ECE 4680 DSP Laboratory 3: Introduction to the Cypress FM4 ARM Cortex®-M4 Board and the Keil IDE

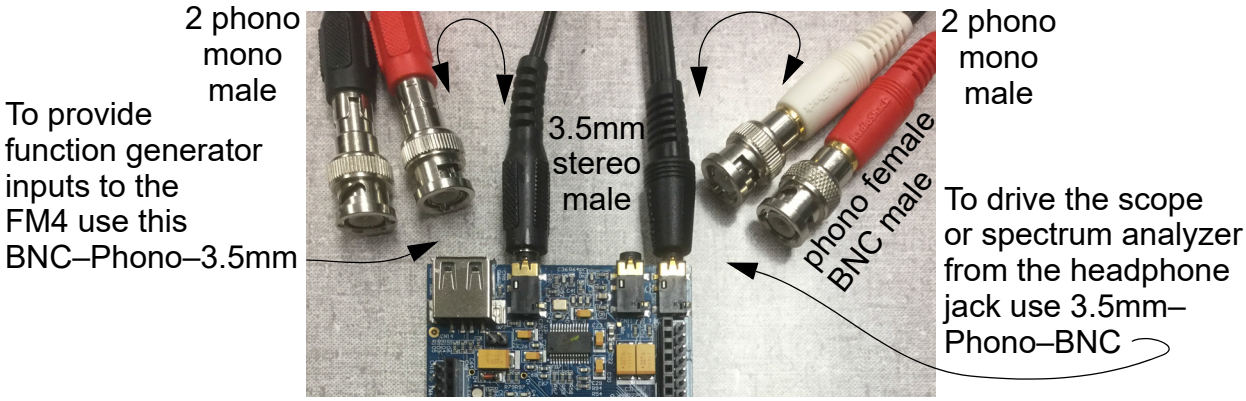
Due Date: _____

In this lab you will get introduced to the hardware and software tools that you will be using throughout the rest of the semester. You will be seeing a lot of screen shots from the Keil integrated development environment as well as waveforms captured using the Analog Discovery 2.

A short lab report is due which documents code you have written and a summary of your results. Screen shots from the scope and any other instruments and software tools should be included as well.

Problems

1. Read though the document `FM4_tools_set_with_Keil.pdf`. If you are setting up the tools on your own system carry out the actual install. If you are working in the DSP lab read through the installation instruction and then start working with the board beginning with the section entitled Testing the Installation. Demo the code found in `fm4_intr_first.c` to your lab instructor by running the code in the debugger. Show the GPIO timing signal on a logic analyzer and the left and right headphone output signals on the scope. Verify that 1 kHz and 2 kHz sinusoids are present. The lab instructor will show you where to find adapter cables to convert from 3.5 mm jacks on the FM4 board to phono jack and finally to BNC as shown in the photo below.



2. Up next you get acquainted with the GUI slider interface and writing to the serial port. Start by opening up the Lab 3 project you installed in Problem 1. Modify the project so that the main module `fm4_loop_intr_first.c` is replaced with `fm4_loop_intr_GUI.c`.

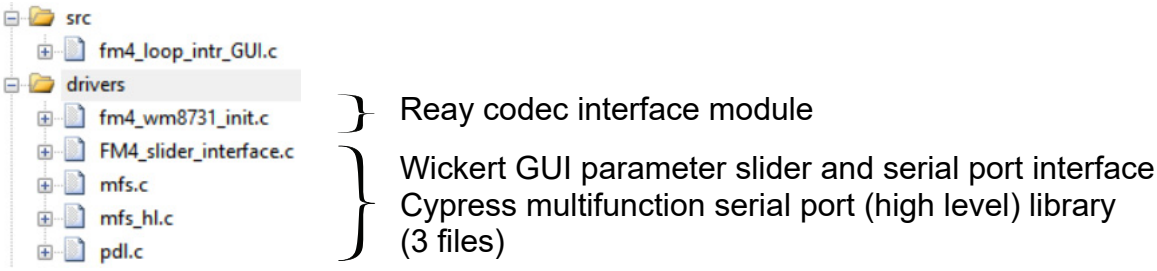
   Study the Code – The code listing for the new module is given below:

```
// fm4_loop_intr_GUI.c

#include "fm4_wm8731_init.h"
```

```
#include "FM4_slider_interface.h"

// Create (instantiate) GUI slider data structure
struct FM4_slider_struct FM4_GUI;

void PRGCRC_I2S_IRQHandler(void)
{
  union WM8731_data sample;
     int16_t xL, xR;

  gpio_set(DIAGNOSTIC_PIN,HIGH);
     // Get L/R codec sample
  sample.uint32bit = i2s_rx();

     // Breakout and then process L and R samples with
     // slider parameters for gain control
     xL = (int16_t) (FM4_GUI.P_vals[0] * sample.uint16bit[LEFT]);
     xR = (int16_t) (FM4_GUI.P_vals[1] * sample.uint16bit[RIGHT]);
     // Do more processing on xL and xR
     // TBD


     // Return L/R samples to codec via C union
     sample.uint16bit[LEFT] = xL;
     sample.uint16bit[RIGHT] = xR;
  i2s_tx(sample.uint32bit);

  NVIC_ClearPendingIRQ(PRGCRC_I2S_IRQn);

     gpio_set(DIAGNOSTIC_PIN,LOW);
}

int main(void)
{

     // Initialize the slider interface by setting the baud rate (460800 or 921600)
     // and initial float values for each of the 6 slider parameters
     init_slider_interface(&FM4_GUI,460800, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0);

     // Send a string to the PC terminal
     write_uart0("Hello FM4 World!\r\n");

     // Some #define options for initializing the audio codec interface:
     // FS_8000_HZ, FS_16000_HZ, FS_24000_HZ, FS_32000_HZ, FS_48000_HZ, FS_96000_HZ
     // IO_METHOD_INTR, IO_METHOD_DMA
     // WM8731_MIC_IN, WM8731_MIC_IN_BOOST, WM8731_LINE_IN
     fm4_wm8731_init (FS_48000_HZ,               // Sampling rate (sps)
                      WM8731_LINE_IN,            // Audio input port
                      IO_METHOD_INTR,            // Audio samples handler
                      WM8731_HP_OUT_GAIN_0_DB,   // Output headphone jack Gain (dB)
                      WM8731_LINE_IN_GAIN_0_DB); // Line-in input gain (dB)
     while(1){
          // Update slider parameters
          update_slider_parameters(&FM4_GUI);
     }
       }
```

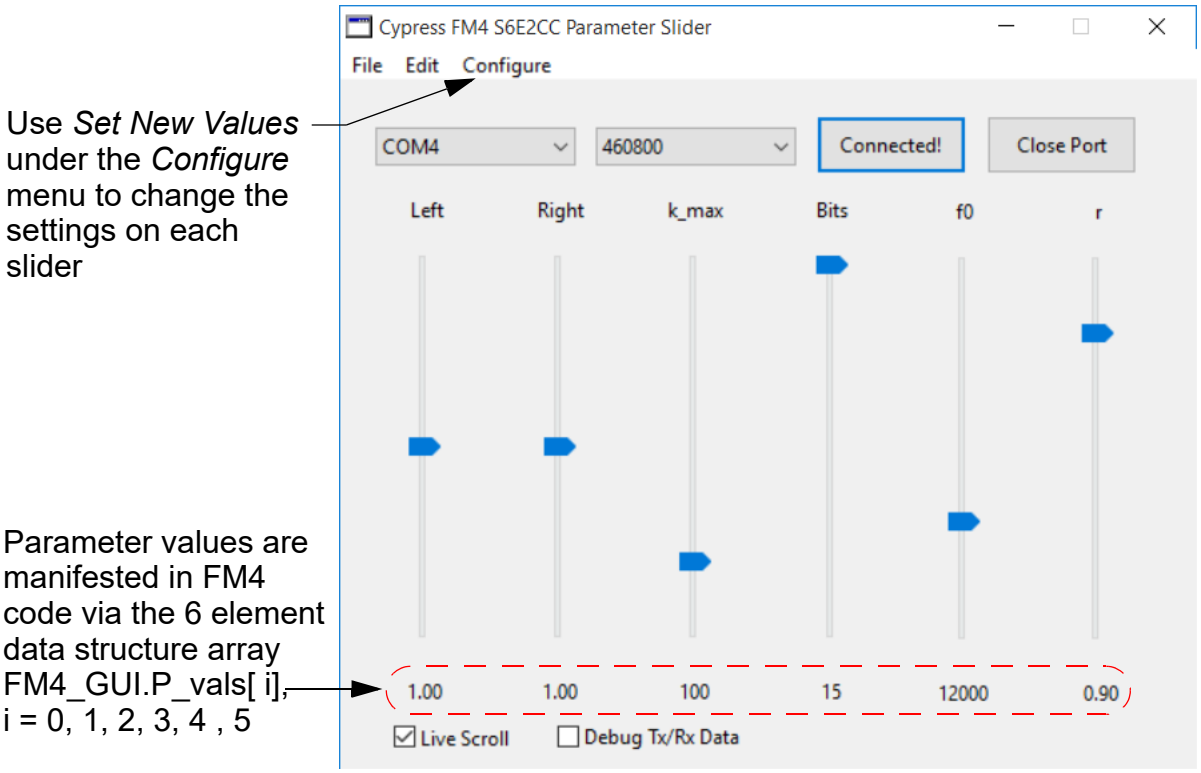
The code begins with two header file includes: (1) `fm4_wm8731_init.h` and (2) `FM4_slid-`

`er_interface.h`. The first file brings in the WM8731 stereo audio codec interface software developed by the textbook author Reay [1]. If you expand the drivers code section in Keil you can see the corresponding C-code module that is being linked into this project. If you



right-click over the include in the Keil editor it gives you the opportunity to open both the `.h` and `.c` files. Take a look, as there are many support functions and `#defines` in these files that may be useful in future programming tasks. Note the function `fm4_wm8731_init()` in main() is used to configure the codec using five inputs. In particular we will on occasion change the sampling rate (first input) from the current value of 48 ksps.

The second include file brings in a virtual serial port interface that allows the FM4 to talk to an RS232 serial com port on a Windows PC. The interface also allows a Windows GUI app to send six `float` parameters to the FM4 in real time. The is the so-called GUI Slider Control depicted below. Each of the six slider can be configured with a title, minimum and max-

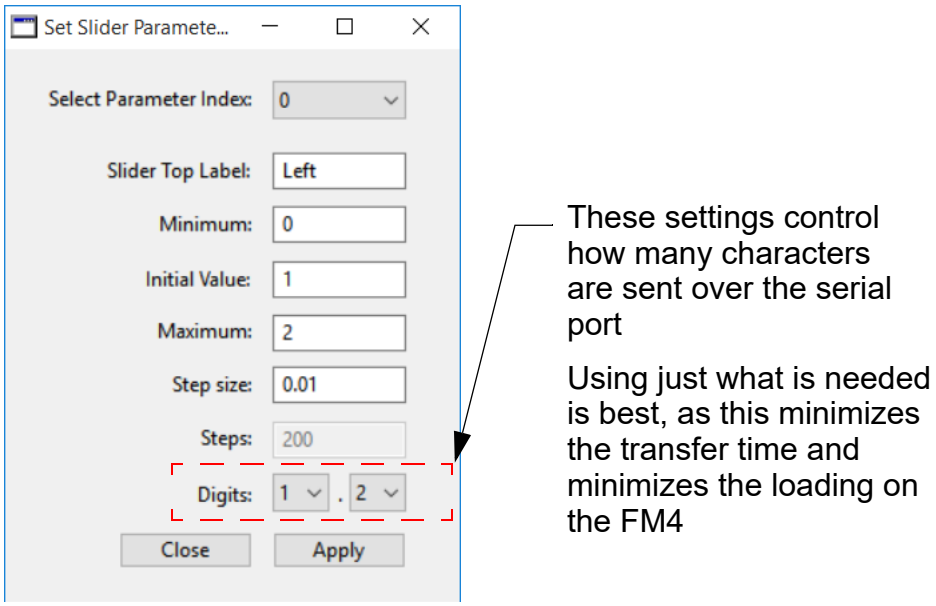


imum range and step size. In the code module `fm4_loop_intr_GUI.c` two parameter sliders are implemented: `P_vals[0]` for gain of the left channel through signal and `P_vals[1]` for

gain of the right channel through signal.

```
// Breakout and then process L and R samples with
// slider parameters for gain control
xL = (int16_t) (FM4_GUI.P_vals[0] * sample.uint16bit[LEFT]);
xR = (int16_t) (FM4_GUI.P_vals[1] * sample.uint16bit[RIGHT]);
```

The configuration of the slider parameters on the app itself is made using Set Slider Parameters under the Configure menu. The setup of slider 0 (here left) is shown below.



Slider 1 (right) is configured similarly. The Configure menu also contains parameter save and restore menu settings so that you can quickly configure the parameter slider setting when you are doing in-class demos of your work. A sample configuration file, `SliderConfig_MAC_Loops.txt` is present in the root folder of the software ZIP, `Lab3_f20201.zip`.

On the FM4 the six parameters are accessible via the data structure `FM4_GUI` defined/instantiated in the third line of code. The FM4_GUI is initialized in `main()` with the line

```
init_slider_interface(&FM4_GUI,460800, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0);
```

The slider variables are held in the `float32_t` array `FM4_GUI.P_vals[k]` for `k = 0,1,2,3,4,5`. The variables are updated in the `while` loop that sits at the bottom of `main()` via the function call

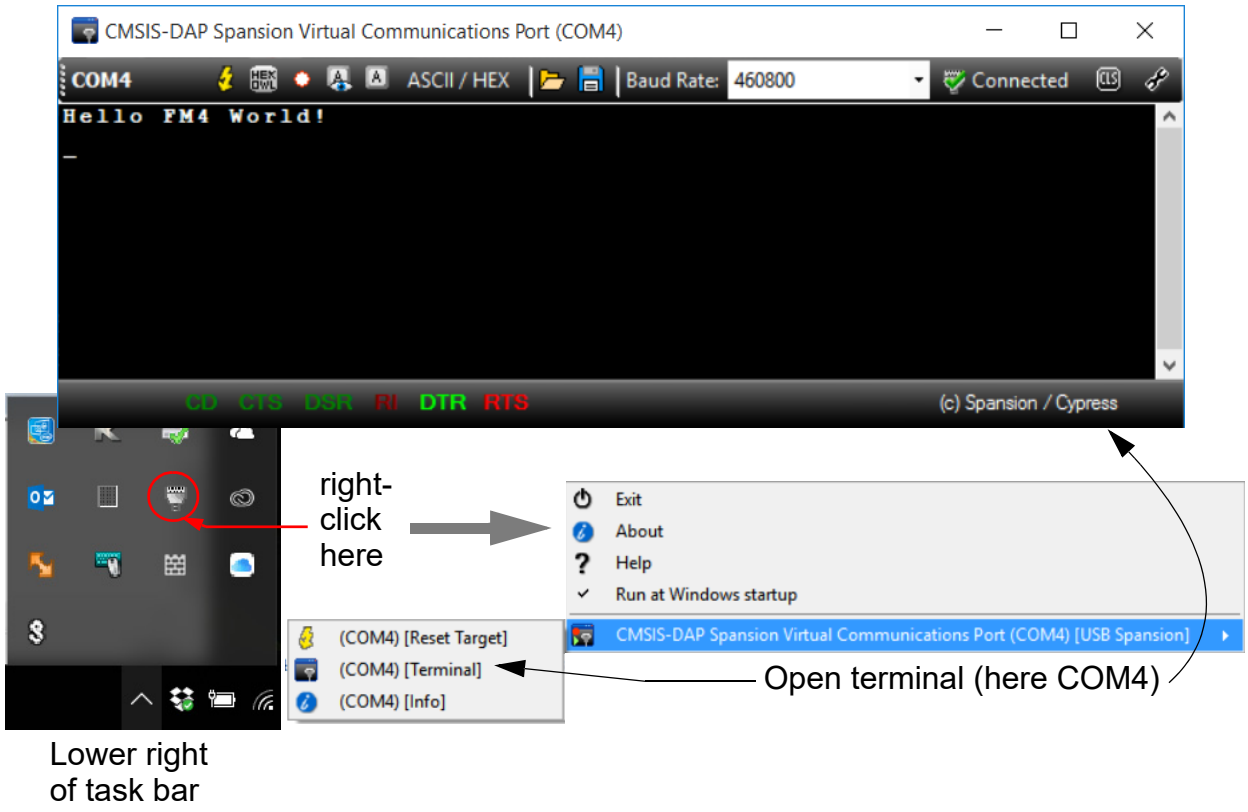
```
update_slider_parameters(&FM4_GUI);
```

The source code modules brought into the project files provide full tx/rx serial port communications. This means that messages and debug information can be sent from the FM4 up to the PC. A serial port terminal was installed , so for example

```
write_uart0("Hello FM4 World!\r\n");
```

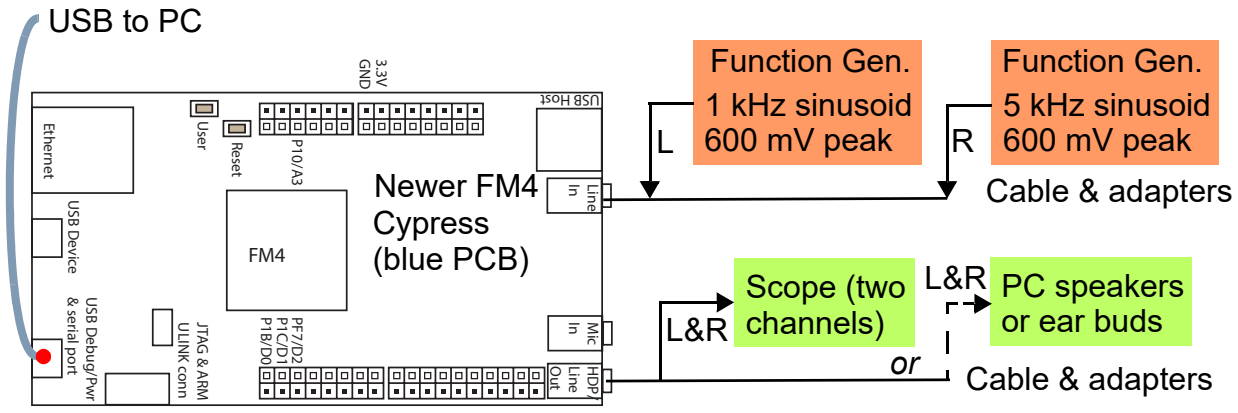
results in the output shown below on the Cypress/Spansion terminal that was installed

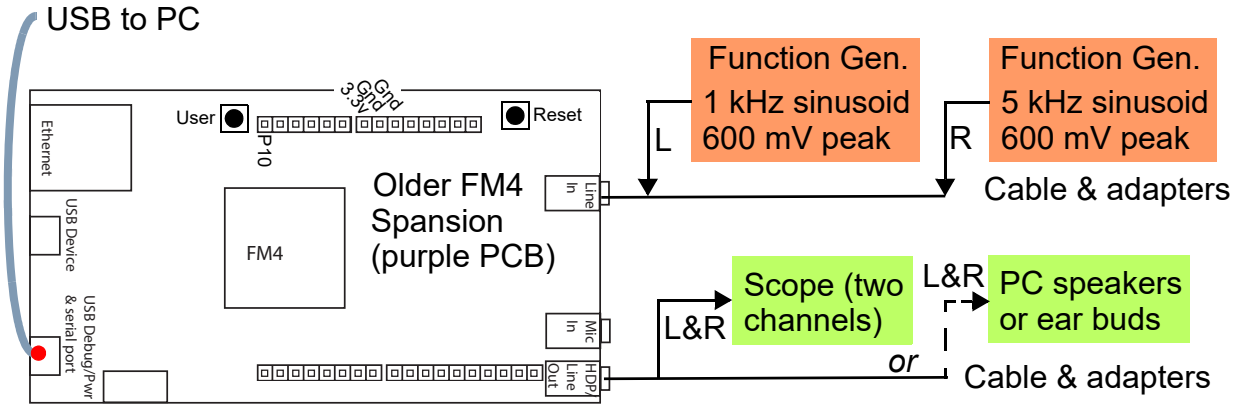
during software/board set up. To open the terminal program go to the up arrow in the right



side of the Windows task bar, and right click on it. See the above screen shots for launching the terminal.

Build this project (F7) and debug it (ctrl-F5). Interface two function generator outputs, Agilent 3325/Agilent 33120/Keysight DSOX6004A are three options, via the adapter cables shown under) to the line input of the FM4. Set the generator outputs to produce 1 kHz and 5 kHz sinusoids at an amplitude of 600 mV peak. View the output waveforms on the scope and verify that the first two GUI sliders (0 and 1) can be used to adjust the the output level in real-time. You will first need to start up the app FM4_GUI_slider found in the root of the proj-





ect folder. Then use the pull down in the upper left to set the proper com port (you may need to run the Windows device manager to very the port or see what the Spansion terminal program has found). Set the baud rate to 460800 and then click the Connect button. Note if in the interim you connected the Spansion terminal program to the FM4 com port, make sure to disconnect it, as only one serial port app can be connected to the same com port at a time. For the GUI slider control to work you will have connect the app to the com port at the correct baud rate (460800 bits/s). Finally, listen to the audio output via the 3.5mm headphone jack using ear buds or the PC speakers found at the lab bench. Give a brief demo to your lab instructor.

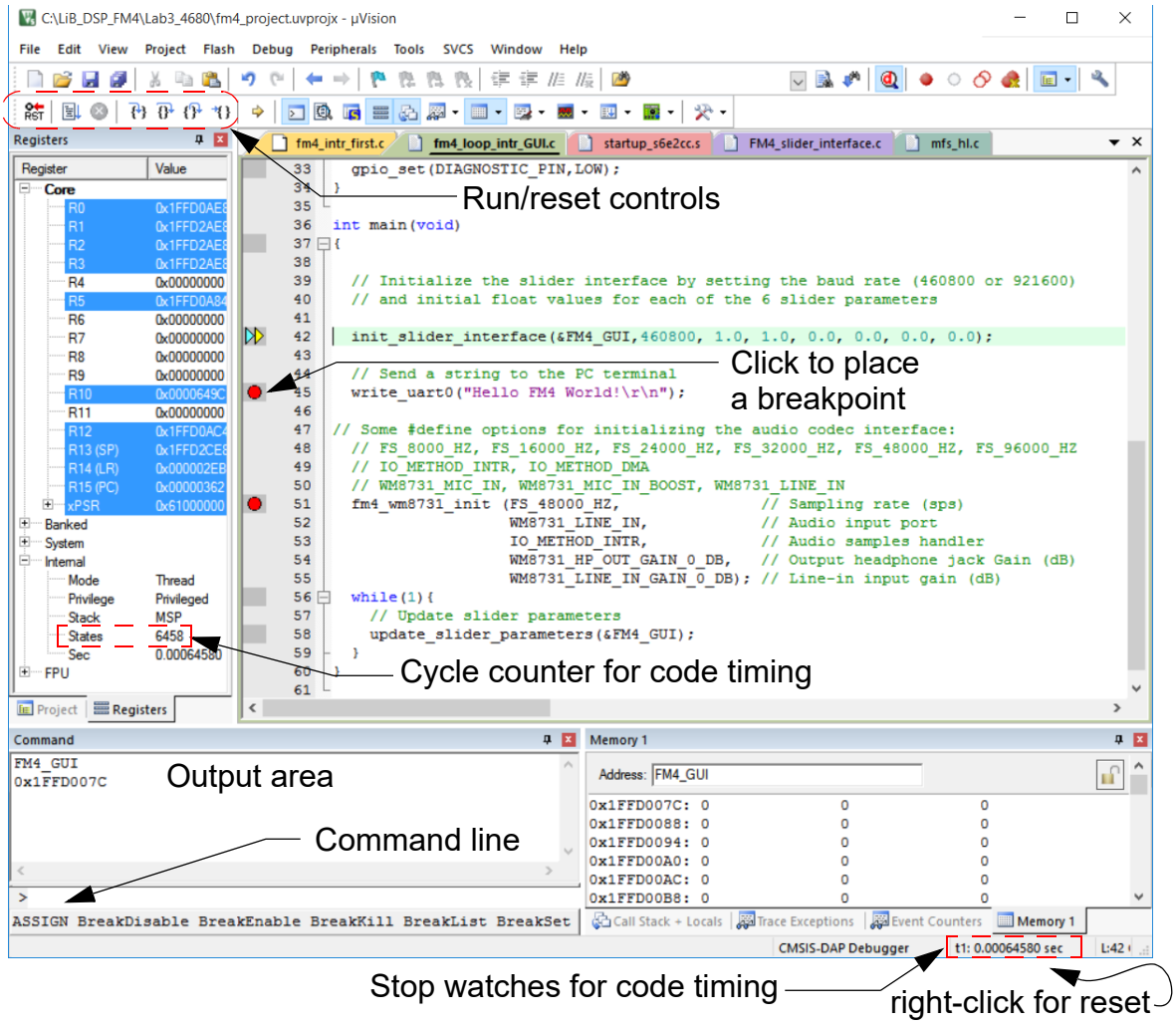
A Little Sampling Theory

3. Using your knowledge of sampling theory and aliasing (recall what you learned in ECE 2610), you will now make some additional observations. For sampling rate f_s the lowpass sampling theorem says that a system composed of an ADC followed by a DAC has a usable frequency band from zero to $f_s/2$ Hz. Signals entering the system above $f_s/2$ will be aliased back to the $[0, f_s/2]$ fundamental alias frequency band. The ADC employed on the FM4 Pioneer Kit incorporates an anti-aliasing filter that effectively prevents signals above $f_s/2$ Hz from passing to the output.

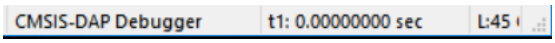
   The sampling rate is set by the first argument to `fm4_wm8731_init()` in `main()`. The current setting is 48 kHz. With the program running, increase the frequency of one the two function generators and verify that the output disappears at about 24 kHz. Rebuild the project with the sampling rate reduced to 24 kHz and find the function generator frequency where the output disappears. Are your observations as expected?

Keil Debugging

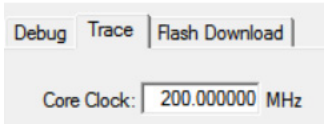
4. Explore the use of Keil's debugging related capabilities, such as setting break points, the watch window, and the memory window, and the command line. Begin by starting the debugger (ctrl-F5 starts and stops the debugger). Set break points at the locations shown in the screen capture below:

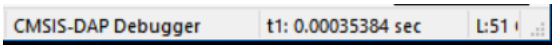


- **Profiling using the States Counter and the Stop Watches**
  - Timing or profiling code is very important in real-time DSP
  - In addition to writing to GPIO to time code, we can use facilities within Keil in combination with break points
  - When the debugger starts see that it is sitting line 42 as shown above
  - Next click (F5) to run to the first breakpoint
  - Note the value of states (CPU cycle counter): before _____; after _____
  - Now run to the next breakpoint and again record the value of states _____
  - See that the CPU cycle count should be around 70769 to write a simple text string out the serial port! A cycle with the FM4 $f_{\text{clk}} = 200\,\text{MHz}$ is 5ns, so $70769 \leftrightarrow 0.35\,\text{ms}$ and with $f_s = 48\,\text{kHz}$ the interrupt period is $20.8\,\mu\text{s}$, meaning it takes more that one sampling clock period to write a formatted string!
  - Stop watches t1 and t2 time code in seconds through the lower right task bar of Keil

when debugging: 

– Make sure the debugger is configured properly by clicking (alt-F7) and then click the Debug tab; then on the far right click the Settings button next toe CMSIS-DAP Debugger pull down; finally select the Trace tab and see that the Core Clock is set at 200 MHz:

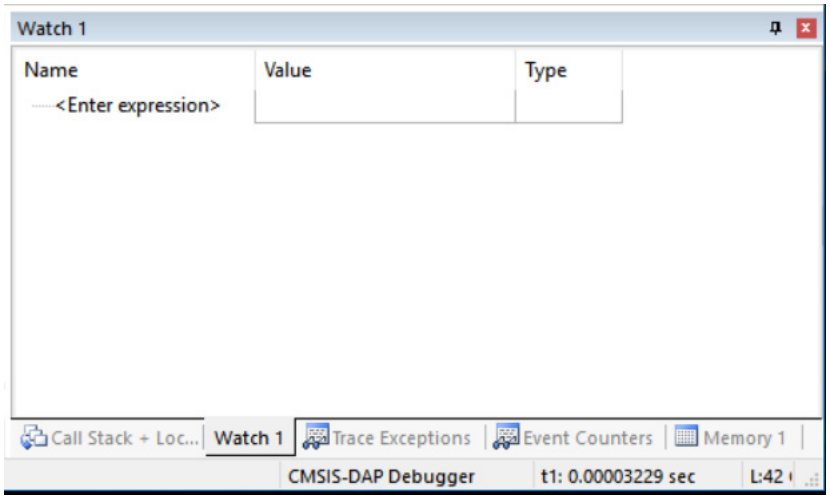


– Stop and restart the debugger and then run to the first breakpoint, reset t1 (right-click) and record the differential time to get to the second breakpoint; verify that the value you observe is similar to: 

(353.8 µs)

- **Adding watches and hovering**

– Again start the debugger and in the lower right viewing right viewing area of Keil click on the Watch 1 tab (do not advance to the first breakpoint))



– Double click <Enter expression> and type the structure variable `FM4_GUI` followed by enter

– You should now see:

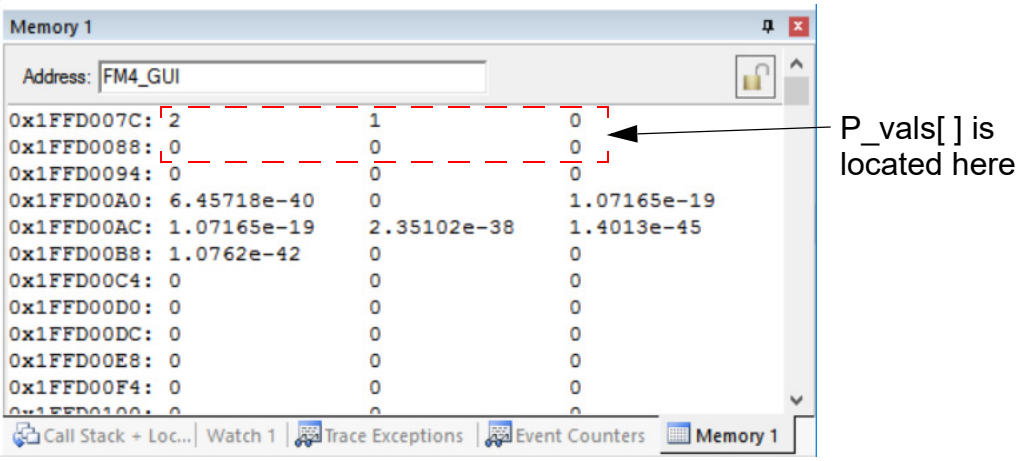
– Expand the view by clicking so that you can see the fields of the data structure, and in particular see the values held by `P_vals[]`

– Run to the first breakpoint (F5) and notice that `P_vals[0]` in particular has now been initialized to 1.0

– Verify that you can also *hover* the mouse pointer over any variable in the code window, such as `FM4_GUI`, and see its value and/or its memory address

– Note also that when you place a variable in the watch window you can also change it by clicking in the value column to change it; when you run again, execution continues with the variable change

- **Viewing memory**
  – Continue from the above exercise and run past the second breakpoint so the code is now running continuously (make sure you are still in the debugger)
  – Click the Memory 1 tab next to the Watch 1 tab and enter the variable name `FM4_GUI` in the Address box (Keil will automatically replace the name with the starting address in memory where the variable is stored)



  – Right click in the memory display area below and from the context menu select Float (and if needed Decimal) for the display mode; see that the first six float values, starting at address `0x1FFD007C` are as shown above (note here `P_vals[0]` was earlier changed to 2.0 in the watch window)
  – The first six float values correspond to array `P_vals[]`; to verify this connect the app FM4_GUI_slider to the FM4 serial port and see that by changing the sliders you see the values in the memory window change accordingly (nice!, real-time debugging)

5. Interface a third slider into the ISR function `PRGCRC_I2S_IRQHandler()` to implement a panning control between the right and left audio channels. You will need to add some more variables. The algorithm needs to implement

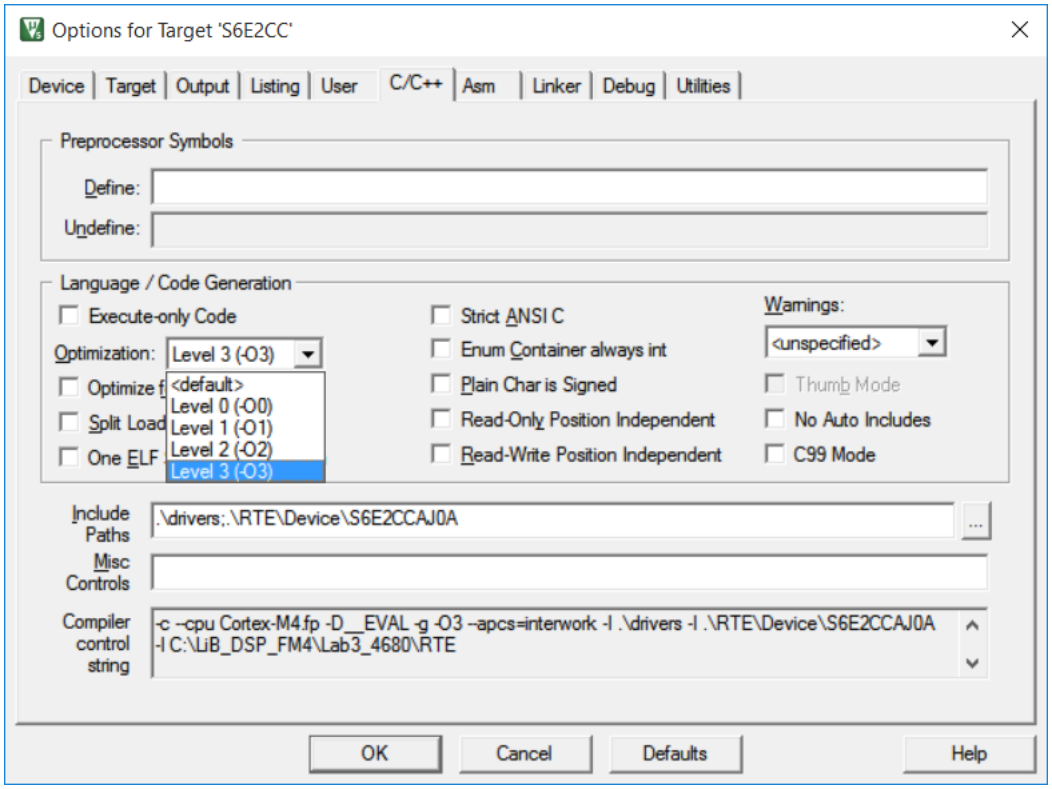
$$\begin{aligned} R_o &= (1-a)R_i + aL_i \\ L_o &= aR_i + (1-a)L_i \end{aligned} \tag{1}$$

where R_i/L_i are the right and left input samples values, R_o/L_o are the right and left output sample values, and a is a variable controlled by the slider that ranges over [0, 1]. In the algorithm the data type needs to be `float32_t`, but in the end you must cast back to `int16_t` for writing to the codec. Configure the third slider (`P_vals[2]`) to serve as a in your C-code. Again input the 1 kHz and 5 kHz sinusoids as before and then listen to what happens as you move the slider up and down. Describe what this simple processing algorithm is doing.

6. In real-time DSP all of the DSP math that runs inside the interrupt service routine (ISR) needs to complete before the sampling rate clock fires the next interrupt. Code profiling is one approach to time individual sections of code, and as seen earlier Keil does indeed support this. With the GPIO on the FM4, it is possible to send an actual timing waveform to a digital pin that reflects the time spent in the ISR relative to the sampling rate clock. You were briefly exposed to this in Problem 1. Again use the logic analyzer or scope to measure ISR from pin `P10/A3` throughout the problem.

   When the ISR has more work to do in later labs, you will want to improve performance using the optimizing C compiler. For now you will characterize the performance of this simple ISR under various optimization levels.

   With debugging turned off click Options for Target... (alt-F7), click the C/C++ tab. This brings up a dialog box with a pull-down that allows you to change compiler optimization settings. The current setting is Level 3 (-O3), which is full optimization.



   Insert a "do-nothing" for loop that performs a repeated multiply and accumulate operation inside the ISR, e.g.,

```
k_max = (int16_t) FM4_GUI.P_vals[3];
for(k = 0; k < k_max; k++)
{
    z += 2.0f*21.0f;
}
```

   You need to make `z` a float32_t type so that you will be exercising the floating point math capabilities of the Cortex-M4. So that you can change the processor loading use the fourth

GUI slider control configured to take integer values ranging from 10 to 500.

Initially set the number of iterations/loops to 100. Holding $f_s = 48$ ksps increase the loop count until the ISR service time is about 50% of the maximum time available, i.e. 20.833/2 μs. Note the value of `k_max` = _____ loops

- Investigate the speedup realized by the default `-O3` optimization by changing to `-O1`; record the new value of time spent in the ISR = _____ μs
- In running the ISR there is overhead, that is time to enter and return from the ISR and time to read and write the signal samples from and to the codec. For the case of `-O3` and the single multiply and accumulate per loop, determine the ISR time as a linear relationship in `k_max`, i.e., find constants A and B in

$$T_{\text{ISR}} = A + B \cdot k_{\max}(\mu\text{s}) \tag{2}$$

A = _____ and B = _____

7. In this last problem you progressively limit the number of bits per sample. The audio codec receives and returns `int16_t`, which is a 16-bit signed integer. Using simple bit shifting under the control of the fifth slider control, write code to eliminate magnitude bits ranging from a maximum of 15 (one sign bit must remain) down to 1 bit. Comment on the audio quality as the number of bits is reduced. Note at the 1 bit setting the output should contain just two amplitude levels. Demo this to the lab instructor. Hint: When you right shift the least significant bit (LSB) is lost. When you shift left a zero bit is shifted in the LSB position. Remember to cast the `P_vals[4]` to `int16_t`.

References

[1] Donald Reay, *Digital Signal Processing Using the ARM® Cortex®-M4*, Wiley, 2016.

Appendix A: CMSIS-DSP Data Types

When programming in C the ARM Cortex-M Software Interface Standard (CMSIS) adopts coding standards for embedded systems from the *MISRA C* (Motor Industry Software Reliability Association). The original MISRA standard was created in 1998 as guidelines for programming C in vehicle electronics. A major impact for our purposes is C *type defs* to insure that the ANSI types are properly represented for a given compiler, e.g. CMSIS includes `stdint.h` which provides:

Standard ANSI C Type	MISRA C Type
`signed char`	`int8_t`
`signed short`	`int16_t`
`signed int`	`int32_t`

Standard ANSI C Type	MISRA C Type
`signed __int64`	`int64_t`
`unsigned char`	`uint8_t`
`unsigned short`	`uint16_t`
`unsigned int`	`uint32_t`
`unsigned __int64`	`uint64_t`

When using CMSIS-DSP and in particular floating point math (think Cortex-M4 and M7), more types are added via `arm_math.h`.

MISRA/ ANSI	MISRA C *like*	Description
`int8_t`	`q7_t`	8-bit fractional data type in 1.7 format.
`int16_t`	`q15_t`	16-bit fractional data type in 1.15 format.
`int32_t`	`q31_t`	32-bit fractional data type in 1.31 format.
`int64_t`	`q63_t`	64-bit fractional data type in 1.63 format.
`float`	`float32_t`	32-bit floating-point type definition.
`double`	`float64_t`	64-bit floating-point type definition.

Use these data types in all of your FM4 coding. The most common types will be `int16_t` and `float32_t`.

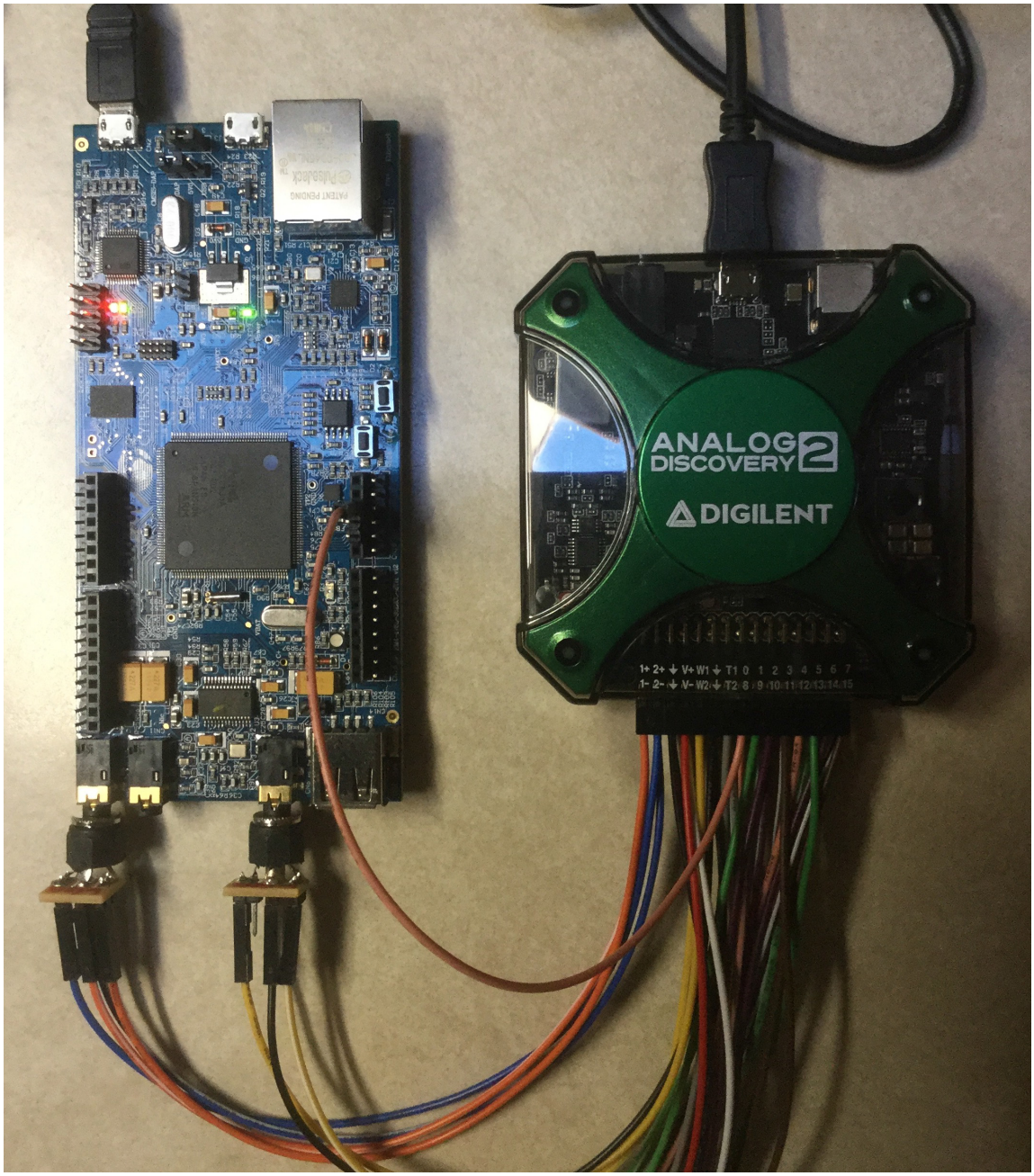
- **Note**: To include `arm_math.h` in a project requires that you begin the includes section of a code module with

```
#define ARM_MATH_CM4
```

Appendix B: Dr. Wickert's Analog Discovery 2 Setup

The preparation and documentation of the FM4 Pioneer Kit experiments was made much easier by have test and measurement tools readily available anywhere by using the Analog Discovery 2 (AD2) from Digilent[1]. To make the AD2 even more user friendly for rapid test and measurement set-up a pair of 3.5mm male jack to male pin header adapters were constructed. Finally a trip of male-to-male pin headers was place over the Arduino pins surrounding the P10/A3 GPIO pin. A photograph of this test set-up is shown below.

1. http://store.digilentinc.com/analog-discovery-2-100msps-usb-oscilloscope-logic-analyzer-and-variable-power-supply/


ANALOG DISCOVERY 2
DIGILENT