PROJECT #1 GUIDE

UCB ECEN 5803 FALL 2021 PROJECT #1: FREESCALE MBED AND KEIL

TABLE OF CONTENTS

INTRODUCTION - MBED

The mbed platform provides free software libraries, hardware designs and online tools for professional rapid prototyping of products based on ARM microcontrollers.

The platform includes a standards-based C/C++ SDK, a microcontroller HDK and supported development boards, an online compiler and online developer collaboration tools.

SOFTWARE DEVELOPMENT KIT (SDK)

The mbed Software Development Kit (SDK) is an open source C/C++ microcontroller software platform relied upon by tens of thousands of developers to build projects fast. We've worried about creating and testing startup code, C runtime, libraries and peripheral APIs, so you can worry about coding the smarts of your next product.

The SDK is licensed under the permissive Apache 2.0 license, so you can use it in both commercial and personal projects with confidence.

The mbed SDK has been designed to provide enough hardware abstraction to be intuitive and concise, yet powerful enough to build complex projects. It is built on the low-level ARM CMSIS APIs, allowing you to code down to the metal if needed. In addition to USB and Networking libraries, a cookbook of hundreds of reusable peripheral and module libraries have been built on top of the SDK by the mbed Developer Community.

HARDWARE DEVELOPMENT KIT (HDK)

The mbed Hardware Development Kit (HDK) provides full microcontroller sub-system design files and firmware for building development boards and custom products that benefit from the native support of the mbed SDK and free mbed Online Compiler and mbed Developer Platform.

The HDK specifies all support components and circuits including the mbed Onboard Interface design that provides simple USB drag-n-drop programming and CMSIS-DAP debug interface for the target microcontroller.

Development boards that are already based on the HDK are the quickest way to get started with the mbed platform. We manufacture official mbed Microcontroller modules that are specifically optimized for flexible rapid prototyping, and are available from distributors worldwide. Our partners are now also creating mbed-enabled hardware such as ultra-low-cost ARM evaluation boards in the popular Arduino form-factor.

- mbed HDK
- mbed Hardware

FREE ONLINE DEVELOPMENT TOOLS

The mbed Compiler is a powerful online IDE that is free for use with hardware implementing the mbed HDK, and tightly integrated with the mbed SDK and Developer Website. Under the hood, it relies on the industry standard ARM professional C/C++ compiler, pre-configured and tested to generate fast, efficient code without fuss.

Login anywhere to get instant access to your development environment, on Windows, Mac, Linux. You can even work from tablets!

Whilst the mbed Compiler provides you your own private workspace, it is also fully integrated with the mbed.org Developer Website so you can easily import libraries and examples. If you choose to, publishing your own code and collaborating with other mbed users is just a few clicks too. The mbed Compiler also supports full export to different toolchains, in case your project demands it as you go to production.

- mbed Compiler
- Collaboration
- Export

WORLDWIDE DEVELOPER COMMUNITY

Using mbed means a huge shared context with other developers, and that means when you have a question, there is less pre-amble, less explanation and less time reproducing issues, and more time getting answers. We're proud that this has helped us grow an active and friendly community of skilled developers that are collectively helping get prototypes made even faster.

But where it really gets interesting is with code. Our developers are sharing thousands of open source repositories and building an extensive cookbook of recipes that you can reuse to build your products.

We've also made contributing back is easy; you can publish a library to mbed.org with a few clicks in the IDE, and let others build on your hard work. In fact, this is how some of our users end up collaborating on hard problems, and even getting contract work.

HAVE FUN!

I. MODULE 1: ASSEMBLY ARITHMETIC

1. From Canvas, download Code2.zip.

2. Follow the Lab_Exercise_2.docx directions.

LAB EXERCISE: SQUARE ROOT APPROXIMATION

OVERVIEW

For this project you will write an assembly code subroutine to approximate the square root of an argument using the bisection method (see following Wikipedia article for details: http://en.wikipedia.org/wiki/Bisection_method). All math is done with integers, so the resulting square root will also be an integer.

HARDWARE

Once you have setup the firmware (see Getting Started document), you simply need to connect the Freedom KL25Z board to your PC via a USB cable, then use Keil MDK to start programming and debugging

In this lab, the program will be loaded to the Freedom KL25Z MCU, and then tested using the debugger in Keil MDK. No other hardware is needed.

REQUIREMENTS

Your code must approximate the square root of an integer between 0 and 2^{31}-1. Using integer math is sufficient (no floating point or fractions are required); your code will return the truncated (integer portion) of the square root.

Your code must be in an assembly language subroutine which is called by a C function for testing. Be sure to use registers according to the ARM calling convention.

SOFTWARE DESIGN

Base your software on the following pseudocode:

```
Approximate square root with bisection method
INPUT: Argument x, endpoint values a, b, such that a < b
OUTPUT: value which differs from sqrt(x) by less than 1

done = 0
a = 0
b = square root of largest possible argument (e.g. ~2^16).
c = -1
do {
        c_old <- c
        c <- (a+b)/2
        if (c*c == x) {
                done = 1
```

```
        } else if (c*c < x) {
                a <- c
        } else {
                b <- c
        }
} while (!done) && (c != c_old)
return c
```

TESTING

In the main function, write code to test that your subroutine works correctly.

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.

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

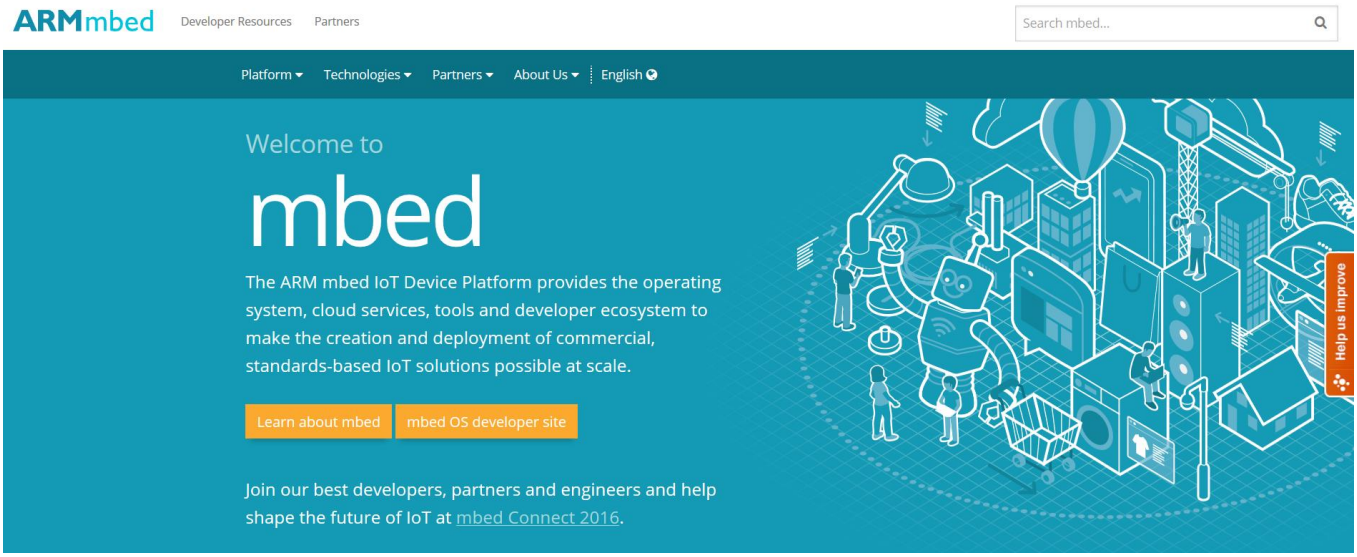
II. MODULE 2: FEEL THE VIBRATIONS

1. Create a program to more fully evaluate the Freescale Kinetis KL25Z MCU for use in a Flowmeter:
   a. Write a program either using the mbed compiler, or in the Keil MDK, to read the accelerometer to sense vibration
   b. Use the "vibration" values to drive a control output. In this case, make the color of the RGB LED change with respect to movement of the board
   c. Read the output from the capacitive touch slider, and use it to dim the LED
2. Estimate the processor load in % of CPU cycles
3. Auto-generate documentation using Doxygen. Provide either an HTML directory or PDF file documenting your codebase.
4. Make a short (less than 1 minute) video showing your board working with the accelerometer input.

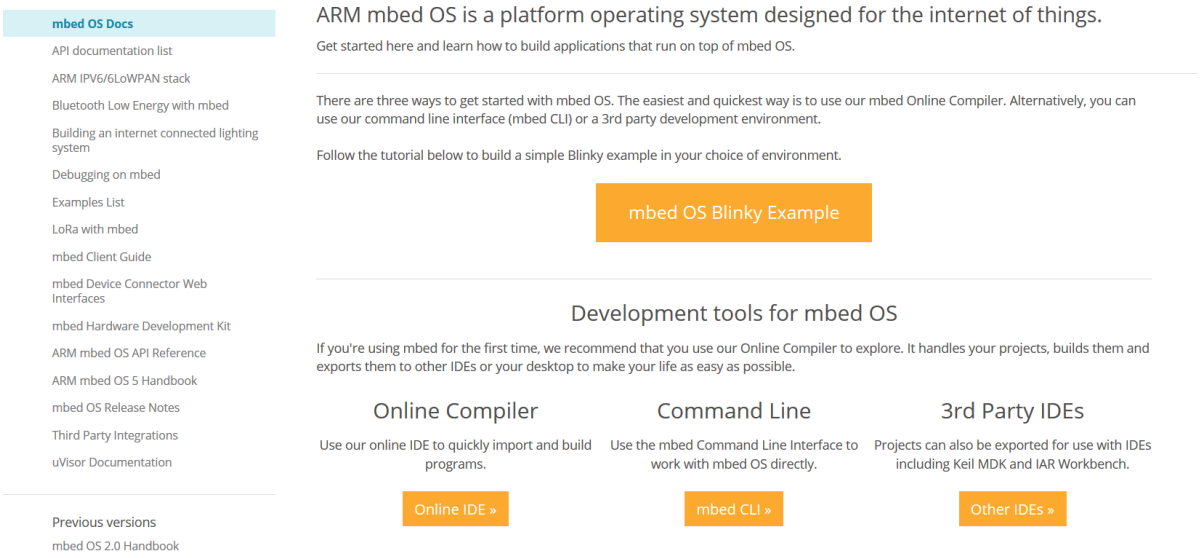
III. MODULE 3: SERIAL PORT DEBUG MONITOR

1. From Canvas, download Code3.zip
2. Follow the Serial Port Debug Monitor instructions:

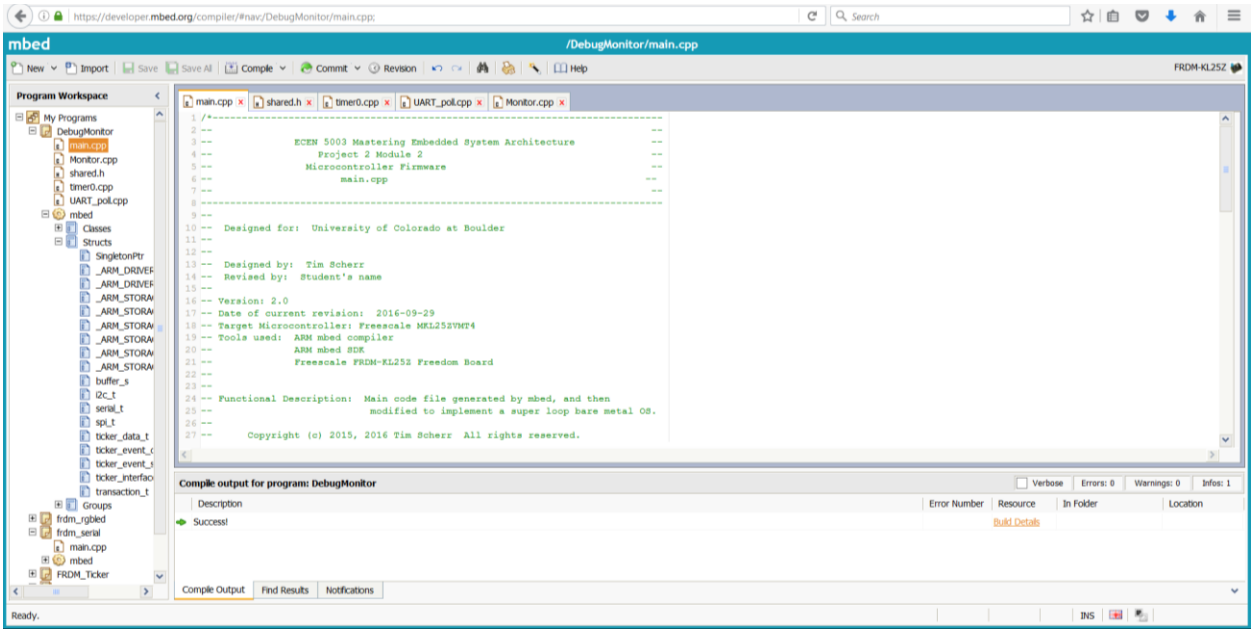
INTRODUCTION



For this Module, we will use the mbed development environment:



You are given a code framework created and then compiled in the mbed environment. You will need to add code to this to complete this module. Although you can import the initial Debug Monitor code to the Keil uVision IDE and develop it from there, use of mBed function calls will create dependencies that are best resolved in the online mBed compiler environment. If you have not done so already, you will need to create an mbed login to get your own dashboard and access to the compiler and libraries. The compiler interface is online, and looks like this:



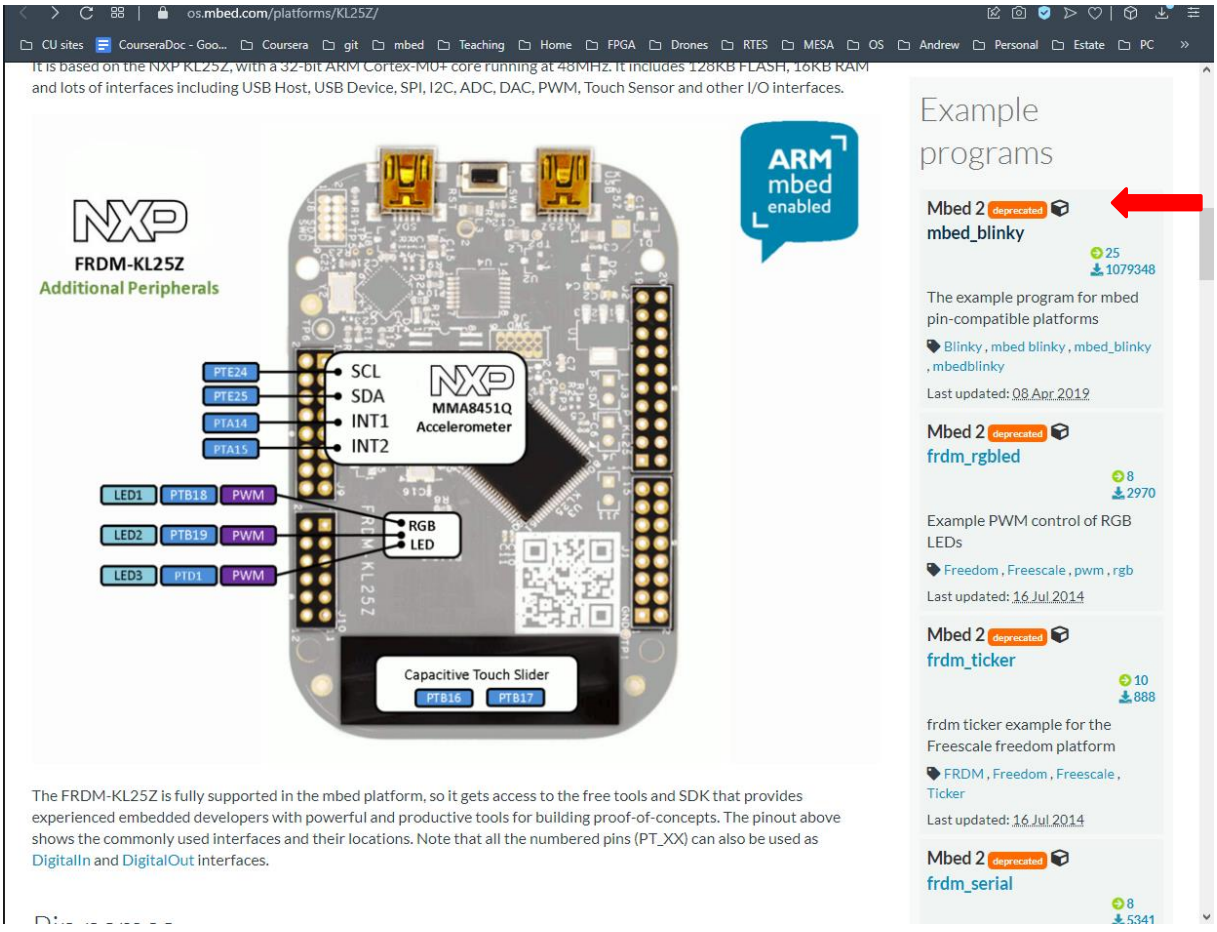
You will start with a project that just has main.cpp, and then add other files to complete the module. Your objective is to build a bare metal OS embedded system with a super loop as the background process, and a timer interrupt as the foreground process. The timer interrupt will allow you to control the allocation of CPU resources between different tasks that run in the background. You will use flags to run tasks in the background process

when needed, and to control the timing of their invocation. As a first step, you will control the timing of a debug monitor display to a terminal window.

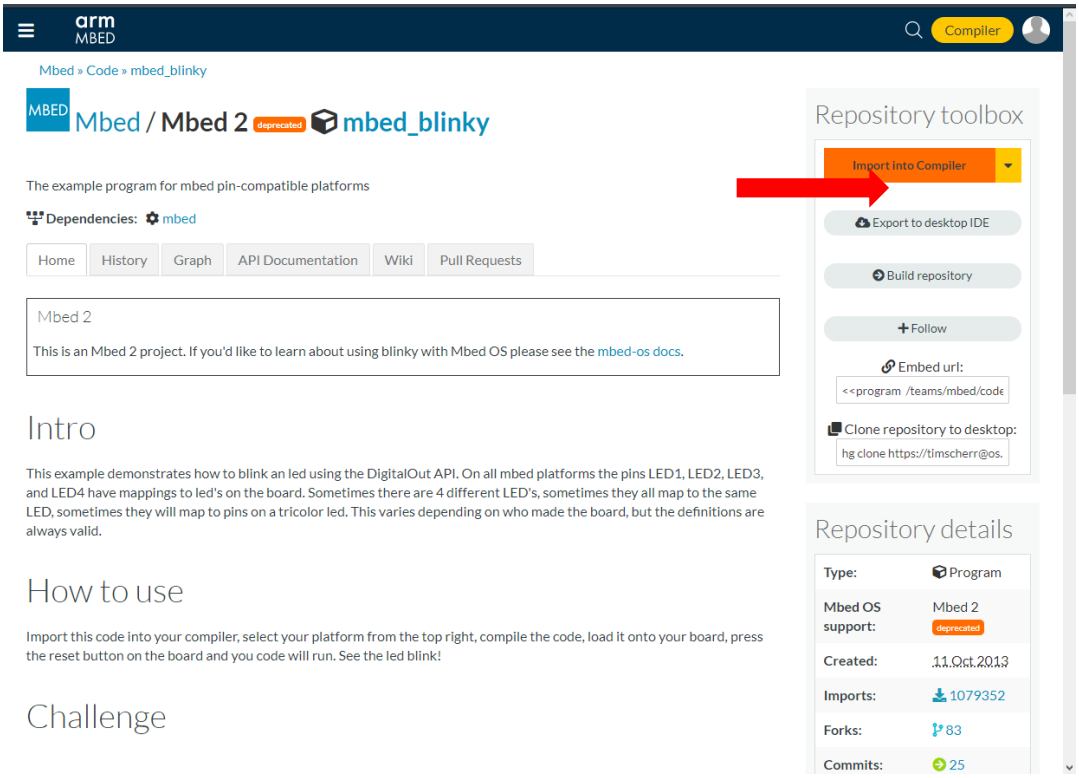
START THE TIMER0 TIMER

STEP 1: ACTIVATE TIMER0

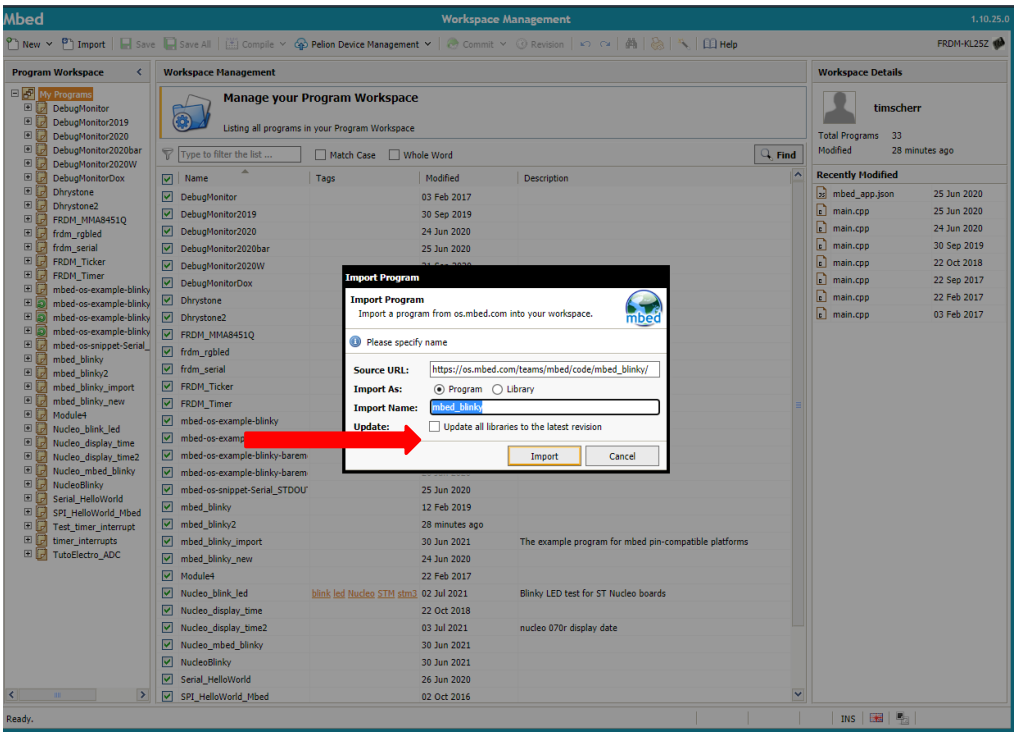
Import the Mbed version 2 mbed_blinky program as a template for the rest of the development. Find this by going to os.mbed.com/platforms/KL25Z/



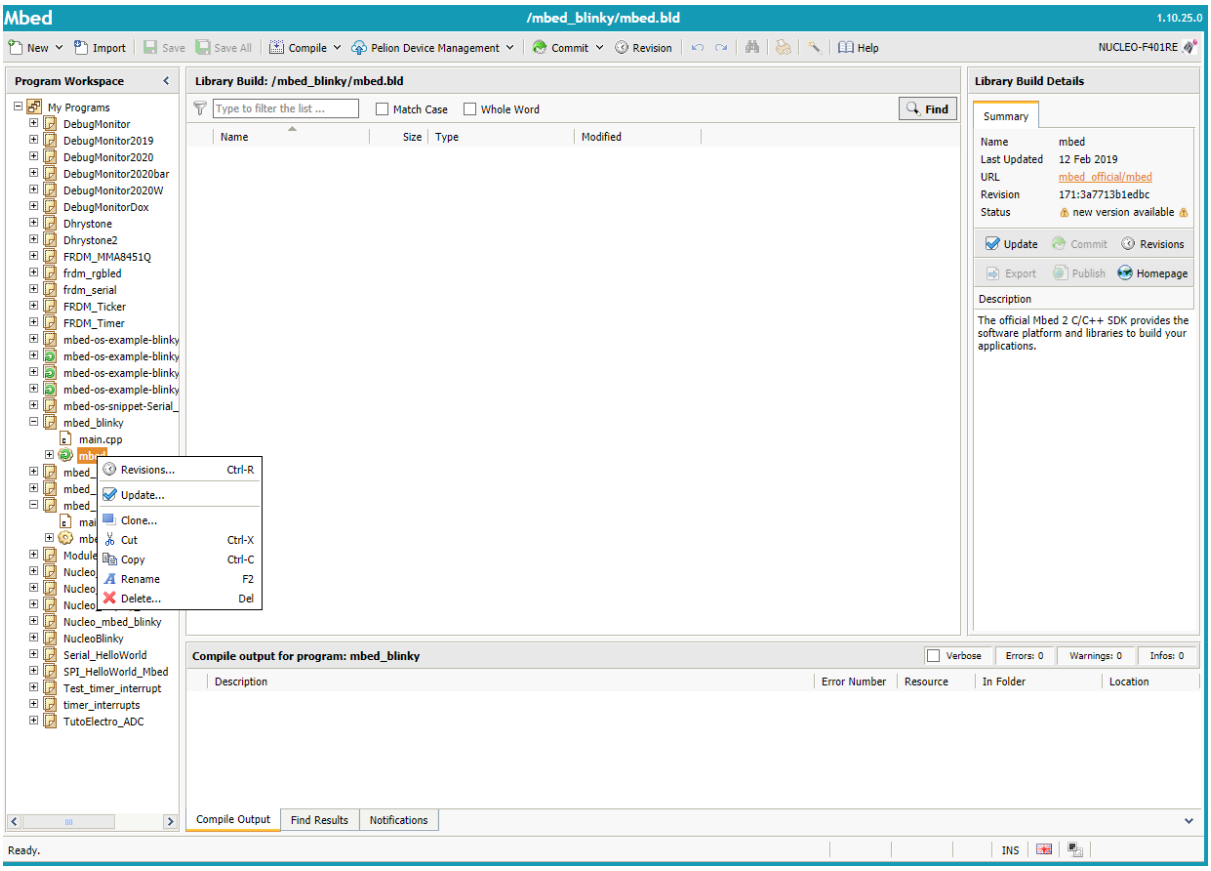
Import this program into your mbed compiler workspace:



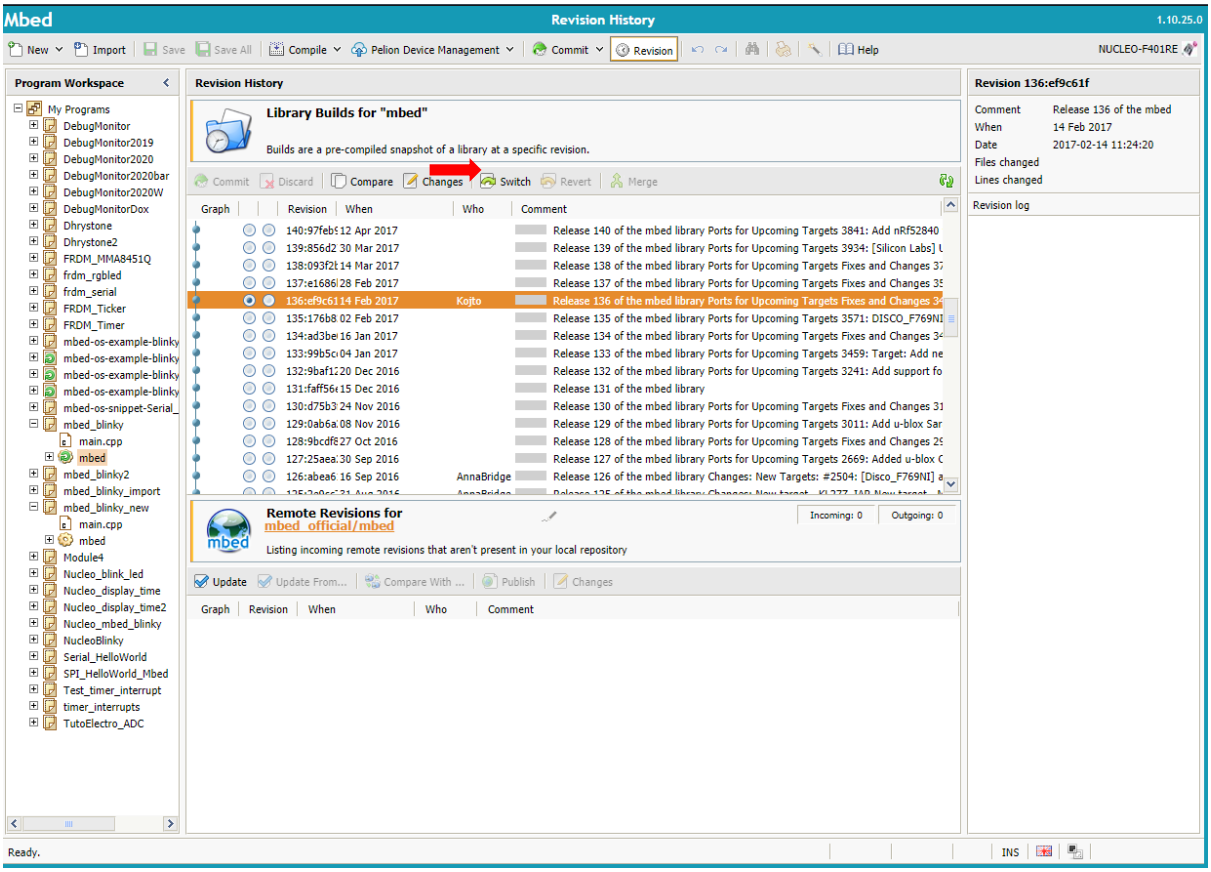
Do not update the libraries



Right click on the mbed library symbol and click on Revisions:



Scroll down and select revision 136, Click on Switch:



To improve your mbed_blinky program, also import the FRDM_Ticker and frdm_helloworld projects. Look at the source code for each project to get ideas about adding a system timer tick and serial port printing function.

Add the system tick timer to your program, and print a message to the serial port.

STEP 2: ADD OTHER MBED COMPONENTS

Add 3 components to control GPIO to the red, green and blue LEDS where indicated in main.cpp. For reference,

RED LED: PORTB, Pin 18
GREEN LED: PORT B, Pin 19
BLUE LED: PORT D, Pin 1

You may try to test the code at this point (optional).

You can download the program directly to the board, or export the program to Keil to be built and then start a debugging session. To export to Keil, see the instructions in “mBedExport.docx” on Canvas. Start a terminal program like putty or Terraterm, and you should see the hello world message if you have chosen the baud rate correctly.

IMPORT SOURCE CODE AND BUILD A DEBUG MONITOR

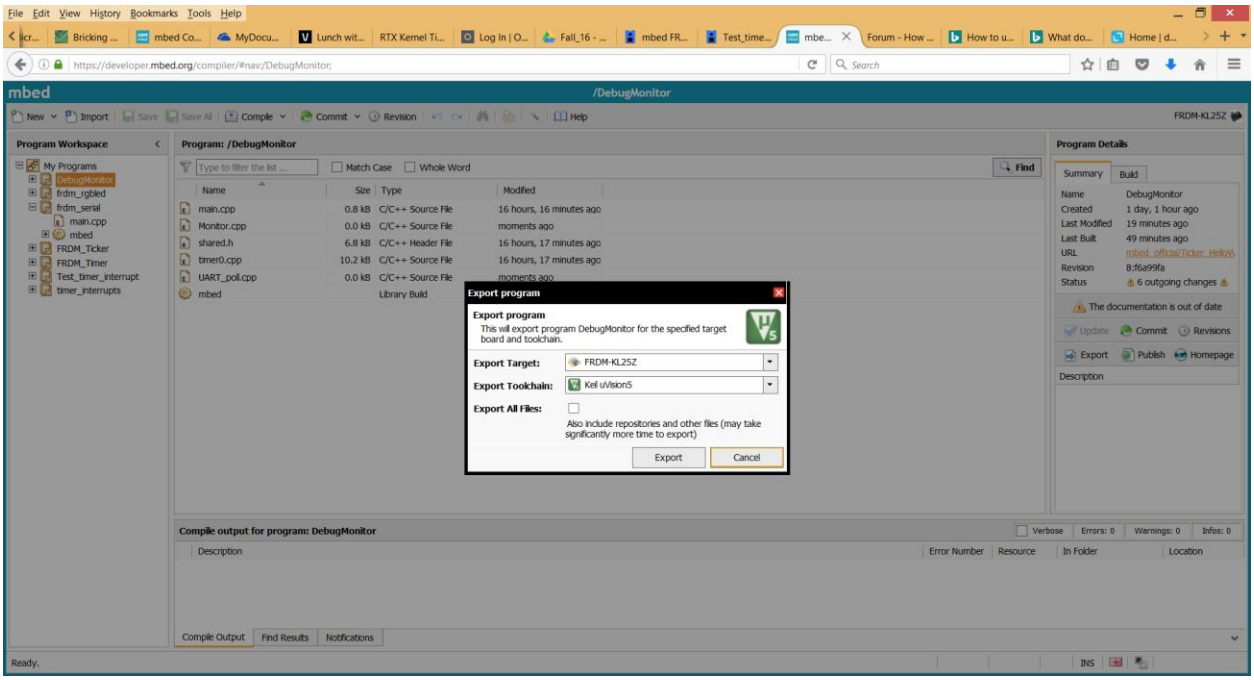
If you haven't already, you will need unzip the files from the Code3.zip file into a source directory for your project. You will move these files from your directory into the mbed online compiler. You may have to create files and cut and paste the information to get all the files created.

You are adding 4 files:

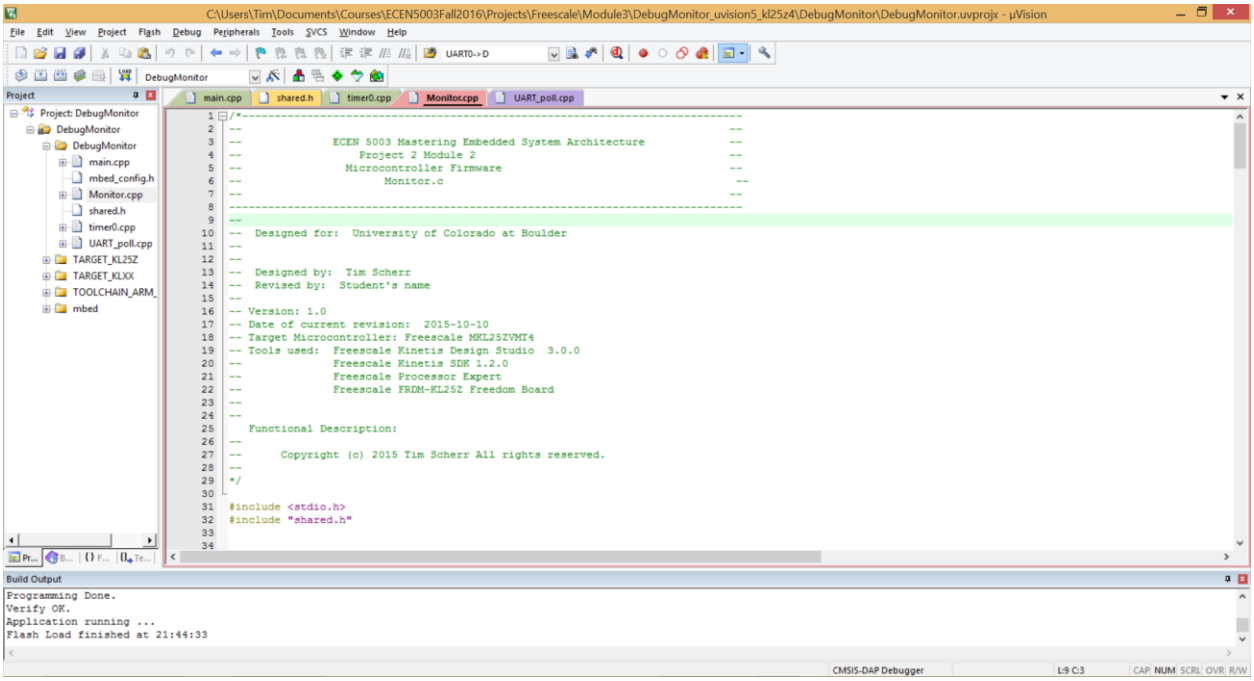
- shared.h, a header file with shared memory variables
- Monitor.cpp, a file with the debugging monitor terminal display code
- UART_poll.cpp, a file with a number of functions to process UART bytes and message strings.
- timer0.cpp, a file with code to be run inside the System Timer Tick interrupt service routine.

You must also replace main.cpp with the code in main.cpp found in Code3.zip. At this point you could rename the project to something more descriptive, like Debug_Monitor.
To insert the code from timer0.c in Code3.zip into the interrupt handler, you must attach the function call to the Ticker ISR as indicated earlier. Use the mbed ticker library function and create a timer than runs the timer0 function once every 100 microseconds where indicated in main.cpp

Confirm that timer0.cpp, shared.h, monitor.cpp, and UART_poll.cpp are included in your source files. Compile this in the mbed compiler, and then right click on the program to export the program for Keil uVision debugging:

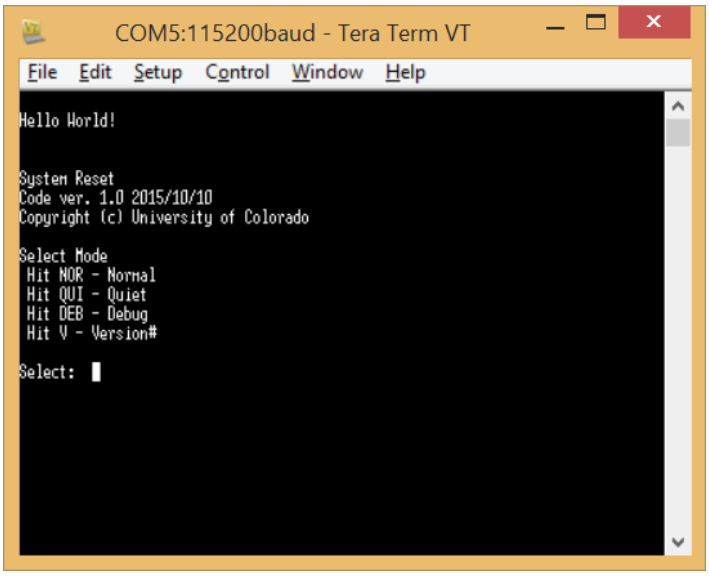


Unzip the file you download from mbed into your source directory. You will see the uVision project file, click on this to import the project into uVision:

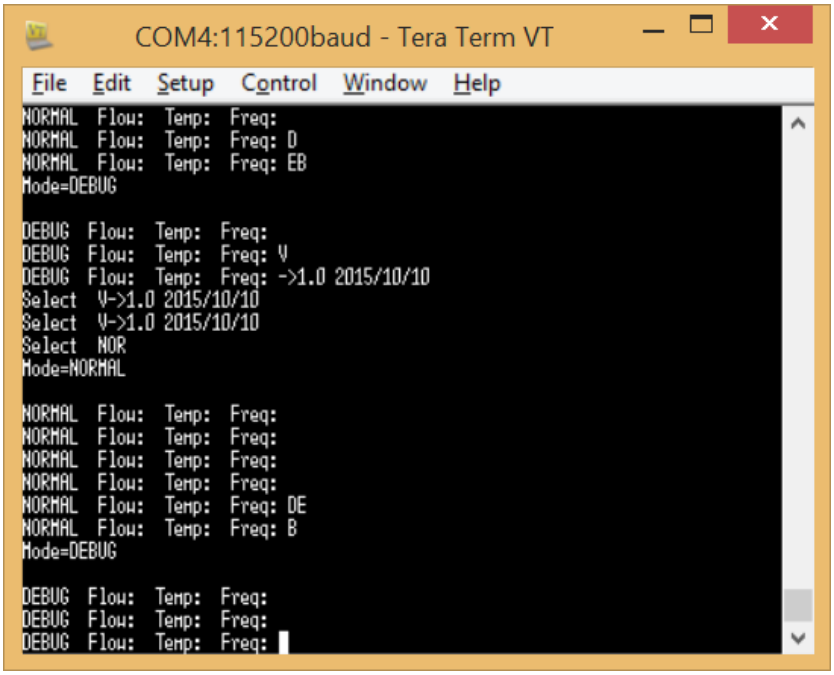


Build this project, and then start a debugging session using your Freedom KL25Z development board.

If you start a terminal program like Terra Term, then you should see



If you enter NOR for normal, or DEB for Debug, or V for Version, you should see something like the following window. However, if you enter QUI, you will see no display at that point. Hitting NOR will resume the data display window.



Congratulations! At this point you have completed building the debug monitor

ADD SOURCE CODE TO THE DEBUG MONITOR AND TIMER ISR

In the code provided, you should see comments to the effect of

// ECEN 5803 add code as indicated
These provide you opportunities to improve the design of the monitor, and to use the timer to blink an LED. Search for the above phrase and you will see some cryptic instructions on what to do, but let the general requirements of the project be your guide.

Improvements you should make:

1. Improve the appearance of the debug monitor display and operation. Provide a personal touch and make screenshots of your results.
2. Add commands to the monitor, including display of the registers R0-R15, and display of section of memory.
3. Insert code using flags in the timer0 function to blink the Red LED with a 1 second period.
4. You do not need to add flow or sensor data to the display just yet, but be thinking about it.

RUN A BENCHMARK

As a separate project, run either the Dhrystone or the Whetstone benchmark program on your target processor using the code provided, which may need to be modified. If running the Dhrystone, calculate the number of VAX DMIPS.

This completes Module 3.

QUESTIONS

Answer these questions in your technical report:

1. What is the count shown in timer0 if you let it run for 30 seconds? Explain why it is this.
2. How much time does the code spend in the main loop versus in Interrupt Service Routines?
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?
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.
5. A GPIO pin is driven high at the beginning of the Timer ISR, and low at the end. What purpose could this serve?
6. Estimate the % of CPU cycles used for the main background process, assuming a 100 millisecond operating cycle.
7. What is your DMIPS estimate for the MKL25Z128VLK4 MCU?

DOCUMENT

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

IV. MODULE 4: BARE METAL FLOWMETER SIMULATION

1. From Canvas download Code4.zip.
2. Write a frequency detection algorithm to determine the vortex frequency from the raw ADC quasi-sine wave samples. Also create code to determine the volumetric flow from the frequency. Once you have completed the code to calculate the volumetric flow, request the simulated ADC data file from your instructor. Each project team data file will be different.
3. Record the reported values of frequency and flow from your monitor program.
4. Estimate the % of CPU cycles used for this process, assuming a 100 millisecond operating cycle.
5. Calculate the power consumption for your complete system (including proposed hardware additions) when in full run mode, and again in low power mode. Include detailed timing assumptions used in each mode.
6. Auto-generate documentation using Doxygen. Provide either an HTML directory or PDF file documenting your codebase.

INTRODUCTION

To calculate flow from frequency and temperature sensors, the embedded system needs to use knowledge of the system model and the physics of the measurement system to convert the sensor measurements into a flow rate. For the case of water, the flow in gallons per minute is

(1) Flow = 2.45*(PID)^2 * v

Where PID is the pipe inner diameter in inches, and velocity is in feet/second.

In Vortex meters, the average fluid velocity is proportional to the frequency of vortex shedding and the width of the bluff body (strut). This proportionality is defined as the Strouhal number, which is dimensionless. Therefore:
St = fd / v, Where: St = Strouhal number
f = frequency of vortex shedding
d = width of bluff body
v = average fluid velocity

The actual width of a bluff body within a specific vortex meter is fixed, therefore, a constant. The frequency of vortex shedding is linearly proportional to the average flowing velocity over a wide range of Reynolds numbers.* Today, most vortex shedding flow meters operate accurately at Reynolds numbers from 10,000 up to 10,000,000.

The vortex shedding flow meter is a volumetric flow meter. Therefore, to define the mathematics of vortex metering, we must first define the following relationships of volumetric flow.

(2) Q = Av

Where: Q = volumetric flow rate
v = average fluid velocity
A = cross sectional area of flow path

If a Strouhal number is substituted for average fluid velocity (" v "), it becomes;

(3) Q = fdA / St where
(4) v = fd/St

Since the Strouhal number, and bluff body width, and the cross sectional area of the flow meter can be considered constants (which is defined collectively as "K"), the equation becomes;

(5) Q = f/K

Similar to other frequency-producing flow meters, such as the turbine meter, this "K" factor can be defined as pulses per unit volume, such as pulses per gallon, pulses per liter, pulses per cubic foot. Therefore, all that is needed is a defined pulse per unit time to indicate the flow rate such as GPM, lpm, or ft/sec.

(6) K = St/dA, where
(7) St = 0.2684 – 1.0356/SQRT(Re), where Re is the Reynolds number.
(8) Re = (Density (in kg/(m^3)) * velocity (m/s) * PID (m)) / viscosity (kg/(m*s))

Density and viscosity of water are a function of temperature

(9) rho(density) = 1000(1 – ((T+288.9414)/(508929.2*(T+68.12963)))*(T-3.9863)^2) in kg/m^3 with T In degrees C.

(10) viscosity = 2.4x 10^-5 * 10^(247.8/(T-140.0) kg/(m*s) = V T is in Kelvin

then velocity becomes a function of only frequency and temperature. We will measure frequency and temperature, and then compute velocity using (4) with St coming from (7) through (10) and **bluff body width=d of 0.5 inches**. **The inside pipe diameter is 2.900 inches**. Then we compute the flow in (1) using the velocity from (4), keeping in mind that the velocity in (4) is in inches/s and will need to be converted. Also not that the PID in (1) is in inches, while in (8) it is in meters.

We will simulate the addition of noise to the frequency measurement by taking ADC samples and adding them to a file of 1000 frequency samples. From this we will compute the frequency for use in (4) above. The algorithm used to compute frequency is at the discretion of the designer (you), but simpler approaches usually work best. You can expect frequencies in the range from 10 Hz to 2000 Hz.

Given the above information, you should:

1. **Design** your algorithm by using a Simulink model in MATLAB.
2. **Test** your algorithm using in Simulink a sine wave generator with added white noise as the test input over 1000 samples, and determine if your algorithm calculates the frequency you expect. The samples are not a series of numbers representing frequency, but rather samples of a sine-like wave with a particular frequency that you need to determine with your algorithm.
3. **Code** the algorithm in C for execution in the microcontroller.

Also required are PWM outputs for the pulse output and to drive the op-amp circuits for the 4-20 mA current loop. A display is also required, but there is no LCD peripheral on the Freescale MKL25Z, so we will assume a SPI port based LCD display.

One approach to save power would be to use the low power ADC mode operating in the background, while the processor went to sleep for 1000 samples, and then woke up to do the flow and temperature calculations. Determine what the power usage of the processor would be during sleep, and during the flow calculation period.

ADD THE ADC, PWM, AND SPI COMPONENTS

STEP 1: ADD THE ADC COMPONENT

From the mbed components Library, add AnalogIn components, with board channel 0 (PTB0) to A/D pin VREFL (internal ADC ch 30), board channel 1 (PTB1) to J10_4 a virtual vortex frequency input (internal ADC ch 9), and board channel 2 (PTB2) to an actual internal TempSensor (internal ADC ch 26). Consult the reference manual for use of the Temperature Sensor.

Next, complete the configuration. Set the configuration for read only configuration, no low power mode, clock divider = 1, long sample time, resolution 8 bit for channel 0 and 16 bit for channels 1 and 2, clock source= bus clock, and all other options use the default, except for continuous mode, which you will enable for channels 1 and

2. To do this you will have to access the ADCx->SC1A, ADCx->CFG1, ADCx->CFG2, and ADCx->SC3 registers. Set the sampling rate to 100 ksps using the equation in Figure 28-62 of the reference manual.

Leave the HW compare configurations and ADC PGA configurations at the default values.

Also note that the ADC in this processor must be calibrated before use. Run the calibration sequence during initialization, and determine if the internal calibration succeeded. The ADC contains a self-calibration function that is required to achieve the specified accuracy. Calibration must be run, or valid calibration values written, after any reset and before a conversion is initiated. The calibration function sets the offset calibration value, the minus-side calibration values, and the plus-side calibration values. The offset calibration value is automatically stored in the ADC offset correction register (OFS), and
the plus-side and minus-side calibration values are automatically stored in the ADC plusside and minus-side calibration registers, CLPx and CLMx. The user must configure the ADC correctly prior to calibration, and must generate the plus-side and minus-side gain calibration results and store them in the ADC plus-side gain register (PG) after the calibration function completes. Prior to calibration, the user must configure the ADC's clock source and frequency, low power configuration, voltage reference selection, sample time, and high speed configuration according to the application's clock source availability and needs. Consult the reference manual for further instructions on how to do the ADC calibration.

STEP 2: ADD OTHER MBED COMPONENTS

For outputs, you need to provide PWMs for the 4-20 current loop and pulse output, and also an interface to the LCD display. We will assume an inexpensive LCD with a SPI interface. Add 2 PWM channels and a SPI channel to drive the outputs. Note this could have also been done using software timers in timer0.c and a function called from main().

Direct the PWMout function to pins PTE30 and PTE31, using Ch3 and Ch4 of TPM0. For the SPI port, use PTC4, PTC5, PTC6, and PTC7.

When you add the SPI component, there will be setup to do in the C code.

Generate the mbed code as before.

IMPORT SOURCE CODE AND CREATE THE FLOW CALCULATION

From CANVAS download Code4.zip. Here you will find revised versions of main.cpp, shared.h, and timer0.cpp. Main.cpp has the stubs for addition of the flow calculations and outputs using the TPM ports. Complete the code to calculate the flow, build it and only then request your frequency input file from the instructor by email. Once you have the file, transfer it to the target and combine it with your ADC temperature data, calculate the flow using the code running in the target, and record the results. Generate the 4-20 output using TPM0 channel PWM with a rate proportional to the flow. Generate the Pulse output() using TPM0 channel 4 PWM with a rate proportional vortex frequency. Write SPI commands to send the flow rate data to the LCD. Flow, temperature, and frequency data should also be displayed on the Monitor output in the terminal window.

This completes Module 4.

QUESTIONS

Answer these questions in your technical report:

1. What is the frequency estimate from your provided sample ADC data?
2. What is the calculated flow you see from your input?
3. What is the range of temperatures you measured with your embedded system?
4. How much time does the code spend in the main loop versus in Interrupt Service Routines?
5. Estimate the % of CPU cycles used for the main foreground process, assuming a 100 millisecond operating cycle.
6. Calculate the power consumption for your complete system (including proposed hardware additions) when in full run mode, and again in low power mode.