Embedded Systems Evaluation – Detailed Technical Report

Customer:	Keithley
Project Name:	2017 Signal Analyzer
Prepared by:	David Tracy, Joyce Cho, Srivishnu Alvakonda
Preparation Date:	November 10th, 2017

TABLE OF CONTENTS

CONTENTS

EXECUTIVE SUMMARY

The following table highlights the critical information that was found during the analysis of the STM32F401RE platform, and is the data used in making a final recommendation on whether Keithley should continue with this embedded platform for their new 2017 Signal Analyzer product.

Analysis Type	Results	Score (1 - 10)
CPU Performance	The Dhrystone Benchmark v2.1 was performed on the platform, with a best-case score of **103 DMIPS**. This is beyond the specification of 100 DMIPS by 3%.	10
Application performance	While analyzing the Signal Analyzer application, the overall CPU usage was around **37%** for both generating and analyzing waveforms simultaneously. This leaves **63%** of the CPU bandwidth idle, which may not be enough for other applications outside the scope of analysis (LAN and USB communication, simultaneous analysis of multiple high-speed ADC channels, etc.)	6
Available Memory	The Signal Analyzer application consumed a large amount of flash and RAM due to the complex nature of the computations involved: **85.5kB (16.7%)** of flash, and **52.8kB (55%)** of RAM. More RAM would be convenient for a signal analyzer to consume more data per processing cycle.	6
DAC	The STM32F401RE does not have a built-in DAC, thus an external DAC must be purchased.	0
ADC	The performance of the built-in ADCs does not meet the speed and resolution requirements of the specification (20-bit @ 192kSps). However they are good enough for lower-speed, lower resolution inputs like temperature measurement.	5
Communication I/O	The STM32F401RE does have considerable I/O (UART, SPI, I2C, I2S, and USB 2.0 are all available and required for product operation). There is no Ethernet peripheral for LAN connectivity.	8
Available Pins	There are ~40 GPIO pins used for switches, buttons, LEDs, and relays. Another ~10-15 need to be used for UART, I2S, SPI, and USB. If driving the VGA display directly, that can consume 24 lines, depending on the technology. Using a **64-pins** of the STM32F401RE is very marginal and is likely to not be enough for this application.	4
MCU Cost	The processor cost is **$2.99** at volumes of 10000, which meets the $3 MCU cost target.	10
System Cost	Overall proposed system cost is $58; the cost target is $20.	6

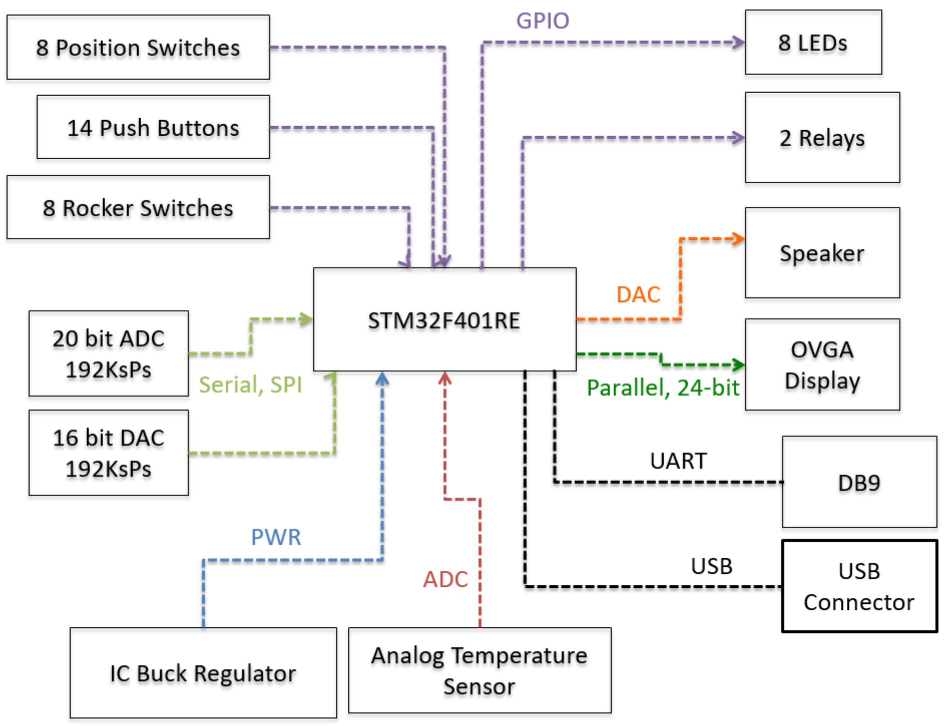
Conclusion: The final score assigned to the STM32F401RE processor (specifically the STM32F401RET6 variant) and overall proposed system design is 6.1/10. This part is not suitable for use in the Keithley 2017 Signal Analyzer, and gets an official **NO-GO** from the project team. Please see the "Recommendations" section for possible alternative solutions.

PROBLEM STATEMENT AND OBJECTIVES

The project team has the explicit objective of evaluating the hardware and software capabilities of the STM32F401RE MCU, with reference to the required specifications by Keithley, using various hardware and software test tools and methodologies. Of particular focus will be evaluation of the DSP capabilities of the processor, as well as the capability to interface between many different devices, via many different peripherals (ADC, DAC, UART, SPI, GPIO, I2S, etc.), some interfaces being very high speed for signal processing and conditioning.

APPROACH AND METHODOLOGY OF EVALUTION

System Block Diagram



The System Block Diagram shows all major subsystem components that are called out as requirements in the RFS. In general, arrows indicate if a subsystem is an *input* or an *output.* Refer to the supplemental Bill of Materials (***BOM_Project2.xlsx***) for proposed part numbers for each subsystem component.

Summary of Approach

In Module 1, the on-board Real-time clock of ST Nucleo was used to display current time on a UART serial terminal. The online Mbed compiler was used to make and compile the code. The RTC can be configured by setting the time in seconds starting from January 1, 1970.

In Module 2, the external peripherals were interfaced with corresponding protocols.

a. Lab_exercise_2_8: Programming Using Mbed API

   LEDs and buttons were interfaced using GPIO pins. The mbed DigitalIn library is used to read the value of a digital input pin. The mbed BusIn library is used to create a number of DigitalIn pins that can be read as one value. The InterruptIn is used to trigger an event when a digital input pin changes.

b. Lab_exercise_2_9: Analog Input and Output

Potentiometers and speakers were interfaced using ADC and PWM. The PWM output is used to generate electrical waves which can be turned into sound by the speaker.

c. Lab_exercise_2_11: Serial Communication

Three types of serial communication are used to interface with three peripherals. The 16x2 LCD using ST7066 controller was interfaced using SPI with the 74HC595N shift register. Also, the DS1631 temperature sensor used I2C protocol and the results were displayed on UART terminal.

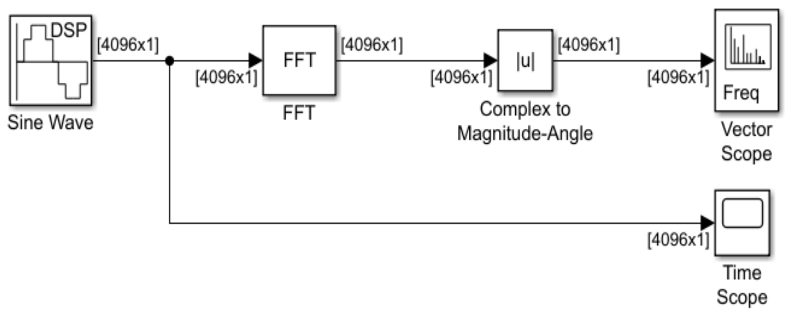
In Module 3, the mBed RTOS was used to evaluate the capabilities of the STM32F401RE to run a multi-threaded environment interfacing to many different external devices. Threads, Mutexes, and semaphores were used to properly evaluate the multi-threaded performance. See *Module 3* section for further details.

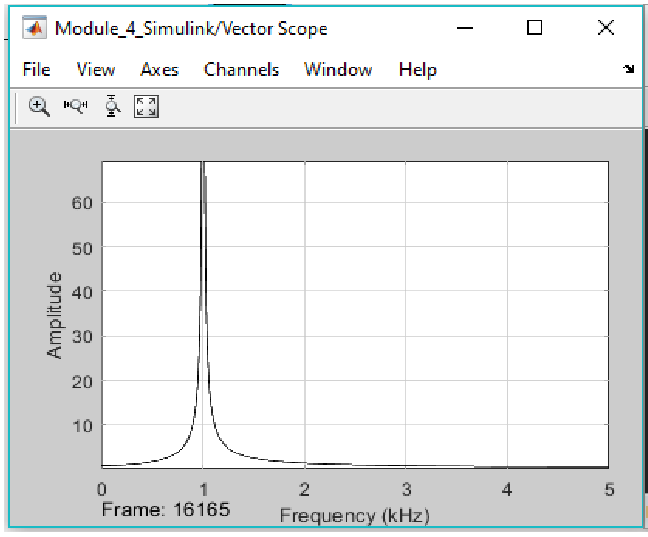
In Module 4, the RTX RTOS, along with ARM DSP libraries and STM32 vendor libraries were used to evaluate the DSP capabilities of the STM32F401RE, by both generating a sinusoidal waveform at a fixed frequency, as well as detecting both sinusoidal and non-sinusoidal waveforms across a range of frequencies. See *Module 4* section for further details.

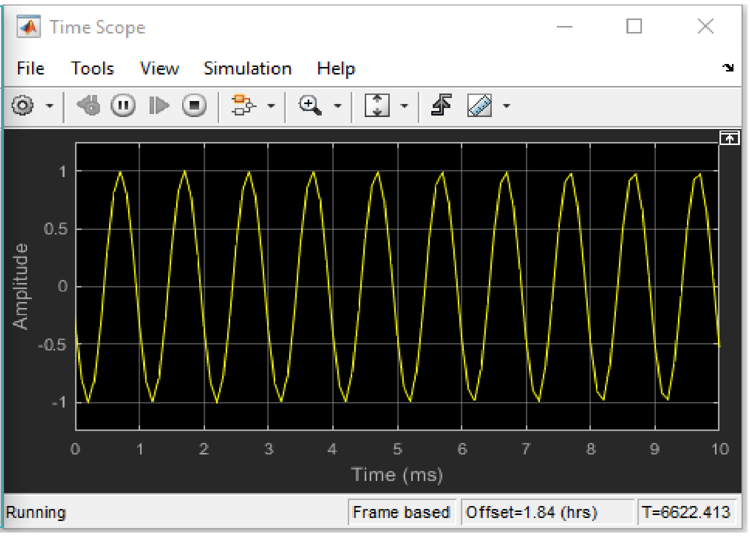
Harmonic Analysis Algorithm

A simple model was created to show the generation of sin samples at the designed 10 kHz sampling rate, as well as the 4096 sample FFT and complex magnitude computation that is used in the MCU. These details are discussed further in the *Module 4* section.

Module 4 Simulink Simulation
- Sine wave generator creates samples at 10kHz
- FFT takes in 4096 time domain samples to process the spectrum
- Complex to Magnitude angle computes the magnitude of the complex FFT output
- Vector Scope plots the frequency domain; Time Scope plots the original signal







MODULE TEST RESULTS

Module 1 – Hello World, What Time is it?

Q1: Where (at what address) does the Reset handler begin in the memory map?

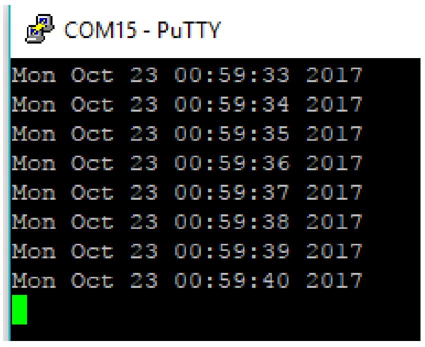
A1: The Reset Vector is at 0x80002E4 -> Refer to figure /Module_1/Pictures/Mod1.1.PNG

Q2: How much memory is used by the code (Led blinking code for the homework)?

A2: The code uses 18.7 Kb of Flash Memory and 1.1Kb of RAM

-> Refer to figure /Module_1/Pictures/Mod1.2.PNG

Q3: Run the mBed Nucleo Example (display_time). Set the time to the current time, and combine this with you're the mBed Nucleo Example (printf) to print the current time to a terminal window on your PC. Capture a screenshot of the terminal window. How much memory is used by this code?



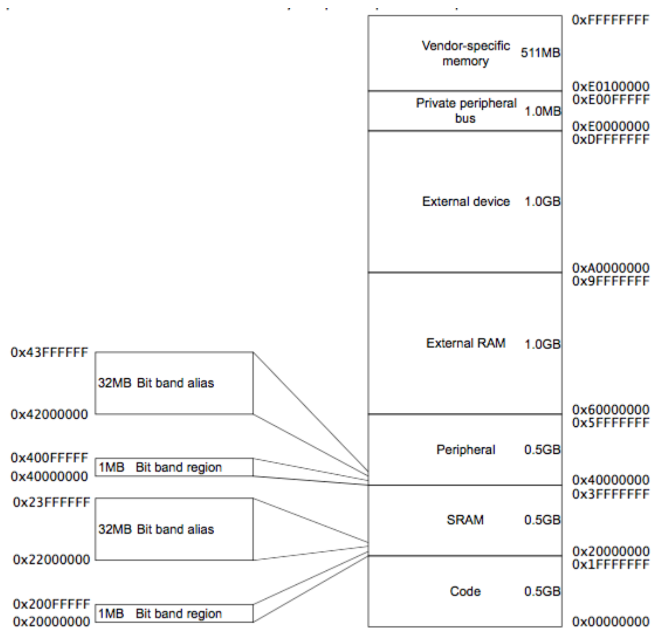
```
Mon Oct 23 00:59:33 2017
Mon Oct 23 00:59:34 2017
Mon Oct 23 00:59:35 2017
Mon Oct 23 00:59:36 2017
Mon Oct 23 00:59:37 2017
Mon Oct 23 00:59:38 2017
Mon Oct 23 00:59:39 2017
Mon Oct 23 00:59:40 2017
```

A3: The code uses 23.00 Kb of Flash Memory and 1.7Kb of RAM

Q4: Explain the memory model of ARM Cortex-M4 with respect to the code memory, data memory, IRQ handlers and peripherals. Explain with the help of a diagram where required.

A4: The Cortex-M4 have a predefined memory map. This allows the built-in peripherals, such as the interrupt controller and the debug components, to be accessed by simple memory access instructions.

The processor has a fixed default memory map that provides up to 4GB of addressable memory. Overall, the 4 GB memory space can be divided into ranges as shown in picture above. The Cortex-M4 design has an internal bus infrastructure optimized for this memory usage. Out of the 4 GB memory space, 512 MB is used by code region to store the program code, 512 MB is used by SRAM region for storage of data and peripheral memory uses 512 MB for peripherals like AHB.



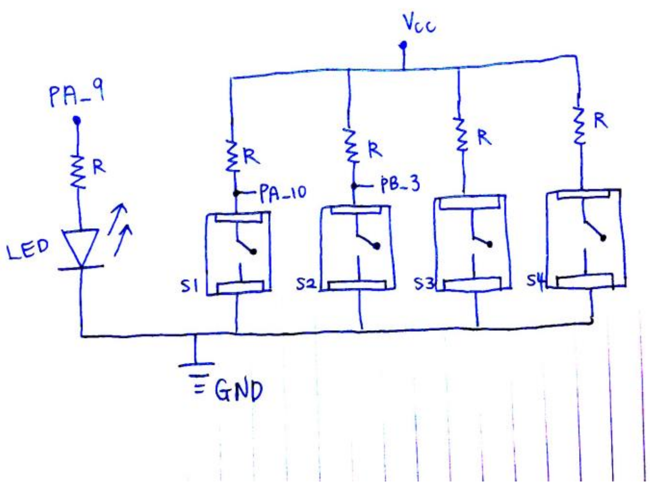
Above: general memory map of Corex-M4

Module 2 – Button, ADC read, LED PWM, and UART

Q1: Try to issue an interrupt on different signal edges (rising edge or falling edge). What changes?

A1: On issuing the interrupt on falling edge, the LED changed state on pressing the button whereas for rising edge the LED changed state after releasing the pressed button.

Q2: Draw a schematic of your final circuit (nothing fancy, can be done by hand and scanned).



Q3: What changes when you adjust the amount by which variable *i* is incremented/decremented?

A3: Variable *i* is used to determine the duty cycle of PWM, thus determining the volume of the speaker. That is, when we increase the value of *i*, the volume of speaker would be turned up. On the other hand, when we decrease the value of *i*, the volume of speaker would be turned down.

Module 3 – RTOS Threads

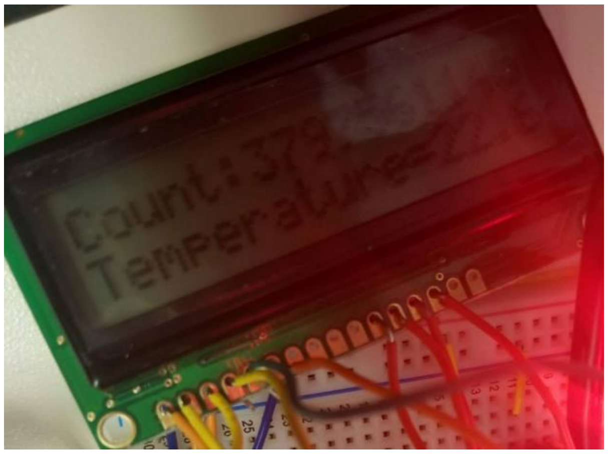
The Keil RTX RTOS was used to schedule 4 threads performing 4 different operations: (1) displaying results from a temperature sensor on the LCD screen, (2) changing the brightness of an external LED, (3) blinking the on-board LED, and (4) displaying a counter on the LCD screen.

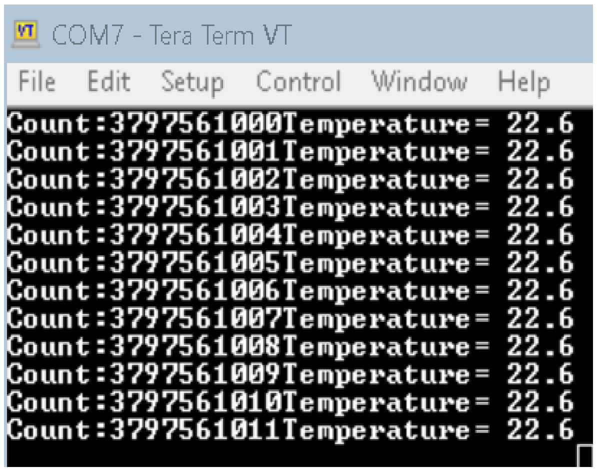
Scheduling of these threads was done on the basis of priority. Since both the temperature thread and counter thread use the same physical LCD resource for displaying content, a mutex was used to prevent conflicts among them (that is, if the temperature thread was currently controlling the LCD, the temperature thread was forced to wait until the resource was unlocked, and vice versa).

The below screenshot illustrates the printouts of both temperature and counter values onto the LCD screen.

Count Value: 3797561145. Temperature Reading: 22.6 degrees C.

In addition to the LCD print out, the count and temperature data is transmitted over the Debug UART.





```
Count:3797561000Temperature= 22.6
Count:3797561001Temperature= 22.6
Count:3797561002Temperature= 22.6
Count:3797561003Temperature= 22.6
Count:3797561004Temperature= 22.6
Count:3797561005Temperature= 22.6
Count:3797561006Temperature= 22.6
Count:3797561007Temperature= 22.6
Count:3797561008Temperature= 22.6
Count:3797561009Temperature= 22.6
Count:3797561010Temperature= 22.6
Count:3797561011Temperature= 22.6
```

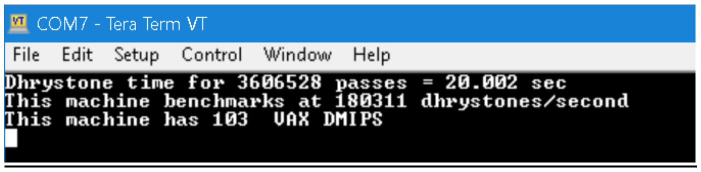
The below table illustrates the basic structure of the Threads, Semaphores, and Mutexes used in the module:

		Semaphores for state control		
#	Threads	Thread gated by:	Thread releases:	Mutexes for shared resources
1	Temperature	Blink Temperature	Temperature Potentiometer	LCD
2	LED Brightness Adjust	Temperature Potentiometer	Potentiometer Count	-
3	Increment Counter	Potentiometer Count	Count Blink	LCD
4	Blink LED	Count Blink	Blink Temperature	-

Each thread is gated by the preceding thread (in the case of Thread1, it is gated by Thread4, and is kicked off by main() during program initialization), and will not be released until the current thread has finished executing.

Dhrystone Benchmark

The results for the Dhrystone benchmarking tests are printed out in the terminal below:



```
Dhrystone time for 3606528 passes  = 20.002 sec
This machine benchmarks at 180311 dhrystones/second
This machine has 103  VAX DMIPS
```

The Dhrystone code is configured to run for 20 seconds per loop, as per the ARM recommendation for benchmarking requirements. The code is modified from the original Dhrystone v2.1 to accommodate the STM32F401 platform. The mBed Timer module is used for keeping track of time.

The target DMIPS for this product is 100 DMIPS. **The test results exceed this performance by 3%**, and is very close to the published value of 105 DMIPS (results may deviate from manufacturer due to limits on optimizations the evaluation team can make).

Module 4 – Signal Analyzer

This module was completed with the aid of several vendor-supplied software and components, as well as several key hardware tools:

1. Reference ARM CMSIS RTOS (RTX) was used as the underlying operating system.
2. STM32 HAL Library, and CubeMX software generator, were used for simplifying the interface between the user application and the hardware layer.
3. ARM DSP math library was used for trigonometric functions (ex. sin computation), and for the CFFT (complex fast Fourier transform) functions.
4. Saleae Logic 8 Logic Analyzer, for timing measurements and UART protocol decoding.
5. Digilent Analog Discovery 2, for sin function generation, oscilloscope and power supply functionality.

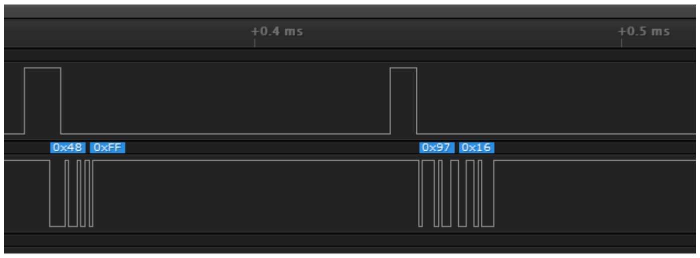
Part 1a. Creating a 1004 Hz tone through the DAC.

A sample generation rate of approximately 10 kHz was chosen for this evaluation. A periodic timer was configured to fire every 100us, setup through the RTOS Timer module. By default RTX is configured to allow no faster than 1ms timer events. The OS_CLOCK macro was redefined to allow a 10x increase in resolution (though timekeeping is off by 10x).

Within the 100us Timer Callback, the DSP math library "sin" function was used to perform optimized sin computations.

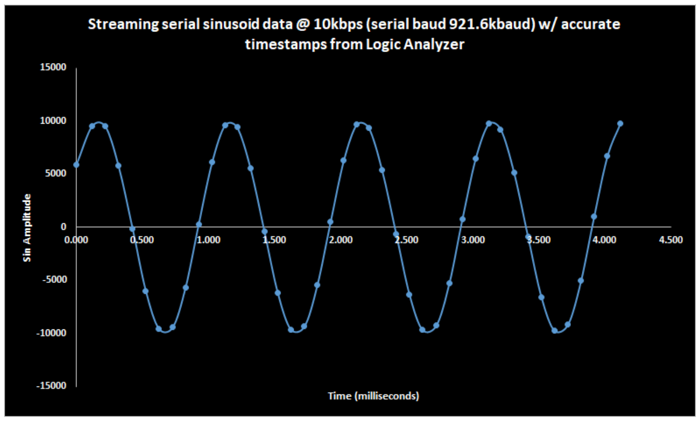
Method	Time	CPU Utilization
Standard Library (math.h)	>100us	>100%
Open Source sin estimation to 5.2 digits (trig_approx.h)	32.5us	32.50%
ARM DSP Math Library (arm_math.h)	7.1us	7.10%

The sin values were converted into a signed 16-bit number to allow for compact data transmission over the UART bus. The baud rate was configured to a very high speed (921.6 kbps) in order to fully transmit the data prior to the next sampling interval.

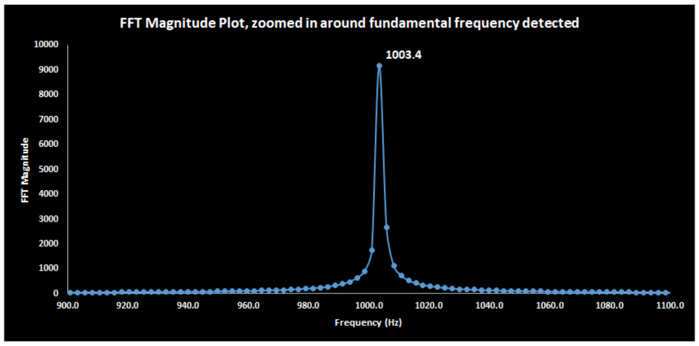


Example timing. Channel 1 high while Timer routine is active (sin computation). An interrupt-driven UART TX is initiated following the computation (2-byte little-endian signed data shown): 0xFF48 = -184, 0x1697 = 5783). Range is constrained to -10000 to +10000.

Sufficient samples were captured, saved to a file, and post-processed to achieve the following plots proving that the 1004 Hz desired output was achieved:



FFT performed in MS Excel shows fundamental frequency very near 1004 Hz:



For more information please see ***Time_and_Frequency_Domain_Analysis.xlsx.***

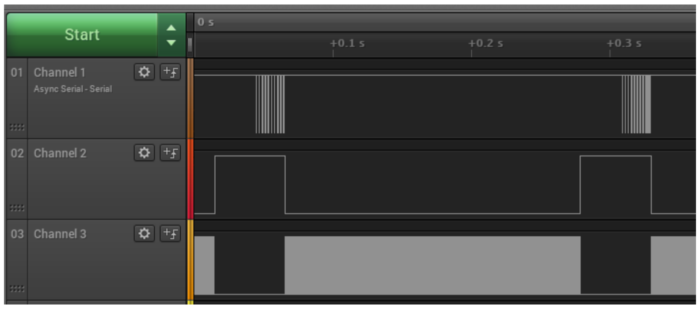
Part 1b. Reading a 1004 Hz tone (and other arbitrary frequencies) through an ADC and performing a harmonic analysis of the data.

Since a sine function generator was available, this evaluation was performed using an actual ADC, as opposed to a simulated ADC reading from an I/O stream (file or serial port). The general architecture for this evaluation is outlined below:

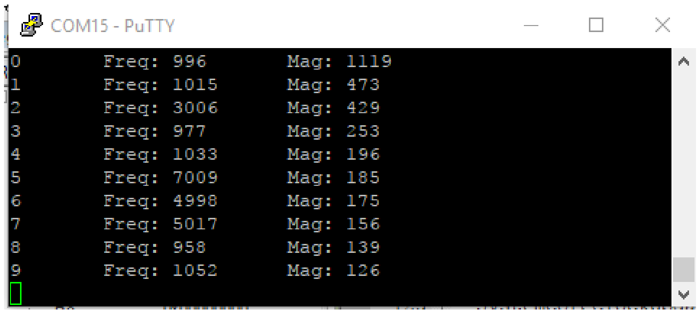
1. Configure ADC in continuous mode (takes samples at fixed frequency forever). Sample rate configured at 20 kSps (12-bit data resolution). Enable the ADC interrupt to process and store the ADC data into a buffer (4096 samples). The data is normalized to values between (-1.0, +1.0) as this aids in harmonic analysis results.
2. After 4096 samples are collected, this sends an OS Signal, which will unblock the main harmonic analysis task, which will analyze the newly collected ADC samples.
3. The harmonic analysis task (1) computes the CFFT on the normalized ADC samples, (2) takes the complex magnitude of the results and stores that in a secondary buffer, and (3)

prints over the UART the "Top 10" frequencies it detected in descending order.

a. An alternate compilation prints out the entire magnitude array, which is stored in a file and post-processed to visually show the FFT results.



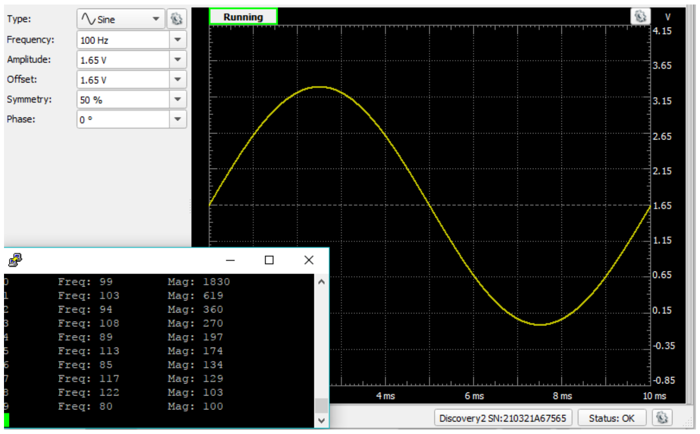
ADC samples collected on CH3. Harmonic Analysis task active while CH2 is high. "Top 10 frequencies" serial data on CH1 (transmitted in blocking-mode during harmonic analysis).



```
0       Freq: 996       Mag: 1119
1       Freq: 1015      Mag: 473
2       Freq: 3006      Mag: 429
3       Freq: 977       Mag: 253
4       Freq: 1033      Mag: 196
5       Freq: 7009      Mag: 185
6       Freq: 4998      Mag: 175
7       Freq: 5017      Mag: 156
8       Freq: 958       Mag: 139
9       Freq: 1052      Mag: 126
```

Screenshot of serial terminal after frequencies are printed for a sinusoidal input source of 1004 Hz. Due to the combination of bin size of the FFT, and sampling rate of the ADC, the frequencies have some rounding error.

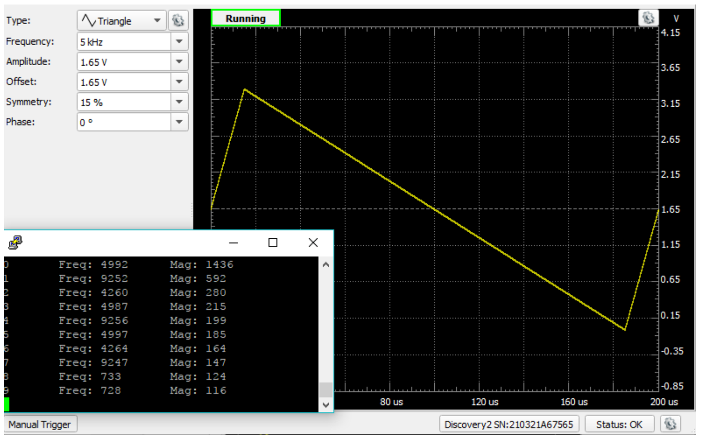
$f_i = (f_{sampling} * i)/N$, where $f_{sampling} \sim 20$ kHz and N is the FFT bin size (4096). By reducing $f_{sampling}$ or increasing N, better resolution around the frequencies of interest are attainable.



Type: Sine
Frequency: 100 Hz
Amplitude: 1.65 V
Offset: 1.65 V
Symmetry: 50 %
Phase: 0 °

```
0       Freq: 99        Mag: 1830
1       Freq: 103       Mag: 619
2       Freq: 94        Mag: 360
3       Freq: 108       Mag: 270
4       Freq: 89        Mag: 197
5       Freq: 113       Mag: 174
6       Freq: 85        Mag: 134
7       Freq: 117       Mag: 129
8       Freq: 122       Mag: 103
9       Freq: 80        Mag: 100
```

The same application (without any code changes) can be used to analyze a broad range of frequencies. In the example above we are correctly detecting the fundamental frequency of a 100 Hz sinusoid. The application can also detect non-sinusoidal inputs (ex a 5000 Hz asymmetrical triangle wave):



Type: Triangle
Frequency: 5 kHz
Amplitude: 1.65 V
Offset: 1.65 V
Symmetry: 15 %
Phase: 0 °

```
0       Freq: 4992      Mag: 1436
1       Freq: 9252      Mag: 592
2       Freq: 4260      Mag: 280
3       Freq: 4987      Mag: 215
4       Freq: 9256      Mag: 199
5       Freq: 4997      Mag: 185
6       Freq: 4264      Mag: 164
7       Freq: 9247      Mag: 147
8       Freq: 733       Mag: 124
9       Freq: 728       Mag: 116
```

Part 1c. Blink the User LED at a rate proportional to the fundamental frequency of the input signal.

After each full cycle of harmonic analysis (around once every 0.25 s), the rate of the LED is updated to reflect the most recent measured fundamental frequency. The frequency rate is scaled by a 0.1x so that the rate is detectable by the human eye (for example, if the input is 1500 Hz, the LED would toggle on/off at a rate of 150 Hz.) The hardware Timer3 interrupt was configured to trigger at each update interval to toggle the LED. An improvement could be made, to use the PWM mode instead, lowering the load to the CPU.

A video capture of the LED rate changing with frequency detected can be found at ***LED_proportional_to_Frequency .MOV*** for a demonstration of the feature.

Part 2 & 3: The splint static analysis results and corrections report can be found at ***Part_2_splint_results.docx***. Full doxygen output can be found at ***Part_3_Doxygen.zip***.

Part 4: Estimate processor load.

Below is a detailed analysis of the CPU usage, broken down by each project part and peripheral/OS block.

Part	Name	CPU Usage	Estimated Period (ms)	Comments
Part 1a	OS Timer2 Task	7.07%	0.100	Sin computation @ 10 kHz
	UART	0.00%	-	UART is interrupt-driven
Part 1b	ADC - not used	1.36%	0.051	Throw away samples
	ADC - in use	6.58%	0.051	Process and store samples
	ADC %time active	80.60%	-	
	ADC average	5.57%	0.051	Average CPU of ADC
	Harmonic Analysis Task	19.40%	264	Computes CFFT, finds peaks, prints in blocking mode over UART
Part 1c	HW Timer2 ISR	0.06%	proportional to freq.	Used to toggle the LED at the fundamental frequency rate

Total CPU Usage: **32.09%**

LIST OF DELIVERABLES

1. Detailed Technical Report
2. Executive Summary Report
3. Bill of Materials (BOM)
4. Modules 1 – 4 Design Files
5. Dhrystone Benchmark Design Files
6. Supplemental Data file
7. Simulink Data file

RECOMMENDATIONS

A more suitable component given the requirements from Keithley is the STM32F76x Cortex-M7 MCU family. This addresses many of the deficiencies of the STM32F401RE:

1. Many more pins available: 82 – 159 GPIO allow for all of the devices to connect without issue, while alleviating layout concerns.
2. Ethernet peripheral available, which could provide a LAN solution without purchasing external hardware.
3. Much more RAM (up to 512 kB), which is useful for high-end DSP processes like FFT.
4. Much higher CPU processing power (462 DMIPS published), which would cut the processing time of complex operation like FFT to a fraction of what they are on the STM32F401RE.
5. 2 built-in DACs, and very high speed ADCs (2.4 MSps), which could eliminate the need for external DAC and ADC, if the bit resolution in the specification could be reduced.
6. A double-precision floating point unit which would further aid in computation of complex functions used in a Signal Analyzer product.
7. The unit price is 2-3x higher than the STM32F401RE… however since it integrates several missing features, this could be an overall system cost reduction.

REFERENCES

Documents

STM32F401RE Reference Manual
STM32F401RE Datasheet
STM32 Nucleo-64 Boards
STM32 Nucleo-64 User Manual
STM32 HAL Library User Guide
ARM mBed handbook (https://os.mbed.com/handbook/Homepage)
ST7066U LCD Driver Datasheet (http://www.newhavendisplay.com/app_notes/ST7066U.pdf)

Tools

Keil uVision v5.24 IDE
MATLAB/Simulink
Saleae Logic (www.saleae.com)
Analog Discover 2 (http://store.digilentinc.com/)
Putty serial terminal
ARM mBed (www.mbed.com)
Digi-key (www.digikey.com)

PROJECT STAFFING

David Tracy
E: datr4356@colorado.edu
Joyce Cho
E: joyce.cho@colorado.edu
Srivishnu Alvakonda
E: srivishnu.alvakonda@colorado.edu