A tunable single *biquad* tunable notch filter, using a GUI slider control talking through the FM4 virtual serial port.

- **Inputs**
  - Power
  - Debug
  - UART
- **Outputs**
  - 3.5mm to jumper adaptors
  - Use GPIO for code timing

$\tilde{f} = 48\, \text{kHz}$
Introduction

- Using a low-cost development kit to explore audio signal processing using an ARM(R) Cortex(R)-M4 microprocessor.
- In this project, we use the Cypress FM4 Pioneer Kit ($50) to develop audio signal processing applications, such as filtering, audio special effects, and spectrum analysis, to name a few.
- A configurable PC GUI app allows for real-time control of DSP apps running on the FM4.
- Hands-on interactive demos allow you to hear and see signals being processed from audio sources such as a microphone(s) and music players.
- Algorithms from simple to very complex can be implemented on the FM4 with its 200 MHz clock and floating point math capabilities.
- We use Python’s scipy stack to design filters and analyze captured results from the Digilent Analog Discovery 2 portable instrumentation system.
Consider the text: Donald S, Reay, *Digital Signal Processing Using the ARM(R) Cortex(R)-M4*, Wiley 2016, which has Web Site support for the FM4.

Visit my Web Site: www.eas.uccs.edu/~wickert/ece4680/ and look for Lab 3 and the associated ZIP package plus other documents.

◊ The code base for audio loop through and the GUI slider app is included.
FM4 Board and Subsystems

- The system out-of-the-box is enhanced through the use of the virtual serial port (com port), the GUI slider control, and the Analog Discovery for test and measurement.

- CPU clock is 200 MHz, with single cycle floating point multiply & add.

Out of the box

Interfaces added in purple
Analog Interface Details and Codec Software

- Line and Mic inputs are provided and a headphone output jack that doubles as line out is also provided.
- The Codec interface code found in the Reay text provides both sample-by-sample and frame-based processing using direct memory access (DMA).

<table>
<thead>
<tr>
<th>ADC</th>
<th>ADC</th>
<th>I2S 2:1</th>
<th>Single Sample Processing</th>
<th>I2S 1:2</th>
<th>DAC</th>
<th>DAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>R</td>
<td></td>
<td>LRLRLR...</td>
<td></td>
<td>L</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Process one sample per IRQ</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>R</td>
<td>I2S 2:1</td>
<td>DAC</td>
<td>DAC</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Process BUSIZE (L + R) samples per DMA cycle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(a) Sample-Based Processing BUFSIZE pingIN pingOUT (b) Frame-Based Processing BUFSIZE pongIN pongOUT</td>
<td></td>
</tr>
</tbody>
</table>

- The FM4 specific circuit interfaces give some insight into additional filtering in the analog domain.
Programming the Board

- ANSI C is used for programming with the MISRA + ARM data types (https://en.wikipedia.org/wiki/MISRA_C)

- Shown below is a basic loop through program under GUI slider control

```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.F_vvals[0] * sample.uint16bit[LEFT]);
    xR = (int16_t) (FM4_GUI.F_vvals[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);
}

int main(void)
{
    NVIC_ClearPendingIRQ(FRGCRC_I2S_IRQHandler);
    gpio_set(DIAGNOSTIC_PIN, LOW);

    // 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, 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 (sp/s)
    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_GAIN_0_DB); // Line-in input gain (dB)

    while(1)
    {
        // Update slider parameters
        update_slider_parameters(&FM4_GUI);
    }
```
Other Examples

- Frequency modulation receiver (discriminator)
- Spectrum analyzer
- Others to see in the DEMOs

FM Receiver

- Choose carrier frequency around 10 – 15 kHz
- Maybe use a second FM4 as a custom signal source with a PN code driving a DDS to produce binary FSK

\[
x_{FM}(t) \xrightarrow{\text{ADC}} x[n] \xrightarrow{\cos() \text{ DDS}} x_I[n] \xrightarrow{\text{LPF } f \approx W} z_I[n] \xrightarrow{\text{LPF } f \approx W} z_d[n] \xrightarrow{\text{DAC}} x_m(t)
\]

Results in a 25 tap filter, but you need two of them

```matlab
% Example code for designing a filter
\text{d\_pass} = 0.2; 
\text{d\_stop} = 80.0; 
fs = 48000; 
fp = 3500; 
fs = 100000; 
n, ff, aa, wts = optfir.remezord([\text{d\_pass}, \text{d\_stop}], [1,0], [10**(d\_pass/20),10**(d\_stop/20)], \text{fsamp}=48000); 
% Bump up the order by 5 to bring down the final d\_pass & d\_stop
n\_bump = n + 5; 
bl = signal.remez[n\_bump, ff, aa(0:5)], wts, Hz=2)
```
Spectrum Analyzer

Collect subframes using a 128 point DMA buffer

GUI Slider App with plot added