PortAudio Application Programming Interface (API)

We continue lectures on using the PortAudio API for real-time DSP using a PC and soundcard.

Creating a PortAudio Passcode

Here is a step-by-step procedure for creating a PortAudio passcode using Prof. De Leon's passcode files and DSP functions. The example is for Macintosh using XCode 3.1. Other procedures for Linux and Windows are very welcome.

1. Create New Project... -> Command Line Utility -> CoreFoundation Tool. Name the project as you wish.

2. We need to add the CoreAudio, AudioToolbox, AudioUnit, and CoreServices frameworks. In the left pane, click on External Frameworks and Libraries. Project -> Add to Project... You will find the above frameworks in /System/Library/Frameworks. These frameworks need not be copied into the destination. You can remove CoreFoundation.

3. We need to add DeLeon's portaudio passcode and DSPFixedPointFunctions folder. In the left pane, click on Source. Project -> Add to Project... Add the main.c, initialize_program.c, process-signal.c, user_data.h files. These files need to be copied into the destination. Note that these are the portaudio-specific files.

4. We need to add the portaudio lib libportaudio.a. In the left pane, click on External Frameworks and Libraries. Project -> Add to Project... You will find the libportaudio.a file in $PA/lib where $PA is the path to your PortAudio installation. This file need not be copied into the destination. Also, in order for your project to find the portaudio.h library, make sure Header and Library Search Paths include $PA.

5. Build and go. You should be able to speak into the microphone and hear output on the speakers. There may be feedback in which case, use headphones instead of speakers.

Review and Structure of TI DSK_APP

The passcode we have been using for the TI 6414DSK is based on TI's DSK_APP which is supplied with the board. The main file is dsk_app.c. Two key code blocks within dsk_app.c is the block which calls initialize_program()

```
#include "user_data.h"

/***************************************/
/* Global variable initializations here */
/***************************************/

void initialize_program()
{

}
```

and the block which calls process_signal()

```
if (pingPong == PING) {
    /* Toggle LED #3 as a visual cue */
    /* DSK6416_LED_toggle(3); */

    /* Process signal sample-by-sample */
    process_signal(gBufferRcvPing[0], gBufferRcvPing[1], &gBufferXmtPing[0],
&gBufferXmtPing[1]);

} else {
    /* Toggle LED #2 as a visual cue */
    /* DSK6416_LED_toggle(2); */

    /* Process signal sample-by-sample */
```

```
    process_signal(gBufferRcvPong[0], gBufferRcvPong[1], &gBufferXmtPong[0],
&gBufferXmtPong[1]);
}
```

If BUFSIZE > 2 we use the process_block() function which will not be discussed here. The process_signal() function in its default form is the familiar passthru or wire:

```
#include "user_data.h"

void process_signal(short inputRight, short inputLeft, short *outputRight, short
*outputLeft)
{
       /*********************************/
       /* Process right channel sample */
       /*********************************/
       *outputRight = inputRight;

       /*********************************/
       /* Process left channel sample */
       /*********************************/
       *outputLeft = inputLeft;

}
```

Prof. De Leon has written initialize_program.c, process_block.c, process_signal.c, user_data.h, and util.h files so that students may develop DSP codes in a consistent and straight-forward manner. The main TI code, dsk_app.c was modified to properly call these functions. Two other "main" programs have been written for convenience which call the functions in the same way that dsk_app.c does. The first reads/writes samples from/to text files while the second uses PortAudio to receive/transmit input/output samples from a PC with a sound card. The former program allows for easy analysis of code operation while the latter allows real-time processing without the need for a DSP board.

In addition, a number of DSP functions were written and supplied in DSPFunctionsFixedPoint. All of these codes (with the exception of dsk_app.c) have been developed for portability.

Differences in the File I/O Passcode and the PortAudio Passcode

The main.c code in the file I/O passcode simply reads/writes samples from/to text files.

```
int main (void)
{
    short inputRightSample, inputLeftSample, outputRightSample, outputLeftSample;
    FILE *inputFilePtr, *outputFilePtr;

    inputFilePtr = fopen("input.txt","r");        /* open input file for reading */
    if(inputFilePtr == NULL) {                    /* make sure the input file is there... */
      perror("Error");                            /* ...message... */
      exit(1);                                    /* ...and exit if not */
    }
    outputFilePtr = fopen("output.txt","w");    /* open output file for writing */

    /******************/
    /* Initialization */
    /******************/
    initialize_program();

    /*****************************************************************************/
    /* Main loop - process right, left input samples; get right, left output samples */
    /*****************************************************************************/
    while (fscanf(inputFilePtr, "%hd%hd", &inputRightSample, &inputLeftSample) != EOF) {
        process_signal(inputRightSample, inputLeftSample, &outputRightSample, &outputLeftSample);
        fprintf(outputFilePtr, "%hd\n%hd\n", outputRightSample, outputLeftSample);
    }

    fclose(inputFilePtr);   /* close input file */
    fclose(outputFilePtr);  /* close output file */
```

```
    return 0;
}
```

and the PortAudio callback function receives/transmits input/output samples from a PC with a sound card

```
/*****************************/
/* PortAudio Callback function */
/*****************************/
static int pa_callback(void *inputBuffer, void *outputBuffer,
                       unsigned long framesPerBuffer,
                       const PaStreamCallbackTimeInfo* timeInfo,
                       PaStreamCallbackFlags statusFlags,
                       void *userData )
{
    int i;
    short *pi, *po;
    short inputRightSample, inputLeftSample, outputRightSample, outputLeftSample;

    pi = (short*)inputBuffer;
    po = (short*)outputBuffer;
    for (i=0; i<framesPerBuffer; i++)
    {
        inputRightSample = *pi++;
        inputLeftSample = *pi++;
        process_signal(inputRightSample, inputLeftSample, &outputRightSample, &outputLeftSample);
        *po++ = outputRightSample;
        *po++ = outputLeftSample;
    }
    return 0;
}
```

Notice the call in both main.c codes is the same (and is the same in dsk_app.c):

```
    process_signal(inputRightSample, inputLeftSample, &outputRightSample, &outputLeftSample);
```

Demo of File I/O Passcode and the PortAudio Passcode

Demonstrations of the passcodes.

These following examples illustrate that one set of DSP routines can thus be written and compiled/executed using any of the three main codes (dsk_app.c, file I/O passcode, PortAudio passcode).

Example: FIR

```
#include "user_data.h"

void process_signal(short inputRight, short inputLeft, short *outputRight, short *outputLeft)
{
    /*******************************/
    /* Process right channel sample */
    /*******************************/
    outputRight = cfir(FILTERORDER, coeffs, states, &oldestStatePtr, inputRight);

    /*******************************/
    /* Process left channel sample */
    /*******************************/
    *outputLeft = inputLeft;
}
```

Example: Wavetable Synthesizer

```
#include "user_data.h"

void process_signal(short inputRight, short inputLeft, short *outputRight, short *outputLeft)
{
    /*******************************/
    /* Process right channel sample */
    /*******************************/
    *outputRight = lookup_waveIntDelta(TABLELENGTH, waveTable, delta, &offset);

    /*******************************/
    /* Process left channel sample */
    /*******************************/
    *outputLeft = inputLeft;
}
```

Example: Moorer Sound Field Simulator

```
#include "user_data.h"

void process_signal(short inputRight, short inputLeft, short *outputRight, short *outputLeft)
{
    short TDL_Output, CF1_Output, CF2_Output, CF3_Output, CF4_Output,
                  CF5_Output, CF6_Output, APF_Input, APF_Output, reverbOutput;
    int accumCF_Output=0;

    /*******************************/
    /* Process right channel sample */
    /*******************************/
    TDL_Output = tdl(TDLD, TDL, &TDL_OldestStatePtr, NUMTAPS, TDL_Gains, TDL_Spacing, inputRight);

    CF1_Output = comb2(CF1D, CF1, &CF1_OldestStatePtr, CF1G1, CF1G2, TDL_Output);

...

    APF_Output = allpass(APFD, APF, &APF_OldestStatePtr, APFG, APF_Input);

    /* Delay reverberator output to align with TDL */
...

    *outputRight = cround(GEARLY*TDL_Output + GLATE*reverbOutput)>>15;

    /*******************************/
    /* Process left channel sample */
    /*******************************/
    *outputLeft = inputLeft;
}
```