * 16Mhz * 8.19702mA/Mhz = 432.802656mW (Slightly Greater Than This)

Code

MATLAB

Flow_Meter_Script.m :

```
% ------------------------------------------------------------------
% Title: Flow Meter Simulation Parameters Loader
% Brief: This file loads parameters for the simulation of a model called Flow_Meter_Simulink.slx.
% Author: Kian Jojare, Viraj Patel
% Course: ECEN 5803 : Mastering Embedded System Architecture
% Date: 23 Oct 2023
% ------------------------------------------------------------------

% Clear console, variables, and all loaded functions
clc;          % Clear console output
clear;        % Clear workspace variables
clear all;    % Clear loaded functions and data

% Prompt user for the stop time of the simulation
Stop_Time = input('Enter The Stop Time In Seconds: ');

% Prompt user for the frequency of the wave in cycles/sec (Hz)
Frequency = input('Enter The Frequency In Hz: ');

% Convert frequency from cycles/sec (Hz) to radian/sec
Frequency = Frequency * 2 * pi;

% The number of samples in a single period of the wave is crucial for its accurate representation.
% Given by the formula: Samples per period = 2*pi / (Frequency * Sample time)
% We keep it greater than 256 for better resolution.
Samples_Per_Period = 1024;  % Define the number of samples per period

% Calculate the time interval between two consecutive samples
Sample_Time = (2 * pi) / (Frequency * Samples_Per_Period);

% Load and run the Simulink model
Model_Name = 'Flow_Meter_Simulink';
open_system(Model_Name);   % Open the Simulink model
sim(Model_Name);           % Run the Simulink simulation
```

Counting Zero Crossing :

```
function y = fcn(u)
    % Persistent variable to keep track of the count
    persistent count;

    % Initialize the count variable if it's empty
    if isempty(count)
        count = 0;
    end

    % Increment count if u is greater than 1
    if u == 1
        count = count + 1;
    end

    % Output the current count
    y = count;
end
```

Calculate Density :

```
function rho = fcn(T)
    rho = 1000*(1 - ((T+288.9414)/(508929.2*(T+68.12963)))*(T-3.9863)^2); % in kg/m^3
end
```

Calculate Viscosity :

```
function viscosity = fcn(T)
    T_Kelvin = T + 273.15; % Convert Celsius to Kelvin
    viscosity = 2.4e-5 * 10^(247.8 / (T_Kelvin - 140));
end
```

Compute Reynolds Number :

```
function Re = fcn(rho, viscosity, velocity)
    %persistent isFirstRun;

    %if isempty(isFirstRun)
    %    velocity = 10; % m/s, set the velocity to 10 m/s on the first run
    %    isFirstRun = false;
    %end

    PID = 2.900 * 0.0254; % Convert from inches to meters
    Re = (rho * velocity * PID) / viscosity;
end
```

Compute Strouhal Number :

```
function St = fcn(Re)
    St = 0.2684 - 1.0356 / sqrt(Re);
end
```

Calculate Velocity :

```
function v = fcn(f, St)
    d = 0.5;
    v = f * d / St; % Average fluid velocity in inch per second
end
```

Compute Flow :

```
% Compute Flow (in gallons per minute)
function Flow = fcn(v)
    PID = 2.900; % Inner diameter in inches
    v_feet_per_second = v / 12; % Convert velocity from inches/s to feet/s
    Flow = 2.45 * PID^2 * v_feet_per_second;
end
```

C Code

flow.cpp :

```
/**
 * \file flow.cpp
 * \author Kiran Jojare, Viraj Patel
 * \brief Implements flow calculation based on temperature and ADC readings for STM32F401xE.
 */

#include "flow.h"
#include <math.h>
#include "adc.h"

float velocity = 10;          ///< Initial velocity value.
float flow;                   ///< Calculated flow value.
float temperature;            ///< Temperature in Celsius.
extern float Frequency;       ///< External reference to frequency calculated from ADC readings.

/**
 * \brief Calculates the fluid flow based on temperature, density, and viscosity.
 *
 * The function reads the temperature and based on it calculates the fluid's density
 * and viscosity. Using these values and the Reynold's number, it computes the
 * Strouhal number and uses it to adjust the velocity. Finally, it calculates
 * and sets the flow rate.
 */
void calculate_flow(void) {
    temperature = readTemperature();  // Read the temperature in Celsius.

    // Compute the fluid density, rho, based on temperature.
    float rho = 1000 * (1 - ((temperature + 288.9414) / (508929.2 * (temperature + 68.12963)))
                * pow((temperature - 3.9863), 2));

    // Compute the fluid's viscosity using the Sutherland formula.
    float T_kelvin = temperature + 273.15;
    float viscosity = 2.4e-5 * pow(10, (247.8 / (T_kelvin - 140.0)));

    // Compute the Reynold's number.
    float Re = (rho * velocity * PID_METER) / viscosity;

    // Compute the Strouhal number based on Reynold's number.
    float St = 0.2684 - 1.0356 / sqrt(Re);

    // Adjust velocity based on Strouhal number and frequency.
    velocity = (Frequency * D) / St;

    // Calculate the flow rate.
    flow = 2.45 * pow(PID_INCH, 2) * (velocity / 12.0);
}
```

flow.h

```
/**
 * \file flow.h
 * \author Kiran Jojare, Viraj Patel
 * \brief Header file for flow calculations related constants and functions for STM32F401xE.
 */

#ifndef FLOW_H
#define FLOW_H

#define D 0.5              ///< Bluff body width in inches.
#define PID_INCH 2.900     ///< Pipe inner diameter in inches.
#define PID_METER 0.07366  ///< Pipe inner diameter in meters (Equivalent of 2.900 inches in meters).

/**
 * \brief Calculates the fluid flow.
 *
 * Computes the fluid flow based on various parameters including temperature,
 * density, and viscosity. Uses predefined constants for bluff body width and
```

```
 * pipe inner diameter.
 */
void calculate_flow(void);

#endif // FLOW_H
```

adc.cpp :

```
/**
 * \file adc.cpp
 * \author Kiran Jojare, Viraj Patel
 * \brief ADC and temperature measurement for STM32F401xE.
 */

#include "adc.h"
#include <math.h>
#include "stm32f401xe.h"
#include "mbed.h"

/// Global variable to store the calculated frequency
float Frequency = 0;

/// ADC channel for internal voltage reference
AnalogIn VREFINT(PA_0);
/// ADC channel for measuring vortex frequency
AnalogIn Vortex_Frequency(PA_1);
/// ADC channel for the on-chip temperature sensor
AnalogIn TempSensor(PA_4);

/// Hardcoded statically allocated array of ADC samples obtained from professor
/// Mirrors the data from ECEN5803Data14.txt
unsigned short ADC_SAMPLES[] = {
        0x7FFF, 0x8404, 0x8808, 0x8C0B, 0x900A, 0x9405, 0x97FB, 0x9BEB, 0x9FD4, 0xA3B5, 0xA78D,
0xAB5B, 0xAF1E, 0xB2D5, 0xB67F, 0xBA1B, 0xBDA9, 0xC127, 0xC495, 0xC7F1, 0xCB3B, 0xCE72, 0xD196,
0xD4A5, 0xD79E, 0xDA81, 0xDD4D, 0xE002, 0xE29F, 0xE522, 0xE78C, 0xE9DC, 0xEC12, 0xEE2B, 0xF029,
0xF20B, 0xF3D0, 0xF578, 0xF701, 0xF86D, 0xF9BB, 0xFAE9, 0xFBF9, 0xFCE9, 0xFDBA, 0xFE6B, 0xFEFC,
0xFF6D, 0xFFBE, 0xFFEE, 0xFFFF, 0xFFEE, 0xFFBE, 0xFF6D, 0xFEFC, 0xFE6B, 0xFDBA, 0xFCE9, 0xFBF9,
0xFAE9, 0xF9BB, 0xF86D, 0xF701, 0xF578, 0xF3D0, 0xF20B, 0xF029, 0xEE2B, 0xEC12, 0xE9DC, 0xE78C,
0xE522, 0xE29F, 0xE002, 0xDD4D, 0xDA81, 0xD79E, 0xD4A5, 0xD196, 0xCE72, 0xCB3B, 0xC7F1, 0xC495,
0xC127, 0xBDA9, 0xBA1B, 0xB67F, 0xB2D5, 0xAF1E, 0xAB5B, 0xA78D, 0xA3B5, 0x9FD4, 0x9BEB, 0x97FB,
0x9405, 0x900A, 0x8C0B, 0x8808, 0x8404,
        0x7FFF, 0x7BFA, 0x77F6, 0x73F3, 0x6FF4, 0x6BF9, 0x6803, 0x6413, 0x602A, 0x5C49, 0x5871,
0x54A3, 0x50E0, 0x4D29, 0x497F, 0x45E3, 0x4255, 0x3ED7, 0x3B69, 0x380D, 0x34C3, 0x318C, 0x2E68,
0x2B59, 0x2860, 0x257D, 0x22B1, 0x1FFC, 0x1D5F, 0x1ADC, 0x1872, 0x1622, 0x13EC, 0x11D3, 0x0FD5,
0x0DF3, 0x0C2E, 0x0A86, 0x08FD, 0x0791, 0x0643, 0x0515, 0x0405, 0x0315, 0x0244, 0x0193, 0x0102,
0x0091, 0x0040, 0x0010, 0x0000, 0x0010, 0x0040, 0x0091, 0x0102, 0x0193, 0x0244, 0x0315, 0x0405,
0x0515, 0x0643, 0x0791, 0x08FD, 0x0A86, 0x0C2E, 0x0DF3, 0x0FD5, 0x11D3, 0x13EC, 0x1622, 0x1872,
0x1ADC, 0x1D5F, 0x1FFC, 0x22B1, 0x257D, 0x2860, 0x2B59, 0x2E68, 0x318C, 0x34C3, 0x380D, 0x3B69,
0x3ED7, 0x4255, 0x45E3, 0x497F, 0x4D29, 0x50E0, 0x54A3, 0x5871, 0x5C49, 0x602A, 0x6413, 0x6803,
0x6BF9, 0x6FF4, 0x73F3, 0x77F6, 0x7BFA,
        0x7FFF, 0x8404, 0x8808, 0x8C0B, 0x900A, 0x9405, 0x97FB, 0x9BEB, 0x9FD4, 0xA3B5, 0xA78D,
0xAB5B, 0xAF1E, 0xB2D5, 0xB67F, 0xBA1B, 0xBDA9, 0xC127, 0xC495, 0xC7F1, 0xCB3B, 0xCE72, 0xD196,
0xD4A5, 0xD79E, 0xDA81, 0xDD4D, 0xE002, 0xE29F, 0xE522, 0xE78C, 0xE9DC, 0xEC12, 0xEE2B, 0xF029,
0xF20B, 0xF3D0, 0xF578, 0xF701, 0xF86D, 0xF9BB, 0xFAE9, 0xFBF9, 0xFCE9, 0xFDBA, 0xFE6B, 0xFEFC,
0xFF6D, 0xFFBE, 0xFFEE, 0xFFFF, 0xFFEE, 0xFFBE, 0xFF6D, 0xFEFC, 0xFE6B, 0xFDBA, 0xFCE9, 0xFBF9,
0xFAE9, 0xF9BB, 0xF86D, 0xF701, 0xF578, 0xF3D0, 0xF20B, 0xF029, 0xEE2B, 0xEC12, 0xE9DC, 0xE78C,
0xE522, 0xE29F, 0xE002, 0xDD4D, 0xDA81, 0xD79E, 0xD4A5, 0xD196, 0xCE72, 0xCB3B, 0xC7F1, 0xC495,
0xC127, 0xBDA9, 0xBA1B, 0xB67F, 0xB2D5, 0xAF1E, 0xAB5B, 0xA78D, 0xA3B5, 0x9FD4, 0x9BEB, 0x97FB,
0x9405, 0x900A, 0x8C0B, 0x8808, 0x8404,
        0x7FFF, 0x7BFA, 0x77F6, 0x73F3, 0x6FF4, 0x6BF9, 0x6803, 0x6413, 0x602A, 0x5C49, 0x5871,
0x54A3, 0x50E0, 0x4D29, 0x497F, 0x45E3, 0x4255, 0x3ED7, 0x3B69, 0x380D, 0x34C3, 0x318C, 0x2E68,
0x2B59, 0x2860, 0x257D, 0x22B1, 0x1FFC, 0x1D5F, 0x1ADC, 0x1872, 0x1622, 0x13EC, 0x11D3, 0x0FD5,
0x0DF3, 0x0C2E, 0x0A86, 0x08FD, 0x0791, 0x0643, 0x0515, 0x0405, 0x0315, 0x0244, 0x0193, 0x0102,
0x0091, 0x0040, 0x0010, 0x0000, 0x0010, 0x0040, 0x0091, 0x0102, 0x0193, 0x0244, 0x0315, 0x0405,
0x0515, 0x0643, 0x0791, 0x08FD, 0x0A86, 0x0C2E, 0x0DF3, 0x0FD5, 0x11D3, 0x13EC, 0x1622, 0x1872,
0x1ADC, 0x1D5F, 0x1FFC, 0x22B1, 0x257D, 0x2860, 0x2B59, 0x2E68, 0x318C, 0x34C3, 0x380D, 0x3B69,
0x3ED7, 0x4255, 0x45E3, 0x497F, 0x4D29, 0x50E0, 0x54A3, 0x5871, 0x5C49, 0x602A, 0x6413, 0x6803,
0x6BF9, 0x6FF4, 0x73F3, 0x77F6, 0x7BFA,
        0x7FFF, 0x8404, 0x8808, 0x8C0B, 0x900A, 0x9405, 0x97FB, 0x9BEB, 0x9FD4, 0xA3B5, 0xA78D,
0xAB5B, 0xAF1E, 0xB2D5, 0xB67F, 0xBA1B, 0xBDA9, 0xC127, 0xC495, 0xC7F1, 0xCB3B, 0xCE72, 0xD196,
0xD4A5, 0xD79E, 0xDA81, 0xDD4D, 0xE002, 0xE29F, 0xE522, 0xE78C, 0xE9DC, 0xEC12, 0xEE2B, 0xF029,
0xF20B, 0xF3D0, 0xF578, 0xF701, 0xF86D, 0xF9BB, 0xFAE9, 0xFBF9, 0xFCE9, 0xFDBA, 0xFE6B, 0xFEFC,
0xFF6D, 0xFFBE, 0xFFEE, 0xFFFF, 0xFFEE, 0xFFBE, 0xFF6D, 0xFEFC, 0xFE6B, 0xFDBA, 0xFCE9, 0xFBF9,
0xFAE9, 0xF9BB, 0xF86D, 0xF701, 0xF578, 0xF3D0, 0xF20B, 0xF029, 0xEE2B, 0xEC12, 0xE9DC, 0xE78C,
```

```
0xE522, 0xE29F, 0xE002, 0xDD4D, 0xDA81, 0xD79E, 0xD4A5, 0xD196, 0xCE72, 0xCB3B, 0xC7F1, 0xC495,
0xC127, 0xBDA9, 0xBA1B, 0xB67F, 0xB2D5, 0xAF1E, 0xAB5B, 0xA78D, 0xA3B5, 0x9FD4, 0x9BEB, 0x97FB,
0x9405, 0x900A, 0x8C0B, 0x8808, 0x8404,
       0x7FFF, 0x7BFA, 0x77F6, 0x73F3, 0x6FF4, 0x6BF9, 0x6803, 0x6413, 0x602A, 0x5C49, 0x5871,
0x54A3, 0x50E0, 0x4D29, 0x497F, 0x45E3, 0x4255, 0x3ED7, 0x3B69, 0x380D, 0x34C3, 0x318C, 0x2E68,
0x2B59, 0x2860, 0x257D, 0x22B1, 0x1FFC, 0x1D5F, 0x1ADC, 0x1872, 0x1622, 0x13EC, 0x11D3, 0x0FD5,
0x0DF3, 0x0C2E, 0x0A86, 0x08FD, 0x0791, 0x0643, 0x0515, 0x0405, 0x0315, 0x0244, 0x0193, 0x0102,
0x0091, 0x0040, 0x0010, 0x0000, 0x0010, 0x0040, 0x0091, 0x0102, 0x0193, 0x0244, 0x0315, 0x0405,
0x0515, 0x0643, 0x0791, 0x08FD, 0x0A86, 0x0C2E, 0x0DF3, 0x0FD5, 0x11D3, 0x13EC, 0x1622, 0x1872,
0x1ADC, 0x1D5F, 0x1FFC, 0x22B1, 0x257D, 0x2860, 0x2B59, 0x2E68, 0x318C, 0x34C3, 0x380D, 0x3B69,
0x3ED7, 0x4255, 0x45E3, 0x497F, 0x4D29, 0x50E0, 0x54A3, 0x5871, 0x5C49, 0x602A, 0x6413, 0x6803,
0x6BF9, 0x6FF4, 0x73F3, 0x77F6, 0x7BFA,
       0x7FFF, 0x8404, 0x8808, 0x8C0B, 0x900A, 0x9405, 0x97FB, 0x9BEB, 0x9FD4, 0xA3B5, 0xA78D,
0xAB5B, 0xAF1E, 0xB2D5, 0xB67F, 0xBA1B, 0xBDA9, 0xC127, 0xC495, 0xC7F1, 0xCB3B, 0xCE72, 0xD196,
0xD4A5, 0xD79E, 0xDA81, 0xDD4D, 0xE002, 0xE29F, 0xE522, 0xE78C, 0xE9DC, 0xEC12, 0xEE2B, 0xF029,
0xF20B, 0xF3D0, 0xF578, 0xF701, 0xF86D, 0xF9BB, 0xFAE9, 0xFBF9, 0xFCE9, 0xFDBA, 0xFE6B, 0xFEFC,
0xFF6D, 0xFFBE, 0xFFEE, 0xFFFF, 0xFFEE, 0xFFBE, 0xFF6D, 0xFEFC, 0xFE6B, 0xFDBA, 0xFCE9, 0xFBF9,
0xFAE9, 0xF9BB, 0xF86D, 0xF701, 0xF578, 0xF3D0, 0xF20B, 0xF029, 0xEE2B, 0xEC12, 0xE9DC, 0xE78C,
0xE522, 0xE29F, 0xE002, 0xDD4D, 0xDA81, 0xD79E, 0xD4A5, 0xD196, 0xCE72, 0xCB3B, 0xC7F1, 0xC495,
0xC127, 0xBDA9, 0xBA1B, 0xB67F, 0xB2D5, 0xAF1E, 0xAB5B, 0xA78D, 0xA3B5, 0x9FD4, 0x9BEB, 0x97FB,
0x9405, 0x900A, 0x8C0B, 0x8808, 0x8404,
       0x7FFF, 0x7BFA, 0x77F6, 0x73F3, 0x6FF4, 0x6BF9, 0x6803, 0x6413, 0x602A, 0x5C49, 0x5871,
0x54A3, 0x50E0, 0x4D29, 0x497F, 0x45E3, 0x4255, 0x3ED7, 0x3B69, 0x380D, 0x34C3, 0x318C, 0x2E68,
0x2B59, 0x2860, 0x257D, 0x22B1, 0x1FFC, 0x1D5F, 0x1ADC, 0x1872, 0x1622, 0x13EC, 0x11D3, 0x0FD5,
0x0DF3, 0x0C2E, 0x0A86, 0x08FD, 0x0791, 0x0643, 0x0515, 0x0405, 0x0315, 0x0244, 0x0193, 0x0102,
0x0091, 0x0040, 0x0010, 0x0000, 0x0010, 0x0040, 0x0091, 0x0102, 0x0193, 0x0244, 0x0315, 0x0405,
0x0515, 0x0643, 0x0791, 0x08FD, 0x0A86, 0x0C2E, 0x0DF3, 0x0FD5, 0x11D3, 0x13EC, 0x1622, 0x1872,
0x1ADC, 0x1D5F, 0x1FFC, 0x22B1, 0x257D, 0x2860, 0x2B59, 0x2E68, 0x318C, 0x34C3, 0x380D, 0x3B69,
0x3ED7, 0x4255, 0x45E3, 0x497F, 0x4D29, 0x50E0, 0x54A3, 0x5871, 0x5C49, 0x602A, 0x6413, 0x6803,
0x6BF9, 0x6FF4, 0x73F3, 0x77F6, 0x7BFA,
       0x7FFF, 0x8404, 0x8808, 0x8C0B, 0x900A, 0x9405, 0x97FB, 0x9BEB, 0x9FD4, 0xA3B5, 0xA78D,
0xAB5B, 0xAF1E, 0xB2D5, 0xB67F, 0xBA1B, 0xBDA9, 0xC127, 0xC495, 0xC7F1, 0xCB3B, 0xCE72, 0xD196,
0xD4A5, 0xD79E, 0xDA81, 0xDD4D, 0xE002, 0xE29F, 0xE522, 0xE78C, 0xE9DC, 0xEC12, 0xEE2B, 0xF029,
0xF20B, 0xF3D0, 0xF578, 0xF701, 0xF86D, 0xF9BB, 0xFAE9, 0xFBF9, 0xFCE9, 0xFDBA, 0xFE6B, 0xFEFC,
0xFF6D, 0xFFBE, 0xFFEE, 0xFFFF, 0xFFEE, 0xFFBE, 0xFF6D, 0xFEFC, 0xFE6B, 0xFDBA, 0xFCE9, 0xFBF9,
0xFAE9, 0xF9BB, 0xF86D, 0xF701, 0xF578, 0xF3D0, 0xF20B, 0xF029, 0xEE2B, 0xEC12, 0xE9DC, 0xE78C,
0xE522, 0xE29F, 0xE002, 0xDD4D, 0xDA81, 0xD79E, 0xD4A5, 0xD196, 0xCE72, 0xCB3B, 0xC7F1, 0xC495,
0xC127, 0xBDA9, 0xBA1B, 0xB67F, 0xB2D5, 0xAF1E, 0xAB5B, 0xA78D, 0xA3B5, 0x9FD4, 0x9BEB, 0x97FB,
0x9405, 0x900A, 0x8C0B, 0x8808, 0x8404,
       0x7FFF, 0x7BFA, 0x77F6, 0x73F3, 0x6FF4, 0x6BF9, 0x6803, 0x6413, 0x602A, 0x5C49, 0x5871,
0x54A3, 0x50E0, 0x4D29, 0x497F, 0x45E3, 0x4255, 0x3ED7, 0x3B69, 0x380D, 0x34C3, 0x318C, 0x2E68,
0x2B59, 0x2860, 0x257D, 0x22B1, 0x1FFC, 0x1D5F, 0x1ADC, 0x1872, 0x1622, 0x13EC, 0x11D3, 0x0FD5,
0x0DF3, 0x0C2E, 0x0A86, 0x08FD, 0x0791, 0x0643, 0x0515, 0x0405, 0x0315, 0x0244, 0x0193, 0x0102,
0x0091, 0x0040, 0x0010, 0x0000, 0x0010, 0x0040, 0x0091, 0x0102, 0x0193, 0x0244, 0x0315, 0x0405,
0x0515, 0x0643, 0x0791, 0x08FD, 0x0A86, 0x0C2E, 0x0DF3, 0x0FD5, 0x11D3, 0x13EC, 0x1622, 0x1872,
0x1ADC, 0x1D5F, 0x1FFC, 0x22B1, 0x257D, 0x2860, 0x2B59, 0x2E68, 0x318C, 0x34C3, 0x380D, 0x3B69,
0x3ED7, 0x4255, 0x45E3, 0x497F, 0x4D29, 0x50E0, 0x54A3, 0x5871, 0x5C49, 0x602A, 0x6413, 0x6803,
0x6BF9, 0x6FF4, 0x73F3, 0x77F6, 0x7BFA
};
/**
 * \brief Read ADC samples and compute the frequency based on zero crossing.
 *
 * The function counts the number of zero crossings in the provided samples and calculates
 * the frequency based on the sample time and the number of samples.
 */
void readADC(void){
       int count = 0; // Counter to track zero crossings
       for(int i=0; i<SAMPLES; i++) {
              if (ADC_SAMPLES[i] == ZERO_CROSSING_LEVEL) {
                     count++; // Increment count on zero crossing
              }
       }
       Frequency = count / (SAMPLES * SAMPLE_TIME * 2);
}

/**
 * \brief Custom configuration for the ADC module.
 *
 * The function sets up the ADC for continuous conversion mode and configures it
 * to measure from the on-chip temperature sensor. It also sets the sampling time
 * and starts the ADC conversion.
 */
void ADC_Custom_Config(void) {
    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; // Enable the ADC1 clock

    ADC1->CR1 = 0;
```

```
    ADC1->CR1 |= ADC_CR1_SCAN;          // Enable scan mode for multiple channels

    ADC1->CR2 = 0;
    ADC1->CR2 |= ADC_CR2_ADON |         // Turn the ADC on
                 ADC_CR2_CONT;          // Set to continuous conversion mode

    ADC1->SQR1 = 0;
    ADC1->SQR3 = ADC_CHANNEL_TEMPSENSOR; // Set to use the on-chip temperature sensor channel

    ADC1->SMPR1 |= ADC_SMPR1_SMP16_2 | ADC_SMPR1_SMP16_1; // Set sampling time to 144+12 cycles

    ADC->CCR |= ADC_CCR_TSVREFE;  // Enable the temperature sensor and voltage reference buffer

    ADC1->CR2 |= ADC_CR2_SWSTART; // Start the ADC conversion
}

/**
 * \brief Read the on-chip temperature sensor and return the temperature value.
 *
 * \return The calculated temperature based on the ADC reading and the STM32 reference manual
formula.
 */
float readTemperature(void) {
    ADC1->CR2 |= ADC_CR2_SWSTART;                  // Start ADC conversion
    while(!(ADC1->SR & ADC_SR_EOC));               // Wait until the conversion is complete
    uint16_t adcValue = ADC1->DR;                  // Get the ADC result

    float voltage = (adcValue * 3.3) / 4095.0;     // Convert the ADC reading to a voltage value
    float temperature = ((voltage - ADC_V25) / ADC_AVG_SLOPE) + 25.0; // Convert voltage to
temperature
    return temperature;
}
```

adc.h :

```
/**
 * \file adc.h
 * \author Kiran Jojare, Viraj Patel
 * \brief ADC configurations and function declarations for STM32F401xE.
 */

#ifndef ADC_H
#define ADC_H

/// Defined value for zero crossing level (mid of 16-bit ADC range)
#define ZERO_CROSSING_LEVEL 0x7FFF
/// Number of samples for the readADC function
#define SAMPLES (1000U)
/// Time interval between each ADC sample
#define SAMPLE_TIME (100E-06)
/// Voltage corresponding to ADC value at 25 degrees Celsius (refer to the specific MCU's datasheet)
#define ADC_V25 0.76
/// Average slope value for temperature calculation based on the ADC reading
#define ADC_AVG_SLOPE 0.0025

/**
 * \brief Read ADC samples and compute the frequency based on zero crossing.
 *
 * The function counts the number of zero crossings in the provided samples and calculates
 * the frequency based on the sample time and the number of samples.
 */
void readADC(void);

/**
 * \brief Custom configuration for the ADC module.
 *
 * The function sets up the ADC for continuous conversion mode and configures it
 * to measure from the on-chip temperature sensor. It also sets the sampling time
 * and starts the ADC conversion.
 */
void ADC_Custom_Config(void);

/**
 * \brief Read the on-chip temperature sensor and return the temperature value.
 *
```

```
 * \return The calculated temperature based on the ADC reading and the STM32 reference manual
formula.
 */
float readTemperature(void);

#endif // ADC_H
```

pwm.cpp :

```
/**
 * \file PWM.cpp
 * \author Kiran Jojare, Viraj Patel
 * \brief PWM output configurations and related functions for STM32F401xE.
 *
 * This file provides functionality to generate PWM signals based on the flow and frequency
 * calculations from the flow sensing system.
 */

#include "pwm.h"
#include "mbed.h"

PwmOut pulse_420(PC_8);  ///< PWM output pin for 4-20mA signaling.
PwmOut pulse_out(PC_9);  ///< Standard PWM output pin.

extern float flow, Frequency;  ///< Externally defined flow and frequency values.

/**
 * \brief Output a PWM signal based on the flow rate.
 *
 * Generates a PWM signal with a frequency inversely proportional to the flow
 * rate (in gallons per second). The duty cycle is set to 50%.
 */
void pulse_420_output(void) {
        if(flow) {
                pulse_420.period(1/(flow/60));    // Convert to Gallons Per Second
                pulse_420.write(0.5);    // Set to 50% Duty Cycle
        }
}

/**
 * \brief Output a PWM signal based on the measured frequency.
 *
 * Generates a PWM signal with a frequency inversely proportional to the
 * frequency from the flow sensing system. The duty cycle is set to 50%.
 */
void pulse_output(void) {
        if(Frequency) {
                pulse_out.period(1/Frequency);    // Set period based on the measured frequency
                pulse_out.write(0.5);    // Set to 50% Duty Cycle
        }
}
```

pwm.h :

```
/**
 * \file PWM.h
 * \author Kiran Jojare, Viraj Patel
 * \brief PWM function declarations for STM32F401xE.
 *
 * This header provides declarations for the PWM signal generation functions.
 */

#ifndef PWM_H
#define PWM_H

/**
 * \brief Output a PWM signal based on the flow rate.
 *
 * Generates a PWM signal with a frequency inversely proportional to the flow rate
 * (in gallons per second). The duty cycle is set to 50%.
 */
void pulse_420_output(void);
```

```
/**
 * \brief Output a PWM signal based on the measured frequency.
 *
 * Generates a PWM signal with a frequency inversely proportional to the frequency
 * from the flow sensing system. The duty cycle is set to 50%.
 */
void pulse_output(void);

#endif // PWM_H
```

Monitor.cpp :

```
/**-----------------------------------------------------------------------------
             \file Monitor.cpp
--                                                                            --
--               ECEN 5003 Mastering Embedded System Architecture             --
--                   Project 1 Module 4                                       --
--                 Microcontroller Firmware                                   --
--                       Monitor.cpp                                            --
--                                                                            --
-------------------------------------------------------------------------------
--
--  Designed for:  University of Colorado at Boulder
--
--
--  Designed by:  Tim Scherr
--  Revised by:  Student's name
--
-- Version: 2.0
-- Date of current revision:  2022-06-20
-- Target Microcontroller: ST STM32F401RE
-- Tools used:  ARM mbed compiler
--              ARM mbed SDK
--              ST Nucleo STM32F401RE Board
--
--
   Functional Description: See below
--
--      Copyright (c) 2015, 2022 Tim Scherr All rights reserved.
--
*/

#include <stdio.h>
#include "shared.h"
#include "memory.h"

//extern bool redLEDFlag;
extern float temperature, flow, Frequency;


/*****************************************************************************
* Set Display Mode Function
* Function determines the correct display mode.  The 3 display modes operate as
*   follows:
*
*  NORMAL MODE       Outputs only mode and state information changes
*                     and calculated outputs
*
*  QUIET MODE        No Outputs
*
*  DEBUG MODE        Outputs mode and state information, error counts,
*                    register displays, sensor states, and calculated output
*                  (currently not all features are operation, could be enhanced)
*
* There is deliberate delay in switching between modes to allow the RS-232 cable
* to be plugged into the header without causing problems.
*****************************************************************************/


//*****************************************************************************/
/// \fn void set_display_mode(void)
///
/// \brief Displays a selection menu over UART.
///
```

```
/// This function sends a series of messages over UART to display a menu
/// allowing the user to choose between different modes. Each mode is
/// highlighted with a different color.
//*****************************************************************************/

void set_display_mode(void)
{
    UART_direct_msg_put("\033[37m\r\n+------------------------+\033[0m");
    UART_direct_msg_put("\r\n\033[37m|  Select Mode           |\033[0m");
    UART_direct_msg_put("\033[37m\r\n|------------------------|\033[0m");
    UART_direct_msg_put("\r\n\033[32m|  NOR - Normal          |\033[0m");
    UART_direct_msg_put("\r\n\033[33m|  QUI - Quiet           |\033[0m");
    UART_direct_msg_put("\r\n\033[35m|  DEB - Debug           |\033[0m");
    //UART_direct_msg_put("\r\n\033[31m|  RED - Red Led(1s)     |\033[0m");
    UART_direct_msg_put("\r\n\033[34m|  REG - Registers Dump  |\033[0m");
    UART_direct_msg_put("\r\n\033[31m|  MEM - Memory Dump     |\033[0m");
    UART_direct_msg_put("\r\n\033[36m|  STK - Stack Dump      |\033[0m");
    UART_direct_msg_put("\r\n\033[97m|  V   - Version#        |\033[0m");
    UART_direct_msg_put("\033[37m\r\n+------------------------+\033[0m");
    UART_direct_msg_put("\033[0m\r\nSelect:  ");
}



//*****************************************************************************/
/// \fn void chk_UART_msg(void)
///
///  \brief - fills a message buffer until return is encountered, then calls
///           message processing
//*****************************************************************************/
/*****************      ECEN 5803 add code as indicated   **********************/
  // Improve behavior of this function
void chk_UART_msg(void)
{
   uchar8_t j;
   while( UART_input() )      // becomes true only when a byte has been received
   {                                  // skip if no characters pending
      j = UART_get();                // get next character

      if( j == '\r' )          // on a enter (return) key press
      {                // complete message (all messages end in carriage return)
                              //UART_msg_put("");
         UART_direct_msg_put("->");
         UART_msg_process();
      }
      else
      {
         if ((j != 0x02) )         // if not ^B
         {                             // if not command, then
            UART_put(j);              // echo the character
         }
         else
         {
           ;
         }
         if( j == '\b' )
         {                             // backspace editor
            if( msg_buf_idx != 0)
            {                        // if not 1st character then destructive
               UART_msg_put(" \b");// backspace
               msg_buf_idx--;
            }
         }
         else if( msg_buf_idx >= MSG_BUF_SIZE )
         {                                // check message length too large
            UART_msg_put("\r\nToo Long!");
            msg_buf_idx = 0;
         }
         else if ((display_mode == QUIET) && (msg_buf[0] != 0x02) &&
                  (msg_buf[0] != 'D') && (msg_buf[0] != 'N') &&
                  (msg_buf[0] != 'V') && (msg_buf[0] != 'R') &&
                                                                  (msg_buf[0] != 'M') && (msg_buf[0]
!= 'S') &&
                  (msg_buf_idx != 0))
         {                           // if first character is bad in Quiet mode
```

```
            msg_buf_idx = 0;        // then start over
         }
         else {                        // not complete message, store character

            msg_buf[msg_buf_idx] = j;
            msg_buf_idx++;
            if (msg_buf_idx > 3)
            {
               UART_msg_process();
            }
         }
      }
   }
}


//*****************************************************************************/
///  \fn void UART_msg_process(void)
///UART Input Message Processing
//*****************************************************************************/
void UART_msg_process(void)
{
   uchar8_t chr,err=0;
//   unsigned char  data;

   // Check if the first character of the message buffer is an uppercase letter
   if( (chr = msg_buf[0]) <= 0x60 )
   {      // Upper Case
      switch( chr )
      {
         // DEBUG Mode
         case 'D':
            if((msg_buf[1] == 'E') && (msg_buf[2] == 'B') && (msg_buf_idx == 3))
            {
               display_mode = DEBUG;
               UART_direct_msg_put("\r\n\033[35mMode=DEBUG\033[0m\n");
               display_timer = 0;
            }
            else
               err = 1;
            break;

         // NORMAL Mode
         case 'N':
            if((msg_buf[1] == 'O') && (msg_buf[2] == 'R') && (msg_buf_idx == 3))
            {
               display_mode = NORMAL;
               UART_direct_msg_put("\r\n\033[32mMode=NORMAL\033[0m\n");
               //display_timer = 0;
            }
            else
               err = 1;
            break;

         // QUIET Mode
         case 'Q':
            if((msg_buf[1] == 'U') && (msg_buf[2] == 'I') && (msg_buf_idx == 3))
            {
               display_mode = QUIET;
               UART_direct_msg_put("\r\n\033[33mMode=QUIET\033[0m\n");
               display_timer = 0;
            }
            else
               err = 1;
            break;

         // VERSION Info
         case 'V':
            display_mode = VERSION;
            UART_direct_msg_put("\033[97m\r\n");
            UART_direct_msg_put(CODE_VERSION);
                              UART_direct_msg_put("\033[0m\r\nSelect:  ");
            display_timer = 0;
            break;
/***************      ECEN 5803 add code as indicated   *********************/
//  Add other message types here
```

```
                                /* For Red Led Flag Mode
                                case 'R':
                                       if((msg_buf[1] == 'E') && (msg_buf[2] == 'D') && (msg_buf_idx
== 3))
            {
               display_mode = RED;
               UART_msg_put("\r\n\033[31mMode=RED LED(1s)\033[0m\n");
               display_timer = 0;
            }
            else
               err = 1;
                                              break;
                                */
         // Register Dump
                                case 'R':
                                       if((msg_buf[1] == 'E') && (msg_buf[2] == 'G') && (msg_buf_idx
== 3))
            {
               display_mode = REGISTERS;
               UART_msg_put("\r\n\033[34mMode=Registers Dump\033[0m\n");
               display_timer = 0;
            }
            else
               err = 1;
                                              break;

         // Memory Dump
                                     case 'M':
                                      if((msg_buf[1] == 'E') && (msg_buf[2] == 'M') && (msg_buf_idx
== 3))
            {
               display_mode = MEMORY;
               UART_msg_put("\r\n\033[31mMode=Memory Dump\033[0m\n");
               display_timer = 0;
            }
            else
               err = 1;
                                              break;

                                     // Stack Dump
                                     case 'S':
                                      if((msg_buf[1] == 'T') && (msg_buf[2] == 'K') && (msg_buf_idx
== 3))
            {
               display_mode = STACK;
               UART_msg_put("\r\n\033[36mMode=Stack Dump\033[0m\n");
               display_timer = 0;
            }
            else
               err = 1;
                                              break;

         // DEFAULT
         default:
            err = 1;
      }
   }
   // Display error messages based on the error code
   else
   {                    // Lower Case
      switch( chr )
      {
        default:
         err = 1;
      }
   }

   if( err == 1 )
   {
      UART_direct_msg_put("\n\rEntry Error!");
   }
   else if( err == 2 )
   {
      UART_direct_msg_put("\n\rNot in DEBUG Mode!");
   }
   else
```

```
   {
    msg_buf_idx = 0;          // put index to start of buffer for next message
      ;
   }
   msg_buf_idx = 0;          // put index to start of buffer for next message


}


//*****************************************************************************
///   \fn   is_hex
/// Function takes
///  @param a single ASCII character and returns
///  @return 1 if hex digit, 0 otherwise.
///
//*****************************************************************************
uchar8_t is_hex(uchar8_t c)
{
   if( (((c |= 0x20) >= '0') && (c <= '9')) || ((c >= 'a') && (c <= 'f'))  )
      return 1;
   return 0;
}

/*****************************************************************************
*   \fn  DEBUG and DIAGNOSTIC Mode UART Operation
*****************************************************************************/
void monitor(void)
{

/*********************************/
/*     Spew outputs              */
/*********************************/
        char buffer[50];

   switch(display_mode)
   {
      case(QUIET):
         {
                                              //redLEDFlag = false;
             UART_msg_put("\r\n ");
             display_flag = 0;
         }
         break;
      case(VERSION):
         {
                                     //redLEDFlag = false;
             display_flag = 0;
         }
         break;
      case(NORMAL):
         {
                                    //redLEDFlag = false;
            if (display_flag == 1)
            {
               UART_direct_msg_put("\r\n\033[32mNORMAL ");
                                   //UART_msg_put(" Flow: ");

               // Printing Flow over UART
                                   sprintf(buffer, " Flow: %.2f", flow);
               UART_direct_msg_put(buffer);
               // ECEN 5803 add code as indicated
               //  add flow data output here, use UART_hex_put or similar for
               // numbers
               //UART_msg_put(" Temp: ");

               // Printing Temperature over UART
                                   sprintf(buffer, " Temp: %.2f", temperature);
               UART_direct_msg_put(buffer);
               //  add flow data output here, use UART_hex_put or similar for
               // numbers
               //UART_msg_put(" Freq: \033[0m");

               // Printing Frequency over UART
               sprintf(buffer, " Freq: %.2f\033[0m ", Frequency);
               UART_direct_msg_put(buffer);
```

```
            //  add flow data output here, use UART_hex_put or similar for
            // numbers
            display_flag = 0;
                                        //wait_ms(500);
         }
      }
      break;
   case(DEBUG):
      {
                                     //redLEDFlag = false;
         if (display_flag == 1)
         {
            UART_direct_msg_put("\r\n\033[35mDEBUG ");
            //UART_msg_put(" Flow: ");

            // Printing Flow over UART
                                    sprintf(buffer, " Flow: %.2f", flow);
            UART_direct_msg_put(buffer);
            // ECEN 5803 add code as indicated
            //  add flow data output here, use UART_hex_put or similar for
            // numbers
            //UART_msg_put(" Temp: ");

             // Printing Temperature over UART
                                    sprintf(buffer, " Temp: %.2f", temperature);
            UART_direct_msg_put(buffer);
            //  add flow data output here, use UART_hex_put or similar for
            // numbers
            //UART_msg_put(" Freq: \033[0m");

            // Printing Frequency over UART
                                    sprintf(buffer, " Freq: %.2f\033[0m ", Frequency);
            UART_direct_msg_put(buffer);
            //  add flow data output here, use UART_hex_put or similar for
            // numbers


/****************      ECEN 5803 add code as indicated   **********************/
            //  Create a display of  error counts, sensor states, and
            //  ARM Registers R0-R15
                                                //print_registers();

            //  Create a command to read a section of Memory and display it
            //dump_memory();

            //  Create a command to read 16 words from the current stack
            // and display it in reverse chronological order.
                                                //display_last_16_stack_words();


            // clear flag to ISR
            display_flag = 0;
                                                //wait_ms(500);
          }
      }
      break;
                  /* Red Led Flag Mode
                  case(RED):
                  {
                          redLEDFlag = true;
                          if (display_flag == 1) {
                                  UART_msg_put("\r\n\033[31mRed LED(1s)\033[0m ");
                                  display_flag = 0;
                          }
                  }
                  break;
                  */
                  // Check if the current mode is REGISTERS
                  case(REGISTERS):
                  {
                           // The following line is commented out, but if active, it would turn
off a red LED flag
                           //redLEDFlag = false;

                           // Check if the display_flag is set
                           if (display_flag == 1) {
```

```
                                        // Send a new line and set the text color to blue
                                        UART_direct_msg_put("\n\r\033[34m");

                                        // Print the register values
                                        print_registers();

                                        // Reset the text color to default and send a new line
                                        UART_direct_msg_put("\033[0m\n\r");

                                        // Reset the display_flag
                                        display_flag = 0;
                                }
                        }
                        break;  // End of REGISTERS case

                        // Check if the current mode is MEMORY
                        case(MEMORY):
                        {
                                // The following line is commented out, but if active, it would turn
off a red LED flag
                                //redLEDFlag = false;

                                // Check if the display_flag is set
                                if (display_flag == 1) {
                                        // Send a new line and set the text color to red
                                        UART_direct_msg_put("\n\r\033[31m");

                                        // Dump the memory values
                                        dump_memory();

                                        // Reset the text color to default and send a new line
                                        UART_direct_msg_put("\033[0m\n\r");

                                        // Reset the display_flag
                                        display_flag = 0;
                                }
                        }
                        break;  // End of MEMORY case

                        // Check if the current mode is STACK
                        case(STACK):
                        {
                                // The following line is commented out, but if active, it would turn
off a red LED flag
                                //redLEDFlag = false;

                                // Check if the display_flag is set
                                if (display_flag == 1) {
                                        // Send a new line and set the text color to cyan
                                        UART_direct_msg_put("\n\r\033[36m");

                                        // Display the last 16 words of the stack
                                        display_last_16_stack_words();

                                        // Reset the text color to default and send a new line
                                        UART_direct_msg_put("\033[0m\n\r");

                                        // Reset the display_flag
                                        display_flag = 0;
                                }
                        }
                        break;  // End of STACK case


      default:
      {
         UART_msg_put("Mode Error");
      }
   }
}
```

Main.cpp :

```
/**-------------------------------------------------------------------------
```

```
   \file main.cpp

--                                                                            --
--              ECEN 5803 Mastering Embedded System Architecture              --
--                  Project 1 Module 4                                        --
--                Microcontroller Firmware                                    --
--                     main.cpp                                              --
--                                                                            --
--------------------------------------------------------------------------------
--
--  Designed for:  University of Colorado at Boulder
--
--
--  Designed by:  Tim Scherr
--  Revised by:  Student's name
--
-- Version: 3.0
-- Date of current revision:  2022-06-20
-- Target Microcontroller: ST STM32F401RE
-- Tools used:  ARM mbed compiler
--              ARM mbed SDK
--              ST Nucleo STM32F401RE Board
--
--
-- Functional Description:  Main code file generated by mbed, and then
--                           modified to implement a super loop bare metal OS.
--
--      Copyright (c) 2015, 2016, 2022 Tim Scherr  All rights reserved.
--
*/

#define MAIN
#include "shared.h"
#undef MAIN

#include "NHD_0216HZ.h"
//#include "DS1631.h"
#include "pindef.h"

#include "adc.h"
#include "flow.h"
#include "pwm.h"

// Define green LED pin
DigitalOut greenLED(LED2);

// LCD setup using SPI communication.
NHD_0216HZ lcd(SPI_CS, SPI_MOSI, SPI_SCLK);

extern volatile uint16_t SwTimerIsrCounter;
extern float Frequency, flow, temperature;

Ticker tick;          //  Creates a timer interrupt using mbed methods

 /****************      ECEN 5803 add code as indicated   ***************/
   // Add code to control LED LD2 here,
   // including a function to flip the LED state on and off

void flip(void) {
    greenLED = !greenLED;  // Toggle the state of the green LED
}


// Set up serial communication over USB
Serial pc(USBTX, USBRX);

// Create timer instance
//Timer custom_timer2;



/*-----------------------------------------------------------------------------
 MAIN function
 *-----------------------------------------------------------------------------*/
```

```
int main()
{
        //custom_timer2.start();
        pc.baud(9600);

        // Initialize the LCD
        lcd.init_lcd();    // Initialize the LCD display.
        lcd.clr_lcd();     // Clear the LCD.

        // ADC channels are initialized using AnalogIn
  // Custom config for ADC
  ADC_Custom_Config();

/****************      ECEN 5803 add code as indicated   *********************/
               //  Add code to call timer0 function every 100 uS
        tick.attach_us(&timer0, 100);

 // Print the initial banner
    pc.printf("\r\nHello World!\n\n\r");
    uint32_t  count = 0;


// initialize serial buffer pointers
   rx_in_ptr =  rx_buf; /* pointer to the receive in data */
   rx_out_ptr = rx_buf; /* pointer to the receive out data*/
   tx_in_ptr =  tx_buf; /* pointer to the transmit in data*/
   tx_out_ptr = tx_buf; /* pointer to the transmit out */


/****************      ECEN 5803 add code as indicated   *********************/
    // uncomment this section after adding monitor code.
   /* send a starting message to the terminal  */
         pc.printf("\n\r\n\r***** Project 1 Module 6 - Viraj_Kiran *****\n\r\n\r");
   UART_direct_msg_put("\r\nSystem Reset\r\nCode ver. ");
   UART_direct_msg_put( CODE_VERSION );
   UART_direct_msg_put("\r\n");
   UART_direct_msg_put( COPYRIGHT );
   UART_direct_msg_put("\r\n");

   set_display_mode();

         //custom_timer2.stop();

    while(1)       /// Cyclical Executive Loop
    {

        count++;               // counts the number of times through the loop
//      __enable_interrupts();
//      __clear_watchdog_timer();

/****************      ECEN 5803 add code as indicated   *********************/
    // uncomment this section after you add the monitor code.

        serial();            // Polls the serial port
        chk_UART_msg();     // checks for a serial port message received
        monitor();           // Sends serial port output messages depending
                        //     on commands received and display mode

   /****************      ECEN 5803 add code as indicated   ***************/
  // Read temperature sensor readings from internal temperature sensor
  readADC();
        //pc.printf("Freq: %f\n\r", Frequency);

        // Calculate flow from obtained internal temperature sensor readings
  calculate_flow();

//  4-20 output ()    // use PWM1 channel 1  proportional rate to flow
  pulse_420_output();

//  Pulse output()   // use PWM1 channel 2  propotional pulse rate to frequency
  pulse_output();

//  LCD_Display()   // use the SPI port to send flow number
        lcd.set_cursor(0, 0);
        lcd.printf("Flow: %.2f ", flow);
        //lcd.printf("T: %.1f ", temperature);
```

```
        //lcd.set_cursor(0, 1);
        //lcd.printf("Fr: %.1f ", Frequency);

//  Write your code here for any additional tasks
               //pc.printf("Temp::: %.f\n\r", readTemperature());


//  End ECEN 5803 code addition



        if ((SwTimerIsrCounter & 0x1FFF) > 0x0FFF)
/****************      ECEN 5803 add code as indicated   *********************/
         /// Add comment explaining what the timer statement above does
        {
            flip();  // Toggle Green LED
        }

    }   /// End while(1) loop

}
```

CPPCheckReport.txt

```
[1m../Code6/Code5/memory.cpp:70:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R0
    ^
[1m../Code6/Code5/memory.cpp:76:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R1
    ^
[1m../Code6/Code5/memory.cpp:82:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R2
    ^
[1m../Code6/Code5/memory.cpp:88:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R3
    ^
[1m../Code6/Code5/memory.cpp:94:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R4
    ^
[1m../Code6/Code5/memory.cpp:100:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R5
    ^
[1m../Code6/Code5/memory.cpp:106:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R6
    ^
[1m../Code6/Code5/memory.cpp:112:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R7
    ^
[1m../Code6/Code5/memory.cpp:118:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R8
    ^
[1m../Code6/Code5/memory.cpp:124:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R9
    ^
[1m../Code6/Code5/memory.cpp:130:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R10
    ^
[1m../Code6/Code5/memory.cpp:136:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R11
    ^
```

```
[1m../Code6/Code5/memory.cpp:142:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, R12
    ^
[1m../Code6/Code5/memory.cpp:149:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, SP
    ^
[1m../Code6/Code5/memory.cpp:156:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, LR
    ^
[1m../Code6/Code5/memory.cpp:163:5: [31merror:[39m Found a exit path from function with non-void
return type that has missing return statement [missingReturn][0m
    MOV R0, PC
    ^
[1m../Code6/Code5/src/NHD_0216HZ.cpp:49:10: [31mwarning:[39m Redundant assignment of 'hi_n' to
itself. [selfAssignment][0m
    hi_n = hi_n = (data & 0xF0);
         ^
[1m../Code6/Code5/src/UART_poll.cpp:271:0: [31mstyle:[39m The function 'UART_direct_hex_put' is never
used. [unusedFunction][0m

^
[1m../Code6/Code5/src/UART_poll.cpp:260:0: [31mstyle:[39m The function 'UART_hex_put' is never used.
[unusedFunction][0m

^
[1m../Code6/Code5/src/UART_poll.cpp:248:0: [31mstyle:[39m The function 'asc_to_hex' is never used.
[unusedFunction][0m

^
[1m../Code6/Code5/src/Monitor.cpp:303:0: [31mstyle:[39m The function 'is_hex' is never used.
[unusedFunction][0m
```