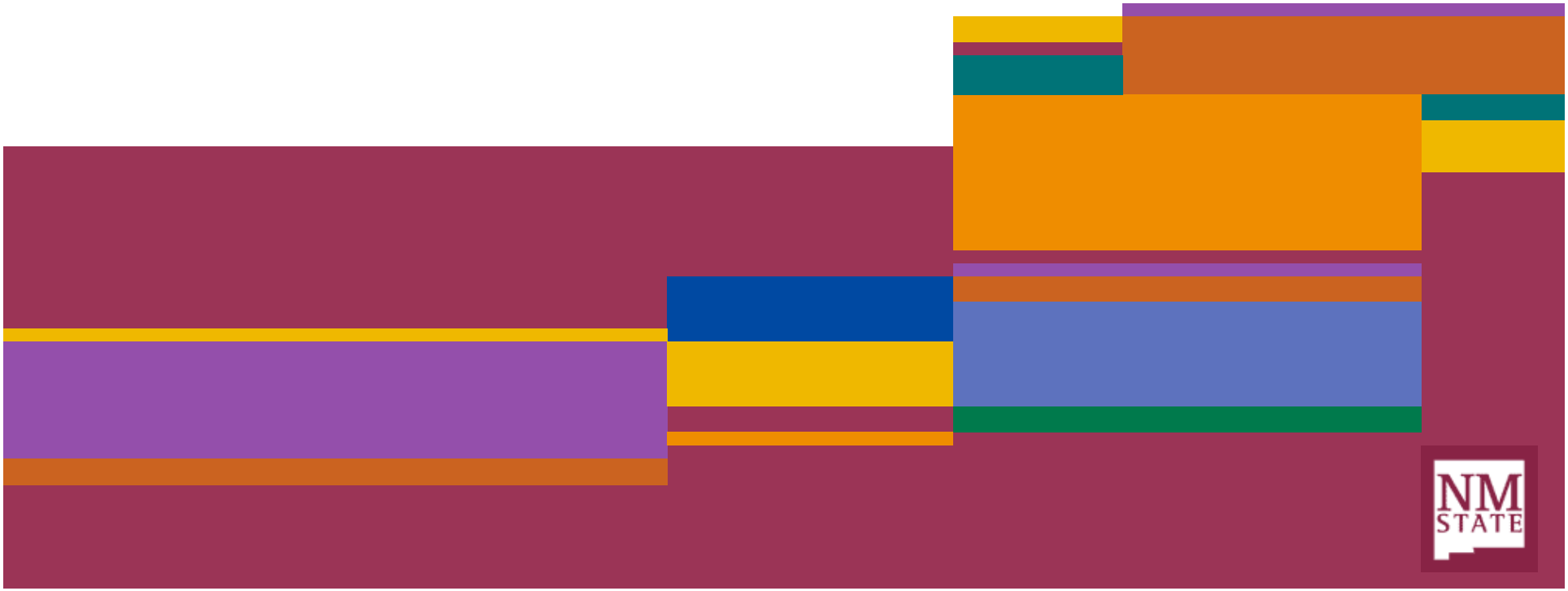
6416DSK Pass Code: DSK_APP



6416DSK Pass Codes on Installation CD

- DSK_APP
  - Digitally processes audio data from line in on AIC23 codec and outputs result on line out. Uses McBSP (bi-directional serial port) and EDMA (enhanced direct memory access) to efficiently handle data transfer without intervention from DSP.
  - DSK_APP is composed of several files (described shortly)
  - **We will use the DSP_APP as the pass code in this course.** However, it has been significantly hacked to allow us an easier transition from the Freescale platform and while allowing more flexibility for course projects.
- SWI_AUDIO
  - Demonstrates how an application can use a codec mini-driver via SIO module in SWI (software interrupt) threads. Audio is read from an input SIO, then sent back out on an output SIO. Application is configured to use DIO (found in CCS lib)

6416DSK Passcodes on Installation CD (cont'd)

- TSK_AUDIO
  - Demonstrates how an application can use a codec mini-driver via SIO module in TSK (task) threads. Audio is read from an input SIO, then sent back out on an output SIO. Application is configured to use DIO (found in CCS lib)

- PIP_AUDIO
  - Demonstrates how an application can use a codec mini-driver via PIP (pipe) module in SWI threads. Audio is read from an input PIP, then sent back out on an output PIP. Application is configured to use PIO adapter (found in this local audio directory)

File Descriptions for DSK_APP

- aic23.c
  - AIC23 codec driver implementation
  - Specific to the Spectrum Digital DSK6416 board
- aic23.h
  - Header file for AIC23 codec driver
  - Contains control word bit-definition macros and declaration of public functions for such things as sampling rate
  - Includes AIC23 control register definitions (10 control registers each 9 bits wide); address of a control register is 7 bits wide. Together, address +content form a 16-bit control word
- build.bat
  - This file will builds the dsk_app project (like a UNIX makefile)

File Descriptions for DSK_APP

- cc_build_Debug.log
  - Generated by CCS. Gives build information such as errors, warnings, etc
- Debug
  - Folder containing among other things downloadable object code
- dsk_app.c
  - Main code for DSK_APP. This code initializes individual DSP/BIOS modules and the main() function is called as the main user thread.
  - The main() function performs application initialization and starts the EDMA data transfers. When main exits, control passes back entirely to DSP/BIOS which services any interrupts or threads.
- dsk_app.cdb
  - Initialization for individual DSP/BIOS modules-requires DSP/BIOS config tool

File Descriptions for DSK_APP (cont'd)

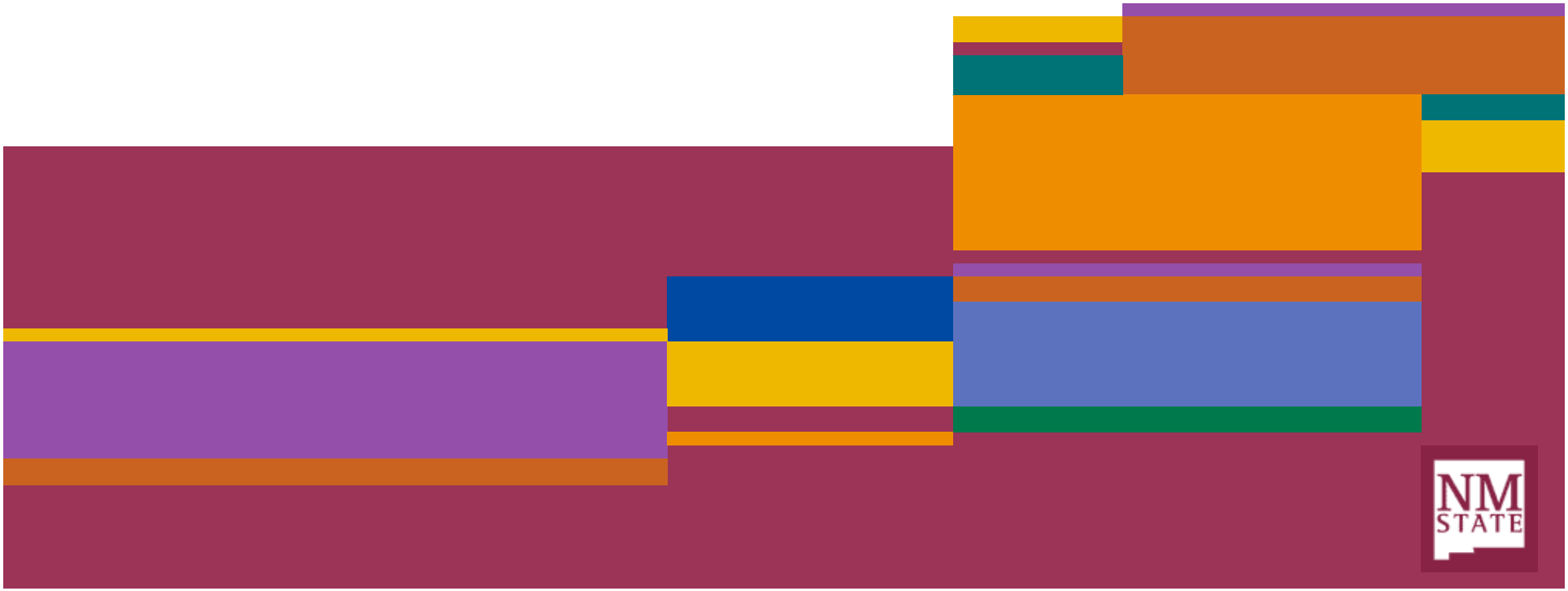
- dsk_app.pjt
  - Project file for CCS which lists project settings, source files, generated files, and other settings. File is autogenerated from CCS.
- dsk_appcfg.h
  - At compile time, CCS will auto-generate DSP/BIOS related files based on DSP/BIOS module settings. This header file contains results of autogeneration and must be included for proper operation.
- dsk_appcfg_c.c, dsk_appcfg.cmd, dsk_appcfg.h62, dsk_appcfg.s62
  - These files are generated by Configuration tool

File Descriptions for DSK_APP (cont'd)

- build-dsk_app.tcf, Debug.lkf, dsk_app.paf, dsk_app.udc
  - The function of these files is unknown
- Students should open dsk_app.c and examine:
  - file header (description)
  - main() function
  - processBuffer() function

  including DeLeon's modifications

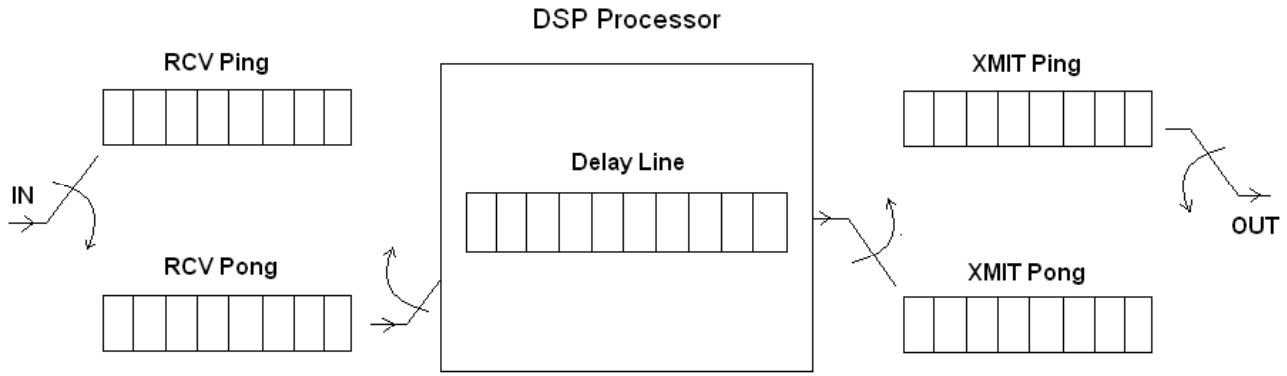
Moving Audio Data (Samples) to/ from DSP core from/to AIC23 Codec



Data Transfers (Samples) in DSK_APP

- McBSP1 is used to control/configure AIC23
  - Codec receives serial commands through McBSP1 that set configuration parameters such as sample rate, sample resolution (bits), etc.
- Audio data is transferred to/from codec through McBSP2
  - EDMA is configured to take every 16-bit audio sample arriving on McBSP2 and store it in a buffer until it can be processed
  - Once it has been processed, EDMA controller sends data back to McBSP2 for transmission
- DSK_APP uses two special techniques to make data transfers more convenient and efficient:
  - 1) Ping-pong data buffering in memory
  - 2) Linked EDMA transfers

Ping-Pong Buffering



- RCV PING buffer receives R/L input samples from codec while XMIT PING buffer transmits R/L output samples to codec
- Simultaneously, we process prior input samples in RCV PONG buffer and move output samples to XMIT PONG buffer
- Once RCV PING buffer is filled with new samples (and XMIT PING buffer is emptied), RCV PONG buffer samples will have been processed and output copied to XMIT PONG buffer
- We flip back and forth between the two pairs of buffers or "Ping-Pong" between the two pairs

Ping-Pong Buffering (cont'd)

- If BUFF_SIZE = 2 (R/L samples), pass code does **sample-by-sample processing**

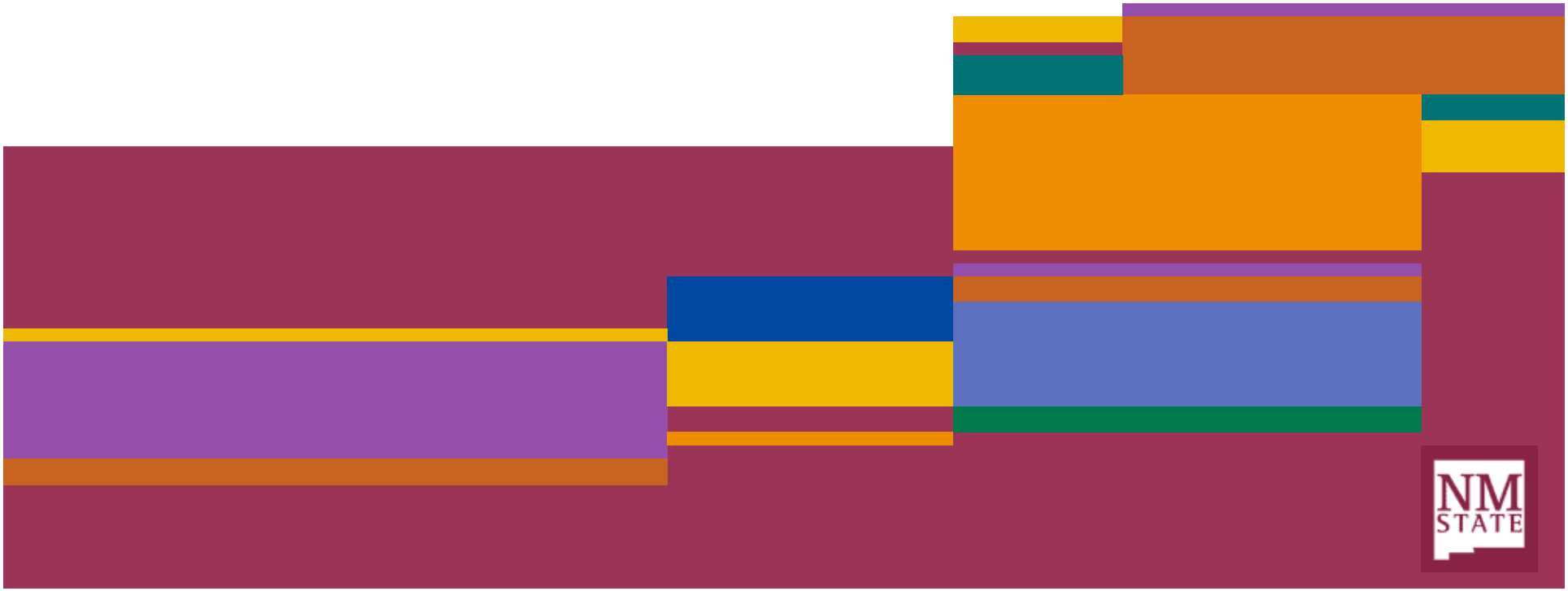
- If BUFF_SIZE > 2, pass code supports applications requiring **block processing** such as those that require an FFT
  - For block sizes greater than two, the right and left samples are interleaved in the block, i.e. right, left, right, left, ...

Linked EDMA transfers

- EDMA is configured to alternately fill PING buffer then PONG buffer
  - When a buffer is filled, EDMA controller generates an interrupt
  - Interrupt handler must reload configuration (ping or pong information) for next buffer before next audio sample arrives
  - EDMA controller automatically links to next configuration, i.e. linked transfers, when current configuration is finished
- Linked transfers simplify life!
  - Programmer does not need to keep track of which buffer to process
  - Correct buffer is automatically passed to process_signal() function
  - Only requirement is that samples **must** be processed before active buffer fills up (otherwise we drop samples)

Modifications to DSK_APP

(if it ain't broke...)



Modifications to DSK_APP

- We make several modifications to DSK_APP code
  - More user-friendly pass code
  - Smooth transition from Freescale Modified Pass Pack to TI DSP_APP
  - Professional-grade pass code for further development

- Modifications are made through changes in dsp_app.c as well as with the addition of four files:

  initialize_program.c, process_block.c, process_signal.c, user_data.h

  and one set of DSP C routines contained in DSPFunctionsFixedPoint:

  ccan.c, cdelay.c, cfir.c, cround.c, impulse.c, lookup_wave.c, lookup_waveIntDelta.c, tap.c, tdl.c, util.h, wrap.c

USER_DATA.H

- We include a user_data.h file which partially serves to replicate the pass.dat file in the Freescale Pass Code.
  - This file declares the initialize_program() and process_signal()/ process_block() routines and all global variables
  - It also includes the DSP utility file util.h (described below)
- For the projects developed in this course, we

  #define BUFFSIZE 2

  (R/L sample) so that we process sample-by-sample as in the Freescale passcode
  - All processing code will reside in process_signal.c.
  - For final projects involving FFT, BUFFSIZE>2, process_block.c

USER_DATA.H

```
#ifndef USER_DATA
#define USER_DATA

void initialize_program();
void process_signal(short inputRight, short inputLeft, short *outputRight, short *outputLeft);
void process_block(short *inputRight, short *inputLeft, short *outputRight, short *outputLeft);

/* BUFSIZE is number of samples in buffer. For sample-by-sample processing set to 2 (right, left) *
/* For block processing, set to 2N for N interleaved pairs right, left, right, left, ... */
#define BUFFSIZE 2

/* Global variables here as extern, initializations in initialize_program.c */

#include "DSPFunctionsFixedPoint/util.h"

#endif
```

UTIL.H

- This file declares all given *fixed-point* DSP routines needed in EE442/EE592 as well additional routines the student develops
- It is assumed these files reside in the folder **DSPFunctionsFixedPoint** and will be described in detail in the next lecture
- In addition, sampling rate definitions are defined here for use in the dsk_app.c code
  - These hex equivalents are found in “aic23.h” file starting on line 115 and are given in 9-bit format.

UTIL.H

```
#ifndef UTIL_H
#define UTIL_H

/* Sampling rate definitions in tlv320aic23.pdf (and in aic23.h) */
/* See "Codec configuration settings" in dsk_app.c */
#define DSKAPP_AIC23_8KHZ     0x000d
#define DSKAPP_AIC23_16KHZ    0                   /* not supported */
#define DSKAPP_AIC23_24KHZ    0                   /* not supported */
#define DSKAPP_AIC23_32KHZ    0x0019
#define DSKAPP_AIC23_44KHZ    0x0023
#define DSKAPP_AIC23_48KHZ    0x0001
#define DSKAPP_AIC23_96KHZ    0x001d

short ccan(short M, short *a, short *b, short *w, short **p, short x);
void cdelay(short D, short *w, short **p);
short cfir(short M, short *h, short *w, short **p, short x);
int cround(int a);
short impulse();
short lookup_wave(short L, const short *table, int delta, short *intOffset, unsigned short *UfracOf
short lookup_waveIntDelta(short L, const short *table, short delta, short *offset);
short tap(short M, short *w, short *p, short i);
short tdl(short M, short *w, short **p, short N, short *gain, short *delta, short x);
void wrap(short M, short *w, short **p);

#endif
```

INITIALIZE_PROGRAM.C

- Within dsp_app.c code, we place a call to initialize_program() in main ()
  - Routine can be found in initialize_program.c file and is to be appropriately coded by the developer
  - Routines serves to mimic the proginit.asm routine in the Freescale code

```
#include "user_data.h"

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

void initialize_program()
{

}
```

processBuffer()

- The processBuffer() routine in dsk_app.c replaces copyData() to more closely resemble the Freescale main event loop which does sample-by-sample processing

- In addition, the modification also gives us an easy option for block processing

processBuffer()

- If BUFFSIZE = 2, we have in the routine

```
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]);
}
```

processBuffer()

- If BUFFSIZE > 2, we have PING (also a PONG part):

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

    /*********************************/
    /* De-interleave into L/R blocks */
    /*********************************/
    for(i=0; i<BUFFSIZE/2; i++) {
        j = 2*i;
        inputRight[i] = gBufferRcvPing[j];
        inputLeft[i] = gBufferRcvPing[j+1];
        }

    /* Process signal block-by-block */
    process_block(inputRight, inputLeft, outputRight, outputLeft);

    /************************/
    /* Interleave L/R blocks */
    /************************/
    for(i=0; i<BUFFSIZE/2; i++) {
        j = 2*i;
        gBufferXmtPing[j] = outputRight[i];
        gBufferXmtPing[j+1] = outputLeft[i];
    }
```

PROCESS_SIGNAL.C

- Within the processBuffer() routine, if BUFFSIZE = 2, we place a call to the process_signal() routine which can be found in the process_signal.c file

```
#include "user_data.h"

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

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

}
```

PROCESS_BLOCK.C

- Within the processBuffer() routine, if BUFFSIZE > 2, we place a call to the process_block() routine

```
#include "user_data.h"

void process_block(short *inputRightBlock, short *inputLeftBlock, short *outputRightB
{
short i;

/*********************************/
/* Process right channel block */
/*********************************/
for(i=0; i<BUFFSIZE/2; i++)
    outputRightBlock[i] = inputRightBlock[i]; /* simple I/O copy */

/*********************************/
/* Process left channel block */
/*********************************/
for(i=0; i<BUFFSIZE/2; i++)
    outputLeftBlock[i] = inputLeftBlock[i]; /* simple I/O copy */
}
```

Modified DSK_APP

- A ZIP file containing the DSK_APP pass code with modifications is available at

`http://www.ece.nmsu.edu/~pdeleon/EE592/TI_6416_Code.html`