Laboratory 1: Introduction to Real-Time Digital Signal Processing

EE3221 Digital Signal Processing
Electrical Engineering and Computer Science Department
Milwaukee School of Engineering
Last Update: 9 March 2017

Contents

1 Overview

The Cypress (Spansion) FM4 board (FM4-176L-S6E2CC-ETH Pioneer Kit) is a low-cost development platform for implementing real-time DSP applications [1]. It features a 200MHz ARM Cortex-M4 based processor with a single-precision floating point unit. Real-time processing of audio signals is enabled by the Wolfson WM8731 audio codec included on the development board, which connects to the I2C (control) and I2S (data) peripheral interfaces of the ARM processor. The Keil MDK-ARM integrated development environment enables software written in C to be compiled, linked, and downloaded to run on the Cypress FM4 board.

Through this exercise you will gain familiarity with the development hardware and software tools.

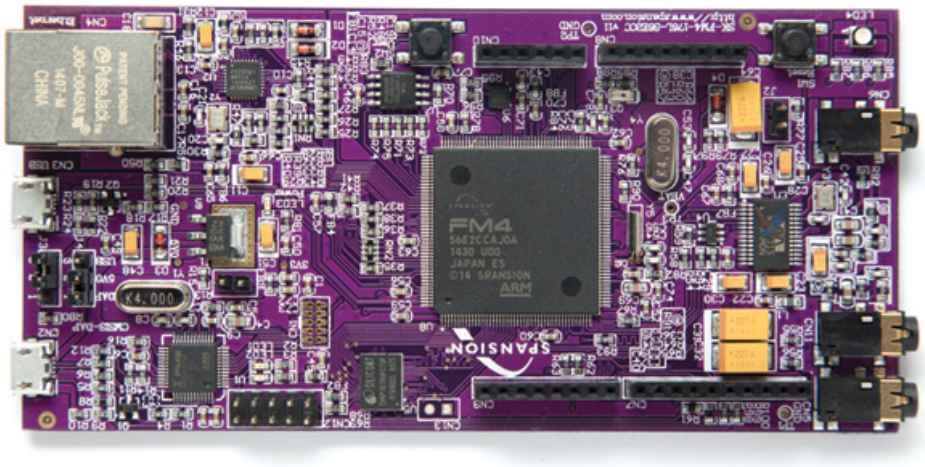

Figure 1: Cypress FM4-176L-S6E2CC-ETH Pioneer Kit (image from http://www.cypress.com/)

.

2 Digital Signal Processing System Block Diagram

A basic DSP system, suitable for processing audio frequency signals in real-time, is shown in Figure 2. The Cypress FM4 hardware show in Figure 1 implements this block diagram.

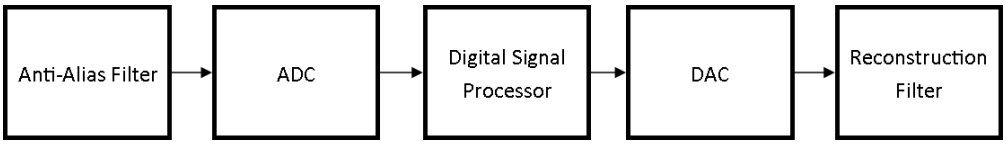

Figure 2: Block diagram of basic DSP system.

The digital signal processing is performed by the ARM Cortex-M4 microcontroller. The anti-alias filter, ADC, DAC, and reconstruction filter functionality are provided by the Wolfson WM8731 codec. This combination of components ensures *proper* analog input and output. In this context, "proper" means that an analog signal can be faithfully reconstructed from its samples so long as the Nyquist Theorem is satisfied. The

quality of reconstruction is a function of numerous parameters, such as the bit-depth of the ADC and DAC converters (our implementations will use 16-bit samples from the WM8731 codec). Suffice it to say that the Cypress FM4 hardware provides very high-quality analog I/O. Details will be investigated in future lab exercises.

3 IDE Download and Installation

MDK-ARM v5.17 is available for download at `https://faculty-web.msoe.edu/prust/armdsp`.

To install the program, run the executable `mdk_517.exe` and follow the instructions.

IMPORTANT: Install MDK-ARM to the default destinations, as the projects provided in this and future labs refer to the default installation directories.

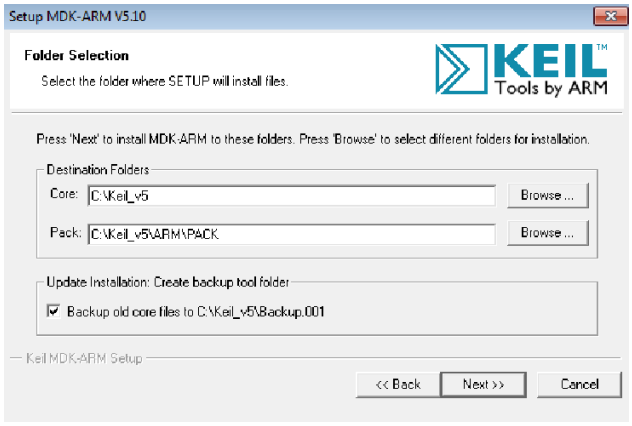

Figure 3: Default installation folder.

IMPORTANT: During installation, you may be asked if you want to install additional device software. It is important that you install all drivers when prompted.

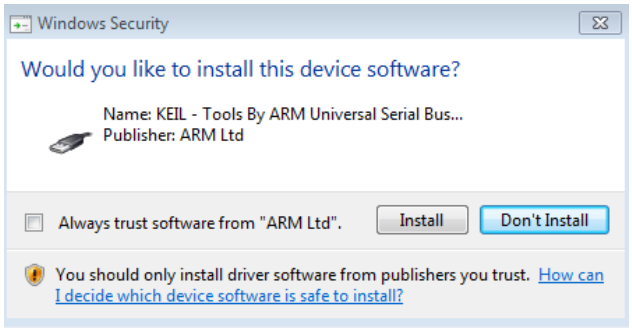

Figure 4: Serial bus driver installation.

After installation completes, run the Keil uVision5 program. A Welcome to Keil Pack Installer window may appear, in which case click OK. The Pack Installer is a utility for installing, updating and removing software

packs. If it does not open automatically, select Manage and Pack Installer from the Project menu. If you are running the Pack Installer for the first time, it may take several minutes for it to update.

Through the Pack Installer window you can select a device and then view available software packs for that device. You can install or update a software pack by clicking in the Action column. In the Device window, select Cypress (or Spansion) FM4 Series S6E2CC S6E2CCAJ0A.

Install or update the following software packs:

- Keil::ARM_Compiler
- Keil::MDK-Middleware
- Keil::FM4_DFP
- ARM::CMSIS

When finished, the Pack Installer window should list these software packs as "Up to date." Close the Pack Installer window.

4 Cypress FM4 Connection and Installation

The Cypress FM4 board connects to a host PC via a USB cable using a CMSIS programming and debugging tool. Connect the provided USB cable to connector CN2 on the Cypress FM4 board. Connect the other end to your PC. Allow Windows to complete the driver installation, which may take several minutes.

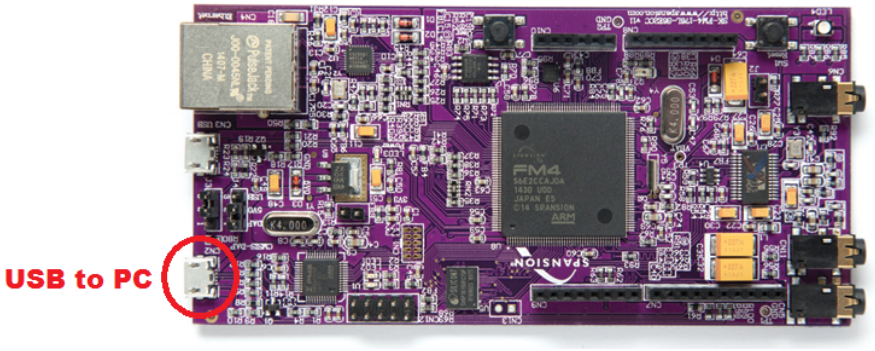

Figure 5: USB connections between PC and Cypress FM4 board.

5 Example Program - Analog Input and Output

The DSP equivalent to the infamous hello world program is to simply copy input samples read from the ADC to the DAC. The C-language source code for such a program is listed in Figure 6. This simple program serves as a template for other programs that are based upon the same interrupt-driven real-time processing model.

```
1  // loop_intr.c
2
3  #include "audio.h"
4
5  volatile int16_t audio_chR=0;
6  volatile int16_t audio_chL=0;
7
8  void I2S_HANDLER(void) {
9
10   gpio_toggle(P2_10);
11
12   audio_IN = i2s_rx();     //32-bits; 16-bits channel left + 16-bits channel right
13   audio_chL = (audio_IN & 0x0000FFFF);
14   audio_chR = ((audio_IN >>16)& 0x0000FFFF);
15
16   audio_OUT = ((audio_chR<<16 & 0xFFFF0000)) + (audio_chL & 0x0000FFFF);
17   i2s_tx(audio_OUT);
18 }
19
20 int main(void)
21 {
22   gpio_set_mode(P2_10,Output);
23   audio_init ( hz48000, mic_in, intr, I2S_HANDLER);
24
25   while(1){}
26 }
```

Figure 6: Listing of program `loop_intr.c`

The C source file `loop_intr.c` listed in Figure 6 operates as follows:

In the function `main()`, an initialization function `audio_init()` is called. This function configures I/O and interrupts such that the WM8731 codec will sample the analog input signal and interrupt the processor at the sampling frequency determined by the parameter passed to the function. In this case, the parameter `hz48000` sets the sampling frequency to 48kHz. The parameter `mic_in` specifies that input to the WM8731 ADC will come from the microphone input (connector CN11) on the Cypress FM4 board. Changing this parameter to `line_in` specifies the line-level audio input (connector CN6) on the Cypress FM4 board. Parameter `intr` and `I2S_HANDLER` specify that interrupt-based (as opposed to polling- or DMA-based) I/O will be used, and the name of the interrupt. As a result, each time an interrupt occurs, the interrupt service routine `I2S_HANDLER` will be called. Exactly one interrupt will occur per sampling period. Both left and right audio channels are processed in the same call to the ISR.

Also in the function `main()`, the function `gpio_set_mode()` is called to configure a general-purpose I/O pin as an output. This pin will be used for interrupt timing measurements in a later part of the lab exercise. Following initializations, the function `main()` enters an endless `while()` loop, doing nothing but waiting for interrupts.

At the beginning of the ISR `I2S_HANDLER()`, the value of the GPIO pin is toggled providing a logic change each time an interrupt occurs. Within `I2S_HANDLER()`, the function `i2s_rx()` returns the current input

sample read from the I2S peripheral. The data contained in variable `audio_IN` (a 32-bit signed integer, declared in `audio.h`) is then composed of 16 bits for the right channel and 16 bits for the left channel. To process each channel independently, the channels are separated in lines 13 and 14. Since this example program is a simple audio loop (i.e., samples read the from ADC are copied to the DAC), the left and right channel samples are packaged into variable `audio_OUT` and sent back to the I2S peripheral using the `i2s_tx()` function.

6 Building and Running Programs

The following steps will guide you through the process of building and running C language DSP programs on the Cypress FM4 board.

1. Download `LAB_1.zip` from the website. Unzip the file to a location of your choosing. This will create a folder named `LAB_1`. Browse to the folder.
2. Open the Vision 5 project DSP_LiB by double clicking on its icon in the `LAB_1` folder. You should see a project structure as shown in Figure 7. Note: If prompted to install a missing package, choose to install the package and wait for the Pack Installer to finish before proceeding.

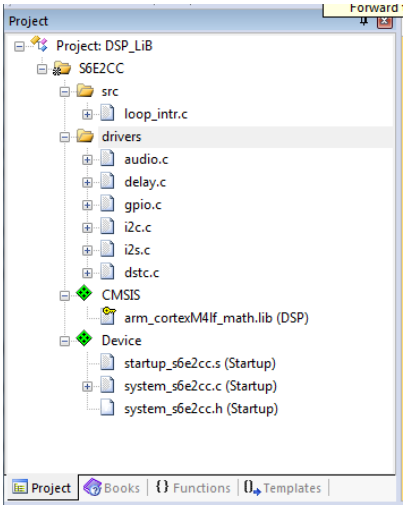

Figure 7: Project structure within Keil uVision.

3. Build the project by selecting Build target from the Project menu or by clicking the Build toolbar button. The project should build without errors.
4. Connect the Cypress FM4 board to the host PC using a USB cable.
5. Plug headphones and a microphone into the corresponding sockets on the FM4 board, as shown in Figure 8
6. Switch to the debugger by clicking on the Start/Stop Debug Session toolbar button. Doing so will automatically download the executable code into the flash memory of the ARM processor.
7. Once the debugger windows have appeared, click on the Run toolbar button.
   IMPORTANT: Never download and run software to the board with headphones affixed to your ears. The onboard headphone amplifier has high gain and could damage your hearing. When testing software, slowly lift headphones to your ears.

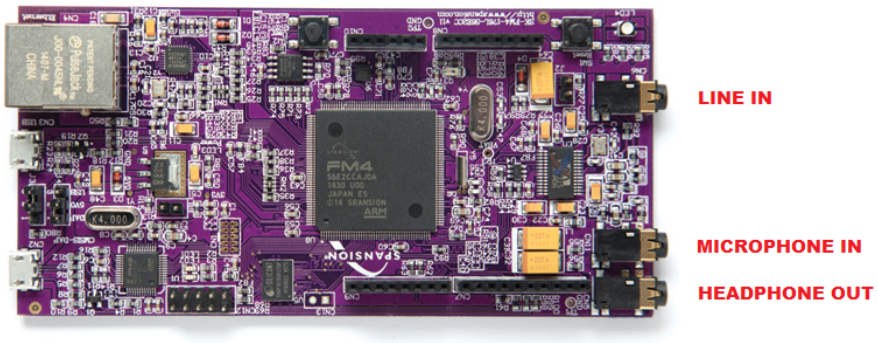

Figure 8: Audio connections on Cypress FM4 board.

8. Once the program is running, you should be able to hear the sounds picked up by the microphone in the headphones. Depending on the characteristics of the microphone and headphones, the sound may be louder or quieter. If you cannot hear anything, try blowing gently onto the microphone.

7 Sampling Rate Verification

As described earlier, a GPIO pin can be used as an indicator for execution time of the real-time program. In the case of `loop_intr.c`, the GPIO pins value is toggled each time an interrupt occurs (line 10):

```
gpio_toggle(P2_10);
```

Since interrupts should occur once per sampling period, the expected signal on pin P10 is a square wave of frequency 24 kHz (sampling rate is 48 kHz). Connect an oscilloscope probe to the selected pin on the FM4 board to confirm this. The GPIO pin may be set (HIGH) or reset (LOW) using program statements

```
gpio_set(P2_10, HIGH);
gpio_set(P2_10, LOW);
```

8 Adding Delay

The program listing shown in Figure 9 shows a simple modification of the audio pass-through program in which delayed samples of the left audio channel are added to the current left audio channel output. The program listing implements the block diagram shown in Figure 10. This example illustrates a key advantage of DSP - digital processing allows a variety of audio effects to be implemented quickly and easily through software modifications.

The delay line is implemented using the array buffer to store samples as they are read from the ADC. Once the array is full, the program overwrites the oldest stored input sample with the current or newest, input sample. Just prior to overwriting the oldest stored input sample in buffer, that sample is retrieved and added to the current input sample and output to the DAC. The length of the delay is determined by the value of the constant `DELAY_BUF_SIZE`. As supplied, the program listing uses a length 24000 array, corresponding to 24000*(1/48000) = 0.5 second delay.

The source file `delay_intr.c` is included in the `Lab_1\src` folder. You can replace `loop_intr.c` with `delay_intr.c` by doing the following:

1. Right-click on `src` in the Keil Project window, choose `Add Existing Files to Group src`, and then navigate to and select the source file `delay_intr.c`.

2. Right-click on `loop_intr.c` in the Keil Project window and choose `Remove File`.

```c
// delay_intr.c

#include "audio.h"

volatile int16_t audio_chR=0;
volatile int16_t audio_chL=0;

#define DELAY_BUF_SIZE 24000

int16_t buffer[DELAY_BUF_SIZE];
int16_t i = 0;


void I2S_HANDLER(void) {   /****** I2S Interruption Handler*****/

int16_t delayed_sample;
int16_t audio_out_chL = 0;

audio_IN = i2s_rx();
audio_chL = (audio_IN & 0x0000FFFF);
audio_chR = ((audio_IN >>16)& 0x0000FFFF);

   delayed_sample = buffer[i];
   audio_out_chL = delayed_sample + audio_chL;
   buffer[i] = audio_chL;
   i = (i+1) % DELAY_BUF_SIZE;

audio_OUT = ((audio_chR<<16 & 0xFFFF0000)) + (audio_out_chL & 0x0000FFFF);
i2s_tx(audio_OUT);

}

int main(void)
{
  audio_init ( hz48000, mic_in, intr, I2S_HANDLER);

  while(1){}
}
```

Figure 9: Listing of program `delay_intr.c`

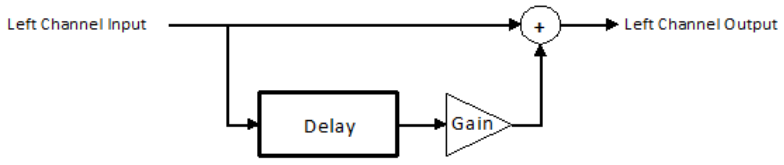

Figure 10: Block diagram of left channel delay.

9 Creating an Echo

An echo effect can be created by introducing a feedback network into the processing. The program `echo_intr.c`, shown in Figure 11, implements such an effect on the left audio channel. Study this example carefully. Experiment with different values of the parameters `DELAY_BUF_SIZE` and `GAIN`.

```
1  // echo_intr.c
2  #include "audio.h"
3
4  volatile int16_t audio_chR=0;
5  volatile int16_t audio_chL=0;
6
7  #define DELAY_BUF_SIZE 16000
8  #define GAIN 0.6f
9  int16_t buffer[DELAY_BUF_SIZE];
10 int16_t buf_ptr = 0;
11
12 void I2S_HANDLER(void)  {
13
14 int16_t delayed_sample;
15 int16_t audio_out_chL = 0;
16
17 audio_IN = i2s_rx();
18 audio_chL = (audio_IN & 0x0000FFFF);
19 audio_chR = ((audio_IN >>16)& 0x0000FFFF);
20
21     delayed_sample = buffer[buf_ptr];
22     audio_out_chL = delayed_sample + audio_chL;
23     buffer[buf_ptr] = audio_chL + delayed_sample*GAIN;
24     buf_ptr = (buf_ptr+1)%DELAY_BUF_SIZE;
25
26 audio_OUT = ((audio_chR<<16 & 0xFFFF0000)) + (audio_out_chL & 0x0000FFFF);
27 i2s_tx(audio_OUT);
28 }
29
30 int main(void)
31 {
32   audio_init ( hz48000, mic_in, intr, I2S_HANDLER);
33    while(1){}
34 }
```

Figure 11: Listing of program `echo_intr.c`

Question 1: Draw a block diagram of the system implemented by `echo_intr.c`.

Question 2: What would happen if the value of `GAIN` were made greater than or equal to 1?

Question 3: Sketch a time-domain plot showing what you think the unit impulse response of the system would be.

10 Comments, Suggestions, and Hints for using the Cypress FM4

- You will find 3.5mm stereo to male header breakout boards, included in the TSC Cypress FM4 kit, convenient for connecting the FM4's audio line in and headphont out jacks to an Analog Discovery board.
  - "T" connection is the Right audio channel
  - "R" connection is the Left audio channel
  - "S" connection is Ground
- You will find 3.5mm stereo to banana cables, available in TSC, convenient for connecting the FM4's audio line in and headphone out jacks to the laboratory equipment.
- Note in `main()` that the sample programs usually take audio from the "mic_in" port. Change this to "line_in" to use the line in port. This is suitable for receiving music from an audio player and from the function generator. Unpowered microphones operate at much lower voltages than line level devices.
- The line in jack on the FM4 should only receive a maximum of 894 mV peak-peak; this is the consumer audio line level. Monitor any signal that is put into this jack by using an oscilloscope. Adjust the level of the signal before connecting it to the line-in jack. To be safe, keep the voltage around 500 mV peak-peak.
- Please do not connect anything other than the supplied microphone to the FM4's mic in jack.
- The FM4 uses standard 3-conductor 3.5mm (1/8 inch) audio connectors. Some device-specific headsets (e.g., for cell phones) have a mic and either 1 or 2 audio output channels and might not behave exactly correctly on the headphone out jack of the FM4. For example, 2015 Apple Earpods (stereo with a mic, requiring 4 conductors) only seem to work properly when the pause button is depressed.
- You may find it helpful to launch a "Debug Session" within Keil for debugging your program. If you notice strange behavior when viewing variables in a "Watch" window, try changing the code optimization to "Level (0) -O0" (change by selecting `Project` → `Options for Target S6E2CC` → `C/C++`).

References

[1] FM4-176L-S6E2CC-ETH - ARM Cortex-M4 MCU Starter Kit Guide, March 26 2015.

[2] ARM University Worldwide Education Program. ARM-based Digital Signal Processing Lab-in-a-Box, 2014.