**ECEN 5803**

**Mastering Embedded Systems Architecture**

**Project Report**

**Submitted By**

**Disha Modi**

**Swati Kadivar**

**Guided By**

**Prof. Timothy Scherr**

## 

## 0) Index or Table of Contents

1. Executive Summary 2
2. Problem Statement 2
   1. Objectives 2
3. Approach and methodology of evaluation 3

### Block diagram or Semi-Schematic diagram 3

### Hardware evaluation 3

1. Module test results 4
2. List of project deliverables 12
3. Recommendations: GO/NoGO 12
4. References 13
5. Project Staffing 13

## 1) Executive Summary

1. Performed approximate square root operation using bi-section method on uVision5 Keil5 platform using Cortex M0+ based FRDM-KL25Z hardware board.
2. Used FRDM-KL25Z in-built accelerometer to measure vibrations and capacitive touch slider to measure touch percentage. Observed vibrations with various LED colors and capacitive touch by means of intensity of the LED brightness.
3. Implemented timer functionality using timer ISR and interacted with UART to print messages through serial port. The serial communication and monitoring libraries developed for UART communication.
4. Added a few debug features like register dump and memory dump in existing features.
5. Implemented ADC interface to measure temperature using inbuilt LM35 temperature sensor available on FRDM-KL25Z.
6. Calculated frequency from 1000 sample sine wave data points using zero-crossing frequency detection algorithm. Used MATLAB Simulink to analyze the frequency detection algorithm.
7. Evaluated the Freescale Kinetis KL25Z MCU for use in a Flow Meter by means of frequency and temperature, implemented a mechanism to calculate flow. In order to calculate the flow, we implemented equations to calculate density, viscosity and velocity.

By this many various implementations, we evaluated performance of the FRDM-KL25z system.

## 2) Problem Statement

To evaluate FRDM-KL25z for a calculation of Vortex flow meter. Implement various programs like a vibration sensor using inbuilt accelerometer, capacitive touch slider indicator, some math operation like square root to understand CPU utilization.

### Objectives:

**Module1:** Write an assembly code subroutine to approximate the square root of an argument using the bisection method.

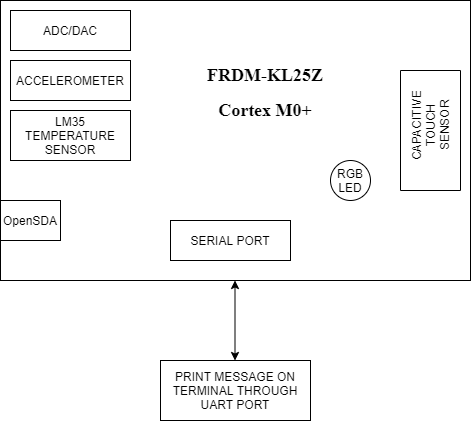
**Module2:** Using inbuilt accelerometer, detect and indicate vibrations by means of various LED colors. Detect capacitive touch on the slider and dim or bright the LED based on the touch sensor activity.

**Module3:** Develop RX/TX functions for UART serial communication. Using these functions implement a debug monitoring program which prints debug messages in different modes. Moreover, personalize additional debug modes like printing registers or memory sections.

**Module4:** Implement timer functionality and by means of ADC interface read the temperature from inbuilt LM35 temperature sensor of FRDM-KL25z. Calculate frequency from the samples given with frequency detection algorithm, sense temperature with LM35 and calculate the flow of the fluid from calculated temperature and frequency.

## 3) Approach and methodology of evaluation

### **a) Block diagram or Semi-Schematic diagram**



### b) Hardware evaluation

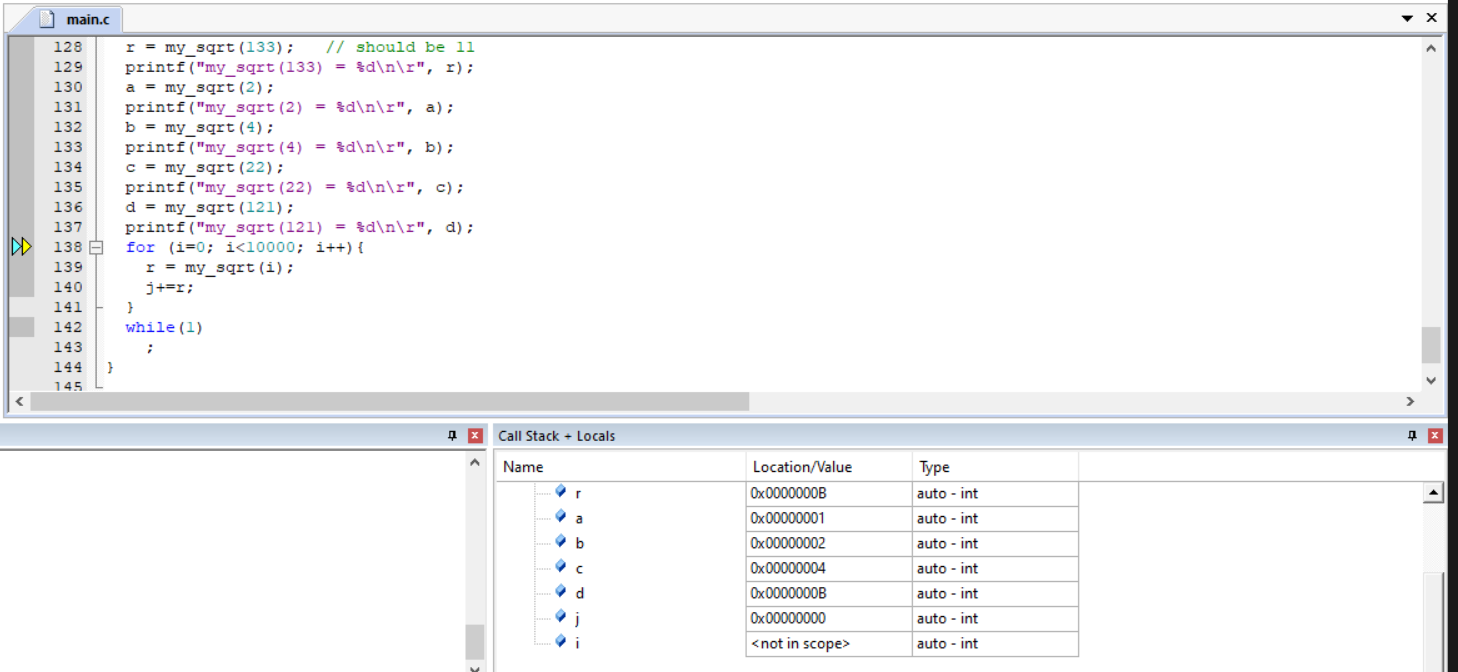
We implemented various applications to evaluate a few inbuilt hardware components like inbuilt LM35, accelerometer, capacitive touch slider etc. Also implemented square root calculator using bisection method. From the first three module implementations, we concluded that FRDM-KL25z is good for small operations but not good for big mathematical operations. We were able to implement a good vibration sensor and touch sensor but while implementing module 4, we faced issues with code size. If we try exhaustive calculations then due to code memory limitation, we may not be able to fit all the library files required for those calculations.

KL25Z supports only 32-bit operations. This means it will truncate the values if it is more than 32-bits and we will not be able to calculate flow values precisely. So KL25z is good for small functionalities but not for big math operations, it would be good to use some DSP processor with more memory and speed. DSP processors are more capable of doing fast floating point calculations and it will have an abundant number of cores, which are more suitable for flow calculation applications.

## 4) Module test results

### **Module 1 results:**

**3. Test your code with these inputs: 2, 4, 22, and 121. Record the results.**

****

****

**4. Estimate the number of CPU cycles used for this calculation.**

|  |  |
| --- | --- |
| Numbers | Cpu cycles |
| 2 | 120 |
| 4 | 103 |
| 22 | 127 |
| 121 | 142 |

**5. Doxygen document Link:**

The Doxygen link index.html is included in the html folder within the submitted project.

### **Module 2 results:**

Project implementation is submitted on canvas.

**2. Estimate the processor load in % of CPU cycles**

Total CPU time = 0.240004 s, Ideal time = 0.2 s

CPU load = (0.240004- 0.2)/ 0.240004 = 16.68%

**3. Doxygen document Link:**

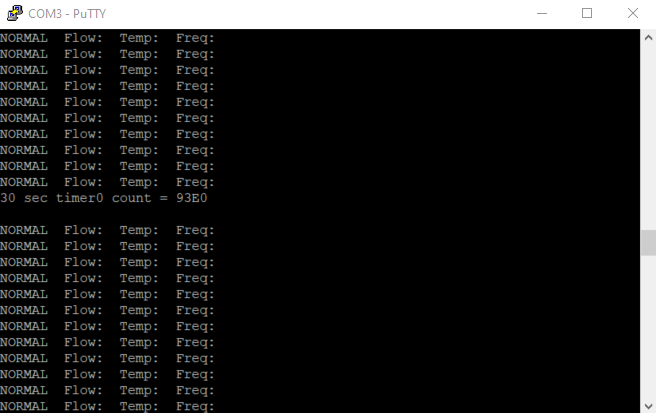
The Doxygen link index.html is included in the html folder within the submitted project.

**4. Video Link:**

[**https://drive.google.com/file/d/1v46HgwV4K\_6gcY-Cw89CIKgapb\_\_CWsg/view?usp=sharing**](https://drive.google.com/file/d/1v46HgwV4K_6gcY-Cw89CIKgapb__CWsg/view?usp=sharing)

### **Module 3 results:**

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

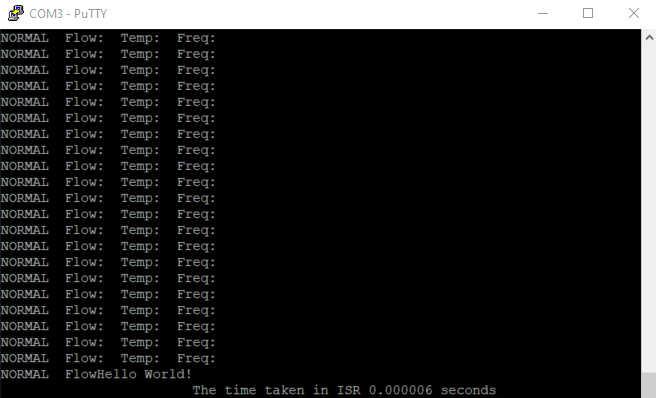
****

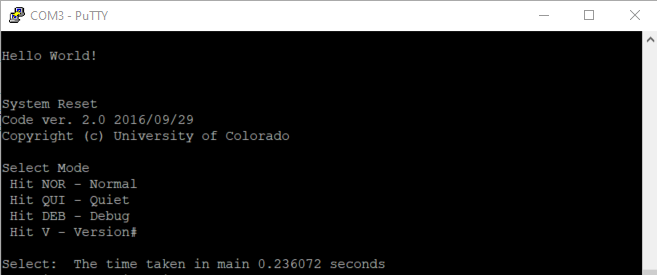
30 sec timer0 count = 37856.

The timer0\_count variable is of 16 bits, this 16 bit gets reset at every 6.5 secs due to reload of count value. So in 30 secs this variable gets reseted 4 times (6.5 x 4 = 26). And in the remaining 4 secs it increments to 37856.

The counter value has slight variation from the number we get when we calculate by converting the frequency to the count value, however this variation is due to the ISR and IO latency.

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

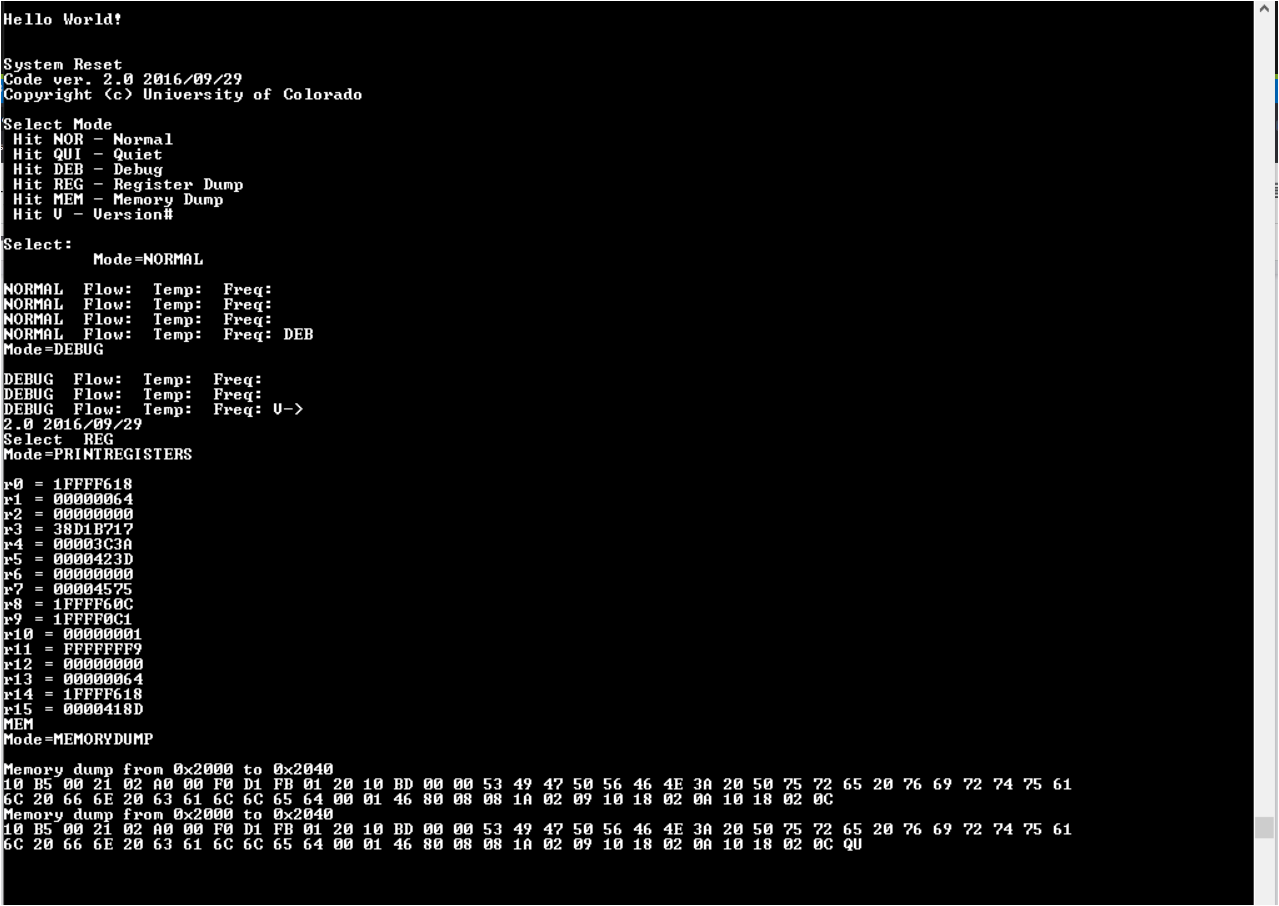
****

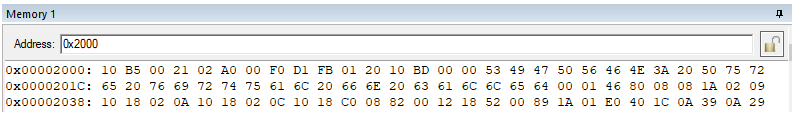
****

The time taken in ISR = 6 microseconds.

The time taken in the main routine is = 0.236 seconds.

**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?**

****

****

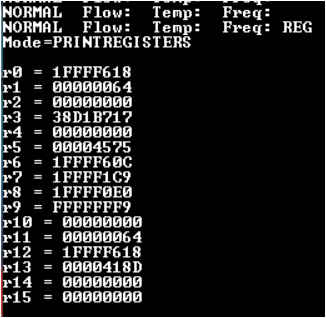
We are not able to read R14 and R15 registers. We believe those are SP and PC registers, which keeps track of the stack pointer and program counter location.

Moreover, we observed that the size of the string we can print depends on the transmit buffer size. So if we transmit a string of length larger than the size of the transmit buffer some of the bytes get overwritten. This gets solved by increasing the size of the transmit buffer. And it is also required to flush the buffer at the end of each string. Similarly for the receive buffer.

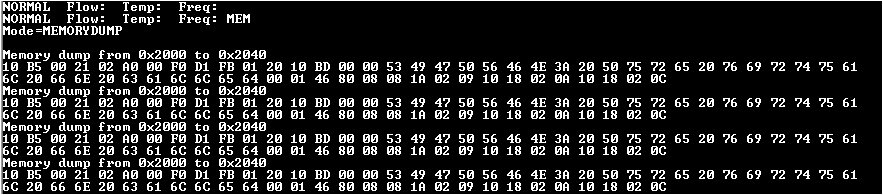
**4. What is the new command you added to the debug menu, and what does it do? Capture a screenshot of the new monitor window.**

We added two new commands to dump registers and memory windows.

Register dump: to get a register dump on the screen, a person needs to enter ‘REG’ and will get R0-R15 registers dumped on the screen. Below is the screenshot of the same.



Memory Dump: We added a feature to read memory dump for 8 bytes. One needs to enter ‘MEM’ to get a memory dump on the screen. Below is the screenshot for the same.



**5. A GPIO pin is driven high at the beginning of the Timer ISR, and low at the end. What purpose could this serve?**

This GPIO pin serves a debug purpose. In our Timer ISR, debugging signal high during Timer0 interrupt on PTB9 indicates entry and exit of execution from ISR. When a GPIO pin goes high, it indicates execution entered ISR and at the end, GPIO pin low indicates we are exiting ISR.

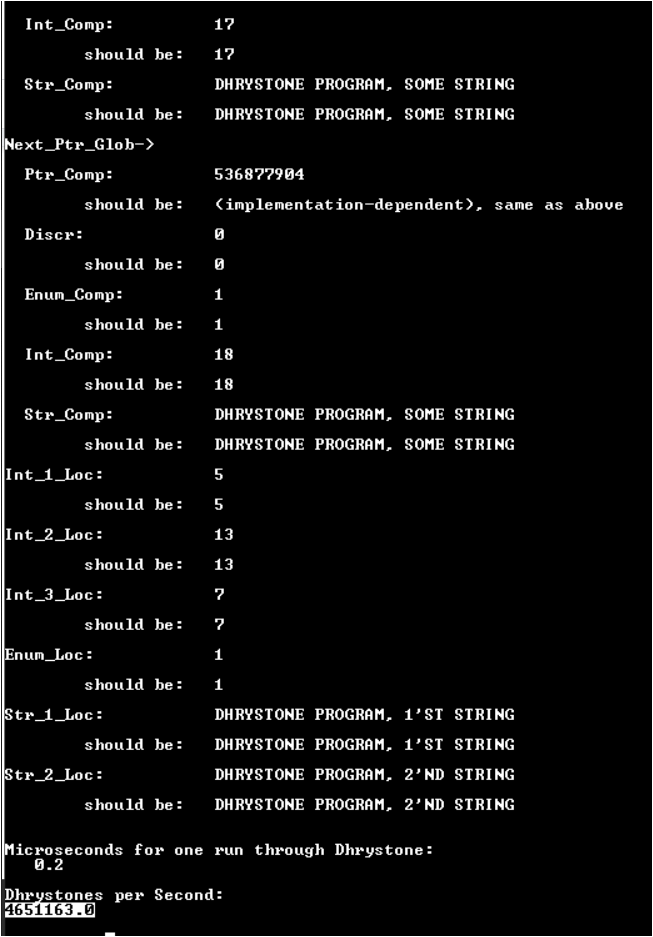
**6. Estimate the % of CPU cycles used for the main background process, assuming a 100millisecond operating cycle.**

CPU time = 0.236 seconds at 100millisecond operating cycle and ISR time is 0.000006 seconds. Timer ISR is set at 100 microseconds and it gets called 1000 times in 100 milliseconds.

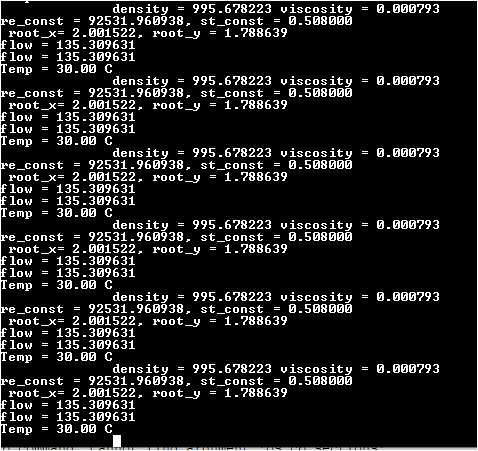
% of CPU cycles used for main background process = 0.236/0.242 (CPU time/ Total time) = 97.52%

**7. What is your DMIPS estimate for the MKL25Z128VLK4 MCU?**

4.651163 DMIPS.



### **Module 4 results:**

****

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

From the zero-crossing frequency detector algorithm we estimated frequency = 40 Hz from provided ADC data.

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

We show the calculated flow = 129 gallons/minute for freq = 40 and temp = 30 degree Celsius.

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

We measured the temperatures in the range of 20 to 33 degree Celsius.

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

We are not using any other interrupt service routines except timer0 to fit the code in given size limitations. Our code is taking 2.048048 seconds for each main loop iteration and ISR time is 0.000006 seconds.

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

At normal 48 MHz frequency CPU consumes 2.048 s in the main foreground process including 2 s wait period. Assuming a 100 millisecond operating cycle 99.99% of CPU cycles used for the main foreground.

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

Full run mode :

Current - 19mA

Power = I^2 \* R = (19mA)^2 \* 10 = 3.61mW

Low power mode:

Current - 12mA

Power = I^2 \* R = (12mA)^2 \* 10 = 1.44mW

Very Low power mode:

Current - 3mA

Power = I^2 \* R = (3mA)^2 \* 10 = 9mW

## 5) List of project deliverables

Below are the files attached in the submitted zip.

1. Doxygen output files for all the Modules
2. Mbed/uVision5 code files

## 6) Recommendations: GO/NoGO

Based on the previous hardware evaluations and analysis, it is NoGo for bare-metal flow meter from our point of view. From the cost, CPU loading and power consumption this product works as expected.

On the opposite side, the reasons for NoGo are as follows.

* According to our experience, we faced program memory issues due to 32k memory limitations. It would be difficult to manage a good flow meter design within the given resource.
* This hardware supports only 32-bit mathematical operations, to have precise flow meter values we need bigger RAM to accommodate all the floating point parameters without it’s value truncated and higher floating point calculating processor.

For this application, we will recommend using TI TMS320 series DSP processors. It will cost the same, but it will provide way more efficient output for the same product.

## 7) References

1. FRDM-KL25Z User's Manual
2. FRDM-KL25Z Pin usage and pinout chart

## KL25P80M48SF0 datasheet

1. KL25 Sub-Family Reference Manual
2. GETTING STARTED WITH ARM USING MBED by Al Williams August 11, 2015
3. <https://www.nxp.com/downloads/en/schematics/FRDM-KL25Z_SCH_REV_E.pdf>
4. <https://groups.google.com/g/comp.arch/c/soXonfwYgdM>

## 8) Project Staffing

**Instructor:**

**Prof. Timothy Scherr**

Teaching Associate Professor

EMBEDDED SYSTEMS ENGINEERING

University of Colorado Boulder

Email: [Timothy.Scherr@colorado.edu](mailto:Timothy.Scherr@colorado.edu)

**Students:**

**Disha Modi**

Graduate Student

University of Colorado Boulder

Email: [disha.modi@colorado.edu](mailto:disha.modi@colorado.edu)

**Swati Kadivar**

Graduate Student

University of Colorado Boulder

Email: [swati.kadivar@colorado.edu](mailto:swati.kadivar@colorado.edu)