Appendix F: Software Listing

```
/*********************************************************
*
*   477grp4 -- Digital Sheet Music Reader and Player.
*
*  Software listing broken up into files as they are seperated
*    in our project.  Each file is seperated with a header
*     just like this one. Most of the comments explain the
*     specifics of the functions in each file.  Mayhem.c is the
*     Main funtion.  There is a definate layer set up to
*     our code
*
*********************************************************/


/*******************************************************
*
*  Initialize.c -- Sets Up Peripherals
*
*
*******************************************************/

#include "Initialize.h"

void init(void){

/////////////
// SPI Initialization
/////////////

         /* Init SPI MASTER TX */
         *(volatile int *)SPICTL = 0;
         *(volatile int *)SPIFLG = 0;

         /* set the SPI baud rate to CCLK/4*64 (781.25KHz @ 200MHz)*/
         *(volatile int *)SPIBAUD = 0x1388;

         /* set up SPIFLAG registers */
         *(volatile int *)SPIFLG = DS1EN;

         /* Set up DAG registers */
         *(volatile int *)SPICTL = DMISO|  /* Disable MISO on transfers */
                  WL32|   /* 32-bit words */
```

```
                    SPIMS|  /* Master mode (internal SPICLK) */
                    SPIEN|  /* Enable SPI port */
                    TIMOD1| /* Initialize SPI port to begin
                     transmitting when DMA is enabled */
                    MSBF|     /* MSB sent first */
                    CLKPL;    /* Active low clock polarity */


////////////////
//  Interrupt Initializations
//////////////////

    *pDAI_IRPTL_PRI = SRU_EXTMISCB1_INT | SRU_EXTMISCB2_INT | //unmask individual interrupts
                      SRU_EXTMISCB3_INT | SRU_EXTMISCB4_INT |
                      SRU_EXTMISCA0_INT;
    *pDAI_IRPTL_RE = SRU_EXTMISCB2_INT;   //make sure interrupts latch on the rising edge
    *pDAI_IRPTL_FE = SRU_EXTMISCB1_INT | SRU_EXTMISCB3_INT |
                     SRU_EXTMISCB4_INT | SRU_EXTMISCA0_INT; // Falling edges for RPG signal

        //assign pin buffer 04 low so it is an input
    SRU(LOW,DAI_PB04_I);
    SRU(LOW,DAI_PB01_I);
    SRU(LOW,DAI_PB02_I);
   // SRU(LOW,DAI_PB07_I);

//Route MISCB singnals in SRU_EXT_MISCB (Group E)

    //route so that DAI pin buffer 04 connects to MISCB1 (Falling Edges)
    SRU(DAI_PB04_O,MISCB1_I);

    //route so that DAI pin buffers 01,02,04 connect to MISCB2-4 (Rising Edges)
   SRU(DAI_PB04_O,MISCB2_I);

   SRU(DAI_PB01_O,MISCB3_I);
   SRU(DAI_PB02_O,MISCB4_I);
  // SRU(DAI_PB07_O,MISCA0_I);

//Pin Buffer Disable in SRU_PINEN0 (Group F)

    //assign pin 01,02,04 low so they are inputs
    SRU(LOW,PBEN04_I);
    SRU(LOW,PBEN02_I);
    SRU(LOW,PBEN01_I);
   // SRU(LOW,PBEN07_I);
```

```
   // Output signals for resets on USB and MIDI
   // SRU(HIGH,PBEN09_I);
   // SRU(HIGH,PBEN10_I);

}


/*********************************************************
*
*  Mayhem.c -- Main function
*
*
*********************************************************/



#include <21262.h>
#include <def21266.h>

#include "menu.h"
#include "Initialize.h"
#include "ISR.h"
#include "IMS_Build.h"
//#include "Middleware_C_Simple/PlayEgn.h"
//#include "Middleware_C/ComDrv.h"
//#include "Middleware_C/DevDrv.h"
//#include "Middleware_C/SMFDecoder.h"
//#include "Middleware_C_Simple/W56964_Driver.h"
#include "IMS_MIDI.h"
#include "FileFunctions.h"
#include "FatFiles/fatFxns.h"
#include "USB_Driver.h"

/* Main code section */

#pragma section("seg_sram", DMAONLY)
unsigned int IMS[100] = {0};

#pragma section("seg_sram", DMAONLY)
unsigned int MIDI[1000] = {0};

#pragma section("seg_sram", DMAONLY)
unsigned int USB[512] = {0};

#pragma section("seg_sram", DMAONLY)
```

```
unsigned int BK_Buffer[512] = {0};


char googleplex = 0 ;  // WILL's DEBUG char
main(){

   int i, nRet = 0, ctr = 0, rctr = 0, lctr = 0, currMenu = 0, NoOpts = 0, level = 0, numOpts = 0;
   extern int right,left,select,back;
   extern int IMSSz;
   char ** temp;
   int header[6];
       int hold = 0, tempo = 0, mspqn = 0, divisor = 60000000;
   extern unsigned int * pex_IMS;

   ////////////////////////////////
   //
   // USB Fat file variables
   //
   ////////////////////////////

   extern unsigned long FAT_RootDirStartSector;
   FatFile findFile;
   char * imageFile = "bwscan.bmp";

   // The complier reserves space in the declaration, so all the menus
   // needed to be 40 characters to correspond to the two lines
   // they would be printed on
   char * mainMenu[4] = {"  Image Processing                      ",
     "   IMS Processing                       ",
     "  Play a MIDI file                      ",
     "   LCD Formatting                       "};

   char *LCDsubs[2] = {"  Adjust Contrast                      ",
     "  Adjust Backlight                      "};

   char * subMenu[3] = {"   Select an Image                      ",
     "    Select an IMS                       ",
     "  Select a MIDI file                    "};

   // This is where I append the menu strings to include the
   // various characters I want them to display on the second line
   // this is why the second index is >20

   mainMenu[0][25] = 0x90; //single eigth note
   mainMenu[0][26] = 0x91; //double eigth note
```

```
mainMenu[0][27] = 0x90;
mainMenu[0][28] = 0x20;
mainMenu[0][29] = 0xdf; //right arrow
mainMenu[0][30] = 0x20;
mainMenu[0][31] = 0x30; //0
mainMenu[0][32] = 0x31; //1
mainMenu[0][33] = 0x30; //0
mainMenu[0][34] = 0x31; //1
mainMenu[0][35] = 0;

mainMenu[1][25] = 0x30; //0
mainMenu[1][26] = 0x31; //1
mainMenu[1][27] = 0x30; //0
mainMenu[1][28] = 0x31; //1
mainMenu[1][29] = 0x20;
mainMenu[1][30] = 0xdf; //right arrow
mainMenu[1][31] = 0x20;
mainMenu[1][32] = 0x2e; //.
mainMenu[1][33] = 0x6d; //m
mainMenu[1][34] = 0x69; //i
mainMenu[1][35] = 0x64; //d
mainMenu[1][36] = 0;

mainMenu[2][25] = 0x91; // double eitgth note
mainMenu[2][26] = 0x20;
mainMenu[2][27] = 0x90;
mainMenu[2][28] = 0x20;
mainMenu[2][29] = 0x91;
mainMenu[2][30] = 0x20;
mainMenu[2][31] = 0x90;
mainMenu[2][32] = 0x20;
mainMenu[2][33] = 0x91;
mainMenu[2][34] = 0;

mainMenu[3][25] = 0xfa; //left bracket
mainMenu[3][26] = 0xc4; //underscore
mainMenu[3][27] = 0xc4;
mainMenu[3][28] = 0xc4;
mainMenu[3][29] = 0xc4;
mainMenu[3][30] = 0xc4;
mainMenu[3][31] = 0xc4;
mainMenu[3][32] = 0xc4;
mainMenu[3][33] = 0xfc; //right bracket
mainMenu[3][34] = 0;
```

```
 LCDsubs[0][27] = 0xd6;
 LCDsubs[0][28] = 0xd7;
 LCDsubs[0][29] = 0xd8;
 LCDsubs[0][30] = 0xd9;
 LCDsubs[0][30] = 0xda;
 LCDsubs[0][31] = 0;

 LCDsubs[1][25] = 0x24;
 LCDsubs[1][26] = 0x20;
 LCDsubs[1][27] = 0x24;
 LCDsubs[1][28] = 0x20;
 LCDsubs[1][29] = 0x24;
 LCDsubs[1][30] = 0x20;
 LCDsubs[1][31] = 0x24;
 LCDsubs[1][32] = 0x20;
 LCDsubs[1][33] = 0x24;
 LCDsubs[1][34] = 0;

init();
interrupt(SIG_DAIH,DAIroutine);
outchar(HIDE_CURSOR);
outchar(CLEAR_LCD);

print("     The Digital    ");
print(" Sheet Music Reader ");
print("  ");
outchar(0x90);
print("  ");
print("and Player");
print("  ");
outchar(0x90);
print("       ");
outchar(0x91);
print("   ");
outchar(0x90);
print("   ");
outchar(0x91);

sl811h_init();
while(1)
  {
        slave_detect();
        if(SLAVE_FOUND)
        {
                 break;
```

```
        }
    }



  /*
  header[0] = 120; //tempo
  header[1] = -4; //key sig
  header[2] = 4; //time sig top
  header[3] = 4; //time sig bot
  header[4] = 56; //MIDI patch#
  header[5] = 1; //cleff


 ////////////////////////////////////////
 //
 //  Waits for USB interupt...right now for select pb
 //
 ////////////////////////////////////////


  while (select == 0){ }
  select = 0;

  MainMenuTemplate();

  temp = mainMenu;
  numOpts = 3;
  NoOpts = 0;
  currMenu = 0;
  level = 0;
  printMenu(temp[currMenu]);



 ////////////////////////////////////////
 //
 //  Main Loop Starts Here.
 //
 ////////////////////////////////////////

while(1){ // Polls through the 4 user input options (right, left, select, back)

  if (left == 1){// Scrolls left if there are options to scroll
```

```
  lctr++;
  left=0;
    if (lctr == 2){
      lctr = 0;
      left = 0;
      if (NoOpts == 1){
      }
      else if (currMenu == 0){
        currMenu = numOpts;
        printMenu(temp[numOpts]);
      }
      else{
        currMenu--;
        printMenu(temp[currMenu]);
      }
    }
}

else if (right == 1){ //scrolls right if there are options to scroll
  rctr++;
  right=0;
    if (rctr == 2){
      rctr = 0;
      right = 0;
      if (NoOpts == 1){
      }
      else if (currMenu == numOpts){
        currMenu = 0;
        printMenu(temp[0]);
      }
      else{
        currMenu++;
        printMenu(temp[currMenu]);
      }
    }
}

////////////////////////////////////////
//
//  Checks if there is a select button hit
//
////////////////////////////////////////

else if (select == 1){
  select = 0;
```

```
if (level == 0){ // Level of menu //
  level = 1;

  MenuTemplate();

  //Could change this structure a litttle bit depending on how we
  //  decide to handle file names.  There really doesn't need to be a sub level
  //  that prints what you already said you wanted to do.  We could just use this
  //  part to call the specific functions from level zero and make the LCD submenu
  //  level one.  But again I have to wait and see what the deal is with file names.

  if (currMenu == 3){
    level = 2;
    //want to go to LCD submenu options
    temp = LCDsubs;
    numOpts = 1;
    currMenu = 0;
    printMenu(temp[currMenu]);
  }
  else if (currMenu == 0) {
    NoOpts = 1;
    printMenu(subMenu[0]);
  }
  else if (currMenu == 1) {
    NoOpts = 1;
    printMenu(subMenu[1]);
  }
  else if (currMenu == 2) {
    NoOpts = 1;
    printMenu(subMenu[2]);
  }

}

else if (level == 1){
 //Function calls to handle various file selection tasks
  if (currMenu == 0) {
   //header = ImageSpecs();
   /*pIMS = IMS; //Each time you want to make an IMS set the pointer back to the
                 // the start of the block
   header[0] = 120;
   header[1] = 1;
   header[2] = 4;
   header[3] = 4;
```

```
   header[4] = 36;
   header[5] = 1; */
   IMS_Build(header);
   read_extmem(dest,IMS,1);

   ImageProcessing(header);
  }
  else if (currMenu == 1) {

    FileNameTemplate();
    printMenu("CALL IMS FILENAME FUNCTION");

    while (back == 0){
    }

  }
  else if (currMenu == 2) {

    FileNameTemplate();
    printMenu("CALL MIDI FILE FUNCTION");

    while (back == 0){
    }
  }

  back = 0;
  select = 0;
  level = 1;
  MenuTemplate();
  printMenu(subMenu[currMenu]);

}

else if (level == 2){

  if (currMenu == 0){
    AdjustContrast();
  }
  else if(currMenu == 1){
    AdjustBacklight();
  }

  back = 0;
  select = 0;
  level = 2; /* stay in same level after returning from function */
```

```
      MenuTemplate();
      printMenu(LCDsubs[currMenu]);

    }

  }

//////////////////////////////////////////
//
//  Checks if there is a back button hit
//
//////////////////////////////////////////

  else if (back == 1) {
    back = 0;

    if (level == 0){
    }
    else if (level == 1){
      NoOpts = 0;
      level = 0;
      temp = mainMenu;
      numOpts = 3;
      MainMenuTemplate();
      printMenu(mainMenu[currMenu]);
    }
    else if (level == 2){
      level = 0;
      NoOpts = 0;
      temp = mainMenu;
      numOpts = 3;
      currMenu = 3;
      MainMenuTemplate();
      printMenu(mainMenu[currMenu]);
    }
  }



} // End of Menu Polling loop.

} //End of Main

/*******************************************************
*
```

```
*  menu.c -- LCD Menu helper functions
*
*
************************************************************/

#include "menu.h"
#include "FileFunctions.h"

   void outchar(int message)
{

    *(volatile int *)TXSPI=message;
}

void print(char message[])
{
    int i,w;

    for(i=0; i<strlen(message); i++)
          {
          for(w=0; w<65000; w++)
          {
          w=w;
          }
      *(volatile int *)TXSPI=message[i];
          }

}


//////////////////////////////////////////////////////
//
// Function for printing the menu on the middle two lines
//
//////////////////////////////////////////////////////

void printMenu(char message[])
{
    outchar(0x110001);
    print("                                          ");
    outchar(0x110001);
    print(message);
}

void printLine2(char message[])
```

```
{
    outchar(0x110001);
    print("                    ");
    outchar(0x110001);
    print(message);
}

void printLine3(char message[])
{
    outchar(0x110002);
    print("                    ");
    outchar(0x110002);
    print(message);
}

//////////////////////////////////////////////////////
//
//  Contrast & Backlight functions, adjust value based
//    on rotating RPG
//
//////////////////////////////////////////////////////

void AdjustContrast(void)
{
  extern int back, right,left;
  int val = 50, i,rctr = 0,lctr = 0;
  char str[10];
  LCDMenuTemplate();
  printMenu("Adjusting Contrast  Current Value = 50");
  *(volatile int *)TXSPI=(0x0F00 | val);

  while (1){

    if (left == 1){
      lctr++;
      left=0;
        if (lctr == 2){
          lctr = 0;
          left = 0;
          if (val == 0){

          }
          else{
            val--;
            i = sprintf(str, "%d", val);
```

```
       *(volatile int *)TXSPI=(0x0F00 | val);

        if (val == 99){
        outchar(0x00080808); // Backspace three spaces
        print(str);
        }
        else if (val < 10) {
        outchar(0x00000808); // Backspace two spaces
        print(" ");  // Print a space
        print(str);
        }
        else {
        outchar(0x00000808); // Backspace two spaces
        print(str);
        }
      }
    }
}

if (right == 1){
  rctr++;
  right=0;
    if (rctr == 2){
      rctr = 0;
      right = 0;
      if (val == 100){

      }
      else{
        val++;
        i = sprintf(str, "%d", val);
        *(volatile int *)TXSPI=(0x0F00 | val);

        if (val < 10) {
          outchar(0x00000808); // Backspace two spaces
          print(" "); // Print a space
          print(str);
        }
        else {
          outchar(0x00000808); // Backspace two spaces
          print(str);
        }
      }
    }
}
```

```
    if (back == 1){
    back = 0;
    return;
    }
  }

}


void AdjustBacklight(void)
{
  extern int back, right,left;
  int val = 50, i,rctr = 0,lctr = 0;
  char str[10];
  LCDMenuTemplate();
  printMenu("Adjusting Backlight  Current Value = 50");
  *(volatile int *)TXSPI=(0x0E00 | val);

  while (1){

    if (left == 1){
      lctr++;
      left=0;
        if (lctr == 2){
          lctr = 0;
          left = 0;
          if (val == 0){

          }
          else{
            val--;
            i = sprintf(str, "%d", val);
           *(volatile int *)TXSPI=(0x0E00 | val);

            if (val == 99){
            outchar(0x00080808); // Backspace three spaces
            print(str);
            }
            else if (val < 10) {
            outchar(0x00000808); // Backspace two spaces
            print(" ");
            print(str);
            }
            else {
            outchar(0x00000808); // Backspace two spaces
```

```
            print(str);
            }

          }
        }
    }

    if (right == 1){
      rctr++;
      right=0;
        if (rctr == 2){
          rctr = 0;
          right = 0;
          if (val == 100){

          }
          else{
            val++;
            i = sprintf(str, "%d", val);
            *(volatile int *)TXSPI=(0x0E00 | val);

            if (val < 10) {
              outchar(0x00000808); // Backspace two spaces
              print(" ");
              print(str);
            }
            else {
              outchar(0x00000808); // Backspace two spaces
              print(str);
            }
          }
        }
    }
    if (back == 1){
      back = 0;
      return;
    }
  }

}

//////////////////////////////////////////////////
//
// Functions to print the top and bottom line
//   vary depending on where in the menu you are
```

```
//
////////////////////////////////////////////////////

void MainMenuTemplate(void)
{
    int w;
  outchar(0x0ce1);
  print("   ");
  print("More Options");
  print("   ");
  outchar(0xdf110003);
  print(" ");
  for(w=0; w<65000; w++){
    w=w;
  }
  outchar(0xfa);
  print("Select");
  outchar(0xfc);
  for(w=0; w<65000; w++){
    w=w;
  }
}

void MenuTemplate(void)
{
    int w;
  outchar(0x0ce1);
  print("   ");
  print("More Options");
  print("   ");
  outchar(0xdf110003);
  print(" ");
  outchar(0x20fa);
  print("Select");
  outchar(0xfc);
  print("    ");
  outchar(0xfa);
  print("Back");
  outchar(0xfc);
  for(w=0; w<65000; w++){
    w=w;
  }
}

void ImageTemplate(void)
```

```
{
    int w;
  outchar(0x0ce1);
  print("   ");
  print("More Options");
  print("   ");
  outchar(0xdf110003);
  print(" ");
  outchar(0x20fa);
  print("Select");
  outchar(0xfc);
  print("  ");
  outchar(0xfa);
  print("Cancel");
  outchar(0xfc);
  for(w=0; w<65000; w++){
    w=w;
  }
}

void FileNameTemplate(void)
{
    int w;
  outchar(0x0ce1);
  print("   ");
  print("Choose a file");
  print("  ");
  outchar(0xdf110003);
  print(" ");
  outchar(0x20fa);
  print("Select");
  outchar(0xfc);
  print("    ");
  outchar(0xfa);
  print("Back");
  outchar(0xfc);
  for(w=0; w<65000; w++){
    w=w;
  }
}

void LCDMenuTemplate(void)
{
  int w;
  outchar(0x0ce1);
```

```
  print("   ");
  print("Change Value");
  print("   ");
  outchar(0xdf110003);
  print("             ");
  outchar(0xfa);
  print("Back");
  outchar(0xfc);
  for(w=0; w<65000; w++){
    w=w;
  }
}
////////////////////////////////////////////////////////////
//
//  Function that gives musical options for an image
//   and creates the beginning of the IMS file
//
////////////////////////////////////////////////////////////
int * ImageSpecs(void)
{
  int i, nRet = 0, ctr = 0, rctr = 0, lctr = 0, currMenu = 0, currSpec = 0, numOpts = 0;
  extern int right,left,select,back;

  int header[6];
  char ** temp;


  char str[5];


  char * specs[6] = {"    Select a Clef   ",
                     "    Select a Key    ",
                     " Time Signature Top ",
                     "  Time Sig. Bottom  ",
                     "  MIDI Instrument   ",
                     "   Select a Tempo   "};

  char * clefSub[2] = {"      Bass Clef     ",
                       "     Treble Clef    "};

  char * keySub[15] = {" C Major (no sharps)",
                       " F Major - (1 flat) ",
                       "Bb Major - (2 flats)",
                       "Eb Major - (3 flats)",
                       "Ab Major - (4 flats)",
```

```
                    "Db Major - (5 flats)",
                    "Gb Major - (6 flats)",
                    "Cb Major - (7 flats)",
                    "G Major - (1 sharp) ",
                    "D Major - (2 sharps)",
                    "A Major - (3 sharps)",
                    "E Major - (4 sharps)",
                    "B Major - (5 sharps)",
                    "F# Major  (6 sharps)",
                    "C# major  (7 sharps)"};


    char * instSub[128] =
{
"   Acoustic Piano   ","    Bright Piano    ","Electric Grand Piano","  Honky-tonk Piano  ","  Electric Piano 1  ",
"  Electric Piano 2  ","    Harpsichord     ","        Clavi       ","       Celesta      ","    Glockenspiel    ",
"     Music Box      ","    Vibraphone      ","      Marimba       ","     Xylophone      ","    Tubular Bell    ",
"      Dulcimer      ","   Drawbar Organ    ","  Percussive Organ  ","     Rock Organ     ","    Church organ    ",
"     Reed organ     ","      Accordion     ","      Harmonica     ","   Tango Accordion  ","Acous Guitar (nylon)",
"Acous Guitar (steel)","Elec. Guitar (jazz) ","Elec. Guitar (clean)","Elec. Guitar (muted)","  Overdriven Guitar ",
"  Distortion Guitar ","  Guitar harmonics  ","   Acoustic Bass    ","Elec. Bass (finger) "," Elec. Bass (pick)  ",
"    Fretless Bass   ","     Slap Bass 1    ","     Slap Bass 2    ","    Synth Bass 1    ","    Synth Bass 2    ",
"       Violin       ","        Viola       ","        Cello       ","     Double bass    ","   Tremolo Strings  ",
"  Pizzicato Strings ","   Orchestral Harp  ","      Timpani       ","  String Ensemble 1 ","  String Ensemble 2 ",
"   Synth Strings 1  ","   Synth Strings 2  ","     Voice Aahs     ","     Voice Oohs     ","    Synth Voice     ",
"   Orchestra Hit    ","      Trumpet       ","      Trombone      ","       Tuba        ","   Muted Trumpet    ",
"    French horn     ","   Brass Section    ","   Synth Brass 1    ","   Synth Brass 2    ","    Soprano Sax     ",
"      Alto Sax      ","     Tenor Sax      ","    Baritone Sax    ","       Oboe         ","    English Horn    ",
"      Bassoon       ","      Clarinet      ","       Piccolo      ","       Flute        ","      Recorder      ",
"     Pan Flute      ","    Blown Bottle    ","      Shakuhachi    ","      Whistle       ","       Ocarina      ",
"  Lead 1 (square)   ","  Lead 2 (sawtooth) ","  Lead 3 (calliope) ","   Lead 4 (chiff)   ","  Lead 5 (charang)  ",
"   Lead 6 (voice)   ","   Lead 7 (fifths)  ","Lead 8 (bass + lead)","   Pad 1 (new age)  ","    Pad 2 (warm)    ",
" Pad 3 (polysynth)  ","   Pad 4 (choir)    ","   Pad 5 (bowed)    ","  Pad 6 (metallic)  ","    Pad 7 (halo)    ",
"   Pad 8 (sweep)    ","     FX 1 (rain)    "," FX 2 (soundtrack)  ","   FX 3 (crystal)   ","  FX 4 (atmosphere) ",
"  FX 5 (brightness) ","   FX 6 (goblins)   ","   FX 7 (echoes)    ","   FX 8 (sci-fi)    ","        Sitar       ",
"       Banjo        ","      Shamisen      ","       Koto         ","      Kalimba       ","      Bagpipe       ",
"       Fiddle       ","       Shanai       ","    Tinkle Bell     ","     Agogo Bells     ","    Steel Drums     ",
"     Woodblock      ","     Taiko Drum     ","    Melodic Tom     ","     Synth Drum     ","   Reverse Cymbal   ",
"  Guitar Fret Noise ","    Breath Noise    ","      Seashore      ","     Bird Tweet     ","   Telephone Ring   ",
"     Helicopter     ","      Applause      ","       Gunshot      "};


  char * timeSubtop[3] = {"    4 notes/meas    ","    3 notes/meas    ","    2 notes/meas    "};
  char * timeSub[2] = {"  4 - quarter notes "," 8 - eighth notes  "};
```

```
char ** currSub[5] = {clefSub, keySub,timeSubtop,timeSub,instSub};
int SubNumOpts[5] = {1,14,2,1,127};
temp = currSub[0];
numOpts = 1;
currMenu = 0;
ImageTemplate();
printLine2(specs[0]);
printLine3(temp[currMenu]);

while(1){ // Polls through the 4 user input options (right, left, select, back)

if (left == 1){// Scrolls left if there are options to scroll
  lctr++;
  left=0;
    if (lctr == 2){
      lctr = 0;
      left = 0;

      if (currMenu == 0){
        currMenu = numOpts;
        i = sprintf(str, "%d", currMenu);
        if (currSpec == 5){
          outchar(0x110602);
          print("   ");
          outchar(0x110602);
          print(str);
        }
        else {
          printLine3(temp[currMenu]);
        }
      }
      else{
        currMenu--;
        i = sprintf(str, "%d", currMenu);
        if (currSpec == 5){
          outchar(0x110602);
          print("   ");
          outchar(0x110602);
          print(str);
        }
        else {
          printLine3(temp[currMenu]);
        }
      }
    }
```

```
}

else if (right == 1){ //scrolls right if there are options to scroll
  rctr++;
  right=0;
    if (rctr == 2){
      rctr = 0;
      right = 0;

      if (currMenu == numOpts){
        currMenu = 0;
        i = sprintf(str, "%d", currMenu);
        if (currSpec == 5){
          outchar(0x110602);
          print("   ");
          outchar(0x110602);
          print(str);
        }
        else {
          printLine3(temp[currMenu]);
        }

      }
      else{
        currMenu++;
        //printLine3(temp[currMenu]);
        i = sprintf(str, "%d", currMenu);
        if (currSpec == 5){
          outchar(0x110602);
          print("   ");
          outchar(0x110602);
          print(str);
        }
        else {
          printLine3(temp[currMenu]);
        }

      }
    }
}

////////////////////////////////////////
//
//  Checks if there is a select button hit
//
```

```
//////////////////////////////////////////

else if (select == 1){
  select = 0;

  if (currSpec == 0){//Assigns values to header
    if (currMenu == 0){
      header[5] = -1;
    }
    else if (currMenu == 1){
      header[5]  = 1;
    }
  }
  else if (currSpec == 1){
    if (currMenu > 0 && currMenu < 8){
      header[1] = -1 * currMenu;
    }
    else if (currMenu >= 8){
      header[1] = currMenu - 7;
    }
    else {
      header[1] = 0;
    }
  }
  else if (currSpec == 2){
    if (currMenu == 0){
      header[2] = 4;
    }
    else if (currMenu == 1){
      header[2] = 3;
    }
    else if (currMenu == 2){
      header[2] = 2;
    }
  }
  else if (currSpec == 3){
    if (currMenu == 0){
      header[3] = 4;
    }
    else if (currMenu == 1){
      header[3] = 8;
    }
  }
  else if (currSpec == 4){
    header[4] = currMenu;
```

```
    }
    else if (currSpec == 5){
      header[0] = currMenu;
    }

    if (currSpec == 5){
        MainMenuTemplate();
        printMenu("creating IMS...select for new image");
       // for (i=0; i < 6;i++){
       //     printf("%d\n",header[i]);
       // }
        while (select == 0){
        }
        select = 0;
        return header;
    }
    currSpec++;
    printLine2(specs[currSpec]);

    if (currSpec == 5){
      select = 0;
      printLine3("          BPM       ");
      numOpts = 250;
      currMenu = 120;
      outchar(0x110602);
      print("120");
      for(i=0; i<65000; i++){
        i=i;
      }

    }
    else {
      select = 0;
      temp = currSub[currSpec];
      printLine3(currSub[currSpec][0]);
      numOpts = SubNumOpts[currSpec];
      currMenu = 0;
    }

  }

////////////////////////////////////////
//
//  Checks if there is a back button hit
//
```

```
//////////////////////////////////////////

  else if (back == 1) {
    back = 0;
         if (currSpec == 0){
           return NULL;
    }
         else {
      currSpec--;
      printLine2(specs[currSpec]);
      temp = currSub[currSpec];
      printLine3(currSub[currSpec][0]);
      numOpts = SubNumOpts[currSpec];
      currMenu = 0;
         }
  }



} // End of Menu Polling loop.


}

/*************************************
*
*  Interrupt Service Routine
*
*************************************/

#include "ISR.h"

void DAIroutine(int sig_int){
    static int interrupt_reg;

extern int left,right,select,back;
       int ctr = 0;

       interrupt_reg = *pDAI_IRPTL_H;

    //test for SRU_EXTMISCB1_INT (Falling Edge)
    if ((interrupt_reg & SRU_EXTMISCB1_INT) != 0){
      if ((*pDAI_PIN_STAT & DAI_PB03) == 0x0){
     //     printf("int on fall..right\n");
          right = 1;
```

```
          left = 0;

      }
      else {
        //  printf("int on fall..left\n");
          left = 1;
          right = 0;

      }

    }

     //test for SRU_EXTMISCB2_INT (Rising Edge)
    if ((interrupt_reg & SRU_EXTMISCB2_INT) != 0){
      if ((*pDAI_PIN_STAT & DAI_PB03) == 0x0){
      //    printf("int on rise..left\n");
          right = 0;
          left = 1;
      }
      else {
      //    printf("int on rise..right\n");
          left = 0;
          right = 1;
      }

    }

    if ((interrupt_reg & SRU_EXTMISCB3_INT) != 0){
        select = 1;
        back = 0;
    }

    if ((interrupt_reg & SRU_EXTMISCB4_INT) != 0){
        back = 1;
        select = 0;
    }


}


/*************************************************************
*
*
*
```

```
************************************************************/



/*******************************************************
*        Funtion:BUS_read
*        Purpose: To read one byte(8-bits) from the main bus to the DSP
*
*        Inputs:  NONE
*        Returns:  RxData
*        Notes: Level 1 function- Core Driven
*                  for step by step understanding refer to peripherals manual
*                  pg 3-21: Core Driven Transfers
*******************************************************/
BYTE BUS_read()
{
    BYTE RxData;

    *(volatile int *)PPCTL=0; /*Step 0:disables parallel port/Clear control regester */

    /*Step 1: setup PPDMA regesters for core use*/
    * (volatile int *)EMPP= 0;
    * (volatile int *)EIPP=EXTUSB+1;


         /*Step 2: Set up PPCTL registers */
         *(volatile int *)PPCTL = PPEN|       /* Enables Parallel port */
                                  PPBHC|     /* implement a bus hold cycle*/
                                  PPDUR20;    /*Data cycle lasts 20 core cycles */
                                  /*PP set for recieve*/
                                  /*DMA disabled*/


         RxData = *(volatile int *)RXPP & 0xFF;

         /* Disables Parallel Port*/
         * (volatile int *)PPCTL= PPBHC|
                                                          PPDUR20;

         //printf("BUS_read: %x\n", RxData);
         return RxData;
}
/*******************************************************
*        Funtion:BUS_writeCommand
```

```
*         Purpose: To write one byte command from the DSP to the USB
*
*         Inputs: TxData
*         Returns:  NONE
*         Notes: level 1 function- Core Driven
*                           Refer to pg 3-21 of peripherals manual
*                            on Core driven Transfers
*******************************************************/
void BUS_writeCommand(BYTE TxData)
{
          int i=0;

    *(volatile int *)PPCTL = 0;

          /* setup ppdma registers for core use */
          * (volatile int *)EMPP = 0;
          * (volatile int *)EIPP = EXTUSB;

          /* Set up PPCTL registers */
          *(volatile int *)PPCTL = PPEN| /* Enables Parallel port */
                                   PPTRAN|     /* transmit (write) */
                                   PPBHC|    /* implement a bus hold cycle*/
                           PPDUR20;  /* Data cycle lasts 20 core cycles */
                                                          /* disabled DMA */
                           //PPALEPL;  /* ALE is active low */

          *(volatile int *)TXPP = TxData; // Shifted into MSB or LSB?

          /* loop to wait for 1st data cycle to complete */
          i++;
          i++;
          i++;
          i++;
          i++;
          i++;

          /* Disables Parallel Port after 1st 8-bit transfer */
          * (volatile int *)PPCTL= PPTRAN|
                                                                  PPBHC|
                                                                  PPDUR20;

}

/*******************************************************
*         Funtion:BUS_write
```

```
*        Purpose: To write one byte of data from the DSP to the USB
*
*        Inputs: TxData
*        Returns:  NONE
*        Notes: level 1 function- Core Driven
*                          Refer to pg 3-21 of peripherals manual
*                           on Core driven Transfers
*******************************************************/
void BUS_writeData(BYTE TxData)
{
         int i=0;

    *(volatile int *)PPCTL = 0;

         /* setup ppdma registers for core use */
         * (volatile int *)EMPP = 0;
         * (volatile int *)EIPP = EXTUSB+1;

         /* Set up PPCTL registers */
         *(volatile int *)PPCTL = PPEN|  /* Enables Parallel port */
                                  PPTRAN|  /* transmit (write) */
                                  PPBHC|   /* implement a bus hold cycle*/
                                  PPDUR20; /* Data cycle lasts 20 core cycles */
                                                        /* disabled DMA */
                                  //PPALEPL;  /* ALE is active low */


         //printf("0x%x \n", TxData);
         *(volatile int *)TXPP = TxData; // Shifted into MSB or LSB?

         /* loop to wait for 1st data cycle to complete */
                  i++;
                  i++;
                  i++;
                  i++;
                  i++;
                  i++;

         /* Disables Parallel Port after 1st 8-bit transfer */
         * (volatile int *)PPCTL= PPTRAN|
                                                              PPBHC|
                                                              PPDUR20;

}
```

```
/*********************************************************
*        Funtion:EZUSB_delay
*        Purpose: Provides a ms Delay specified by input
*
*        Inputs: tt- ms delay wanted
*                        start- not used, legacy variable
*        Returns:  NONE
*        Notes: level 1 function
*
*
*********************************************************/

void EZUSB_Delay(WORD delay)
{
    extern int t_on;

    timer_set(2000000000, delay * 200000);
    timer_on();                         /* start timer */
    t_on=0;
    while(t_on==0);
    timer_off();  /*stop timer*/
}


void timer_isr(int sig)
{
    extern int t_on;
    t_on=1;
}

/*22222222222222222222222222222222222222222222222222222222222222
2
2                           LEVEL 2 FUNCTIONS
2
22222222222222222222222222222222222222222222222222222222222222*/
//*********************************************************************
// Byte Read from SL811H
// a = register address
// return = data in register
//*********************************************************************
BYTE SL811Read(BYTE a)
{
         BUS_writeCommand(a);
         return(BUS_read());
}
```

```
//*****************************************************************
// Byte Write to SL811H
// a = register address
// d = data to be written to this register address
//*****************************************************************
void SL811Write(BYTE a, BYTE d)
{
         BUS_writeCommand(a);        /*set USB address to write to*/
         BUS_writeData(d);  /*store data to USB address*/
}

//*********************************************************
// Buffer Read from SL811H
// addr = buffer start address
// s    = return buffer address where data are to be save/read
// c     = buffer data length
//*********************************************************
void SL811BufRead(BYTE addr, BYTE *s, BYTE c)
{
         if (BulkOnly)
         {
             int d = c;
             int n;
             while (d>0)
             {
                 *s = 0;
                           for (n=24; n>=0; n-=8)
                           {
                               if(d>0)
                               {
                                             BUS_writeCommand(addr++);
                                             *s |= (BUS_read()  << n) & (0xFF << n);
                                             d--;
                               }
                           }
                           *s++;
             }
         }
         else
         {
                  while (c--)
                  {
                      BUS_writeCommand(addr++);
                           *s++ = BUS_read();
```

```
                }
        }
}


//*****************************************************************************************
// USB Bulk-only Read/Write
//
//*****************************************************************************************
int BulkRW(BYTE epaddr, BYTE packing, int Sector, BYTE *pData)
{
    int tmp;
    sCBW CBWstruct = {0};
    sCSW CSWpacket = {0};
    BYTE bCBWFlags;
    BYTE bCBWLUN;
    BYTE CBWCBLength;
    BYTE bReadWrite;

    BulkOnly = 1;

    //-----------------------------------------------------
    // Send Command Block Wrapper
    //-----------------------------------------------------

    //MassStorageReset(1);

    sl811_usb_ready();
    sl811_usb_ready();
    sl811_usb_ready();

    CBWstruct.bCBWSignature = 0x55534243;
    CBWstruct.bCBWTag = 0x50494B41;
    CBWstruct.bCBWDataTransferLength = 0x00020000;
    bCBWFlags = 0x80;
    bCBWLUN = 0x00;
    CBWCBLength = 0x0A;
    bReadWrite = 0x28; // Read
    CBWstruct.bCBWCB[0] = ((Sector & 0xFF00) >>8) | ((Sector & 0xFF0000) >> 8) | ((Sector & 0xFF000000) >>8);
    CBWstruct.bCBWCB[1] = 0x00000001 | ((Sector & 0xFF) << 24);
    CBWstruct.bCBWCB[2] = 0;
    CBWstruct.bCBWCB[3] = 0;

    printf("%2x %2x %2x %2x\n", Sector & 0xFF00, (Sector & 0xFF0000) << 8, (Sector & 0xFF000000) << 16, Sector & 0xFF);
```

```
    if(epaddr == bEPOut) //Writing Data
    {
        print("WRT: ");
        CBWCBLength = 0x0C;
        bReadWrite = 0x2A;
        bCBWFlags = 0;
    }

    CBWstruct.CBWPacked = (bCBWFlags<<24) | (bCBWLUN<<16) | (CBWCBLength<<8) | bReadWrite;

    if(!usbXfer(1, bEPOut, PID_OUT, 0, 64, 0x1F, (BYTE*)&CBWstruct))
    {
        BulkOnly = 0;
        return FALSE;
    }

    //MassReset = 1;

    //-----------------------------------------------------
         // IN or OUT Bulk-only data stage
         //-----------------------------------------------------
                  if (uDev[1].bEPAddr[epaddr] & 0x80)                    // host-to-device : IN token
                  {
                      if(packing == FALSE)
                                    BulkOnly = 0;
                           if(!usbXfer(1, bEPIn, PID_IN, 0, uDev[1].wPayLoad[bEPIn], 512, pData))
                           {
                               printf(" FAILED1\n");
                               //BulkOnly = 0;
                                    //return FALSE;
                           }
                  }
                  else                                                   // device-to-host : OUT token
                  {
                      for(tmp=0; tmp<8; tmp++)
                      {
                                    if(!usbXfer(1, bEPOut, PID_OUT, 0, uDev[1].wPayLoad[bEPOut], 64, pData +
(tmp*64)))
                                    {
                                        printf(" FAILED2\n");
                               //BulkOnly = 0;
                                    //return FALSE;
                                    }
                      }
                  }
```

```
        //-----------------------------------------------------
        // Get Command Status Wrapper from USB device
        //-----------------------------------------------------
        BulkOnly = 1;

        while(!usbXfer(1, bEPIn, PID_IN, 0, uDev[1].wPayLoad[bEPIn], 13, (BYTE*)&CSWpacket))
        {
            printf("CSW error: retrying\n");
            EZUSB_Delay(5);
        }

        //printf("BulkRW - Sig: %8x  Tag: %x  DataResidue: %8x CSWStatus: %8x\n\n", CSWpacket.dCSWSignature,
CSWpacket.dCSWTag, CSWpacket.dCSWDataResidue, CSWpacket.bCSWStatus);

        //MassReset = 0;
    BulkOnly = 0;
        return TRUE;
}


/*****************************************************
*
*  FileFunctions.c -- functions to write to external memory
*                     using a buffered write
*
*****************************************************/

#include "FileFunctions.h"

/*************************************************************
*
*  BCO Type function.  The idea is to be able to write a single
*    byte to the function then have it wait until its full,
*    then write to the external memory.  Should replace fprintf
*
*
*************************************************************/

void
ExIMSWrite(char ch)
{
   extern int buff;
   extern int buffCtr;
```

```
   extern int IMSSz;
   extern unsigned int * pex_IMS;
    if (buffCtr == 3){
      buff = ch | buff;
      write_extmem(&buff, pex_IMS, 1);
      pex_IMS = pex_IMS + 4;
      buffCtr = 0;
      buff = 0;
    }
    else {
      buff = (ch << (24 -(8*buffCtr))) | buff ;
      buffCtr++;
    }
    IMSSz++;

}

void
ExMIDIWrite(char ch)
{
   extern int buff;
   extern int buffCtr;
   extern int MIDISz;
   extern unsigned int * pex_MIDI;
    if (buffCtr == 3){
      buff = ch | buff;
      write_extmem(&buff, pex_MIDI, 1);
      pex_MIDI = pex_MIDI + 4;
      buffCtr = 0;
      buff = 0;
    }
    else {
      buff = (ch << (24 -(8*buffCtr))) | buff ;
      buffCtr++;
    }
    MIDISz++;

}

void
IMSBuffFlush(void)
{
    //writes whatever remains in the buffer to the SRAM

   extern unsigned int * pex_IMS;
```

```
   extern int buff;
   extern int buffCtr;

   write_extmem(&buff, pex_IMS, 1);
   buff = 0;
   buffCtr = 0;
   pex_IMS = pex_IMS + 4;

}

void
MIDIBuffFlush(void)
{
    //writes whatever remains in the buffer to the SRAM

   extern unsigned int * pex_MIDI;
   extern int buff;
   extern int buffCtr;

   write_extmem(&buff, pex_MIDI, 1);
   buff = 0;
   buffCtr = 0;
   pex_MIDI = pex_MIDI + 4;

}


/*************************************************************
*
*  fatFxns.c -- Lower level functions to handle FAT system
*
*************************************************************/


#include "fatFxns.h"
#include "USB_Driver.h"

/*
#pragma varlocate "BIG_BUFF" FAT_CurrentSector, FAT_CurrentByte

extern unsigned long        FAT_CurrentSector;
extern unsigned long        FAT_CurrentByte;
*/

//char FatCurrentPath[FAT_MAX_PATH];
```

```
//#pragma udata FAT_C_UDATA
//#pragma code     FAT_C_CODE


/*       Initialisations diverses
 */

void cf_set_position(unsigned long lba, unsigned char sector_cnt)
{
    int temp[512] = {0};
    //int input[128] = { 0x11111111, 0x22222222, 0x33333333, 0x44444444, 0x55555555,
         //             0x66666666, 0x77777777, 0x88888888, 0x99999999, 0xaaaaaaaa,
         //              0xbbbbbbbb, 0xcccccccc, 0xdddddddd, 0xeeeeeeee, 0xffffffff};


    extern unsigned int * pex_USB;
    extern unsigned int USB[512];
    int test;
    int Sector;

    lba = lba & 0x0fffffff;
    lba = lba | 0xe0000000;

    Sector = lba;

//   BulkRW(bEPOut, PACKED, Sector, (BYTE *)input);
   BulkRW(bEPIn, NOTPACKED, Sector, (BYTE *)temp);


         printf("\n");
   pex_USB = USB;
   write_extmem(&temp,pex_USB,512);

  // read_extmem(&hold[0],pex_USB+20,1);
}

unsigned char cf_read8(void)
{
         unsigned int buf0;
         extern unsigned int * pex_USB;
    //extern unsigned int USB[512];
```

```
          read_extmem(&buf0, pex_USB,1);

    pex_USB += 4;
          buf0 = buf0 & 0x000000FF;
//        printf("buf0 from exread: %x\n",buf0);

          return buf0;
}

unsigned int cf_read16() {

          unsigned char buf0, buf1;
    extern unsigned int * pex_USB;

          read_extmem(&buf0, pex_USB,1);
          pex_USB += 4;
          read_extmem(&buf1, pex_USB,1);
          pex_USB += 4;
          //printf("buf0 from 16: %x\n",buf0);
//        printf("buf1 from 16: %x\n",buf1);
//        printf("buf1 from 16: %x\n",(((int)buf1 << 8) | buf0));
          buf0 = buf0 & 0x000000FF;
          buf1 = buf1 & 0x000000FF;

          return (((int)buf1 << 8) | buf0);
}

/*****************************************************
*
*   IMS_Build.c -- Sets up and builds the IMS file
*
*****************************************************/

#include "IMS_Build.h"
#include "FileFunctions.h"

/*************************************************************
 *
 * int MIDI_SharpFlatAdjust(int keySig, int MIDInum)
 *
 * Arguments: keySig: 1-7 = sharps, 0 = no sharps or flats
 *               -1..-7 = flats
 *
 * Return value: Sharp or Flat corrected MIDI number
 *
```

```
 * This function adds or subtracts one based on if it needs to be
 *   sharped or flatted.
 *
 ************************************************************/

int
MIDI_SharpFlatAdjust(int keySig, int MIDInum)
{
  int i = 0;
  int temp = 0;
  int check = 0;
  //printf("Adjusting for key signature... \n");
  //printf("MIDInum = %d, keySig = %d\n",MIDInum,keySig);

  if (keySig == 0){
    return MIDInum;
  }
  else{

         while(check == 0){
      if ((MIDInum % (CNOTE + (12 * i)) == 0) && (MIDInum / (CNOTE + (12 * i)) == 1)){
        temp = CNOTE;
                  //printf("Cnote..., i = %d\n",i);
                  check = 1;
      }
      else if((MIDInum % (BNOTE + (12 * i)) == 0) && (MIDInum / (BNOTE + (12 * i)) == 1)){
        temp = BNOTE;
                  //printf("Bnote...\n");
                  check = 1;
      }
      else if((MIDInum % (ANOTE + (12 * i)) == 0) && (MIDInum / (ANOTE + (12 * i)) == 1)){
        temp = ANOTE;
                  //printf("Anote...\n");
                  check = 1;
      }
      else if((MIDInum % (GNOTE + (12 * i)) == 0) && (MIDInum / (GNOTE + (12 * i)) == 1)){
        temp = GNOTE;
                  //printf("Gnote...\n");
                  check = 1;
          }
      else if((MIDInum % (FNOTE + (12 * i)) == 0) && (MIDInum / (FNOTE + (12 * i)) == 1)){
        temp = FNOTE;
        //printf("Fnote..i = %d\n",i);
                  check = 1;
      }
```

```
  else if((MIDInum % (ENOTE + (12 * i)) == 0) && (MIDInum / (ENOTE + (12 * i)) == 1)){
    temp = ENOTE;
               //printf("Enote...\n");
               check = 1;
  }
  else if((MIDInum % (DNOTE + (12 * i)) == 0) && (MIDInum / (DNOTE + (12 * i)) == 1)){
    temp = DNOTE;
               //printf("Dnote...\n");
               check = 1;
  }
       i++;
}

if (keySig == 1 && temp == FNOTE){
  return MIDInum + 1;
}
else if (keySig == 2 && (temp == FNOTE || temp == CNOTE)){
  return MIDInum + 1;
}
else if (keySig == 3 && (temp == FNOTE || temp == CNOTE || temp == GNOTE)){
  return MIDInum + 1;
}
else if (keySig == 4 && (temp == FNOTE || temp == CNOTE || temp == GNOTE ||
                              temp == DNOTE)){
  return MIDInum + 1;
}
else if (keySig == 5 && (temp == FNOTE || temp == CNOTE || temp == GNOTE ||
                              temp == DNOTE || temp == ANOTE)){
  return MIDInum + 1;
}
else if (keySig == 6 && (temp == FNOTE || temp == CNOTE || temp == GNOTE ||
                              temp == DNOTE || temp == ANOTE || temp == ENOTE)){
  return MIDInum + 1;
}
else if (keySig == 7 && (temp == FNOTE || temp == CNOTE || temp == GNOTE ||
            temp == DNOTE || temp == ANOTE || temp == ENOTE || temp == BNOTE)){
  return MIDInum + 1;
}
     /*Checking for flats here...*/
     else if (keySig == -1 && temp == BNOTE){
  return MIDInum - 1;
}
else if (keySig == -2 && (temp == BNOTE || temp == ENOTE)){
  return MIDInum - 1;
}
```

```
    else if (keySig == -3 && (temp == BNOTE || temp == ENOTE || temp == ANOTE)){
      return MIDInum - 1;
    }
    else if (keySig == -4 && (temp == BNOTE || temp == ENOTE || temp == ANOTE ||
                              temp == DNOTE)){
      return MIDInum - 1;
    }
    else if (keySig == -5 && (temp == BNOTE || temp == ENOTE || temp == ANOTE ||
                              temp == DNOTE || temp == GNOTE)){
      return MIDInum - 1;
    }
    else if (keySig == -6 && (temp == BNOTE || temp == ENOTE || temp == ANOTE ||
                              temp == DNOTE || temp == GNOTE || temp == CNOTE)){
      return MIDInum - 1;
    }
    else if (keySig == -7 && (temp == BNOTE || temp == ENOTE || temp == ANOTE ||
                temp == DNOTE || temp == GNOTE || temp == CNOTE || temp == FNOTE)){
      return MIDInum - 1;
    }

  }

  return MIDInum;
}

/*************************************************************
 *
 * int MIDI_NumAssign(int position, int keySig, int cleff)
 *
 * Arguments: position on staff from bottom staff line
 *            key signature: 1-7 = sharps, 0 = no sharps or flats
 *               -1..-7 = flats
 *            cleff: 1 = treble, -1 = bass
 *
 * Return value: MIDI number for note passed
 *
 * This function will find the corresponding MIDI number for
 *   for a note that is passed to it based on its cleff, key signature,
 *   and position on the staff.
 *
 *************************************************************/

int
MIDI_NumAssign(int position, int keySig, int clef)
{
```

```
        int temp = 0;
        int dir = 0;
        int i = 0;
        int j = 0;
        int counter = 0;

        //printf("Key Sig: %d\n",keySig);

        if (position >= 0){
                dir = UP;
        }
        else{
                dir = DOWN;
                position = position * -1;
  }

        if (clef == 1){
                temp = MID_E;
        }
        else{
                temp = LOW_E;
        }

        //printf("temp initial: %d\n",temp);

for (j = (position / 7); j >= 0 ; j--){
                if (j == 0){
                        counter = position % 7;
                }
                else{
                        counter = 7;
                }
        if (dir == UP){

    for (i = 0; i < counter; i++){
           if ((i == 0) || (i == 4)){
             temp++;
                  /*printf("temp 1: %d\n",temp);*/
           }
           else{
                  temp++;
                  temp++;
                 /* printf("temp 2: %d\n",temp);*/
           }
         }
```

```
         }
         else if (dir == DOWN){
      for (i = 0; i < counter; i++){
             if ((i == 2) || (i == 6)){
               temp--;
             }
             else{
                    temp--;
                    temp--;
                  }
           }
         }
  }

  /*call sharps and flats adjust...*/
  temp = MIDI_SharpFlatAdjust(keySig,temp);
  //printf("temp final: %d\n\n\n",temp);

  return temp;
}

/*********************************************************
 *
 * IMS_Build.c
 *
 * Authors: Ben McQuiston
 *
 * C function that writes the header created from the image parameter
 *  function that the user defines through the LCD.  Creates the IMS.
 *
 * Last Revision:
 *       - November 28, 2006
 *
 *********************************************************************/

void
IMS_Build(int *header)
{/*
  int i;
  FILE *fp1;
  fp1 = fopen("output.ims","w");
  if (fp1 == NULL){
            printf("Error in opening IMS output for writing");
  }
```

```
  for (i = 0; i < 5; i++){
  fprintf(fp1,"%c",(char)header[i]);
  }
  fclose(fp1);
  */
  int i = 0;

  for (i = 0; i < 6 ; i++){
      ExIMSWrite((char)header[i]);
  }

}
/*************************************************************
 *
 * IMS_Append.c
 *
 * Authors: Ben McQuiston
 *
 *  Called from the Note Analyzer, just writes two bytes corresponding
 *  to a given note from the image.  Appends the IMS each time.
 *
 * Last Revision:
 *       - November 28, 2006
 *
 *************************************************************/
void
IMS_Append(int *header, int position, int noteLen)
{
  int keySig,clef;
  int temp;
  //FILE *fp1;
  //fp1 = fopen("output.ims","a");
  //if (fp1 == NULL){
//         printf("Error in opening IMS output for writing");
  //}

  keySig = header[1];
  clef = header[5];

  temp = MIDI_NumAssign(position,keySig,clef);

  ExIMSWrite((char)(temp & 0x000000FF));
  ExIMSWrite((char)(noteLen & 0x000000FF));
  //fprintf(fp1,"%c",(char)(temp & 0x000000FF));
  //fprintf(fp1,"%c",(char)(noteLen & 0x000000FF));
```

```
  //fclose(fp1);

}

/*********************************************************
*
* IMS_MIDI.c -- Function that converts from IMS to MIDI
*
*********************************************************/
/*********************************************************
 *
 * IMS_MIDI.c
 *
 * Authors: Ben McQuiston
 *
 * C function to convert our house IMS files into a usable MIDI file
 *
 * Last Revision:
 *       - October 25, 2006
 *                          Major structure created for Mthd print and
 *          first MTrk with tempo specifiers
 *
 *       - October 29, 2006
 *                          Note processing code added, to decide how many bytes
 *                          a given track needs to be, and write the actual delta
 *                          times and note values to the MIDI file.
 *
 *********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "FileFunctions.h"

int
ImsToMIDI(void)
{
         //printf("Generating MIDI file...\n");

         extern unsigned int * pex_IMS;
         extern unsigned int * pex_MIDI;
         // extern unsigned int IMS;
         extern unsigned int IMS[100];
```

```
          extern unsigned int MIDI[1000];
          extern int IMSSz;
          extern int MIDISz;

          int checkBuff[50] ={0};
          char notes[200] = {0};
          int temp;
    int i = 0, j = 0;
          int tempo;
          int divisor = 60000000;
          int mspqn;

          int fileMrk;
          int numTrkByts = 0;
          int ctr = 0;

          //FILE *fp1;
          //FILE *fp2;

          char ch;
          char keySig;
          char timeSigTop;
          char timeSigBot;
          char instr;
          char noteVal;


//        fp1 = fopen(filename, "rb");
//        fp2 = fopen("output.mid","w");

          //if(fp1 == NULL){
//          printf("ERROR\n");
//          return -1;
//        }

  pex_MIDI = MIDI;

/*******************************************************
*
*  Mthd header to file...this will always be the same
*
*******************************************************/


  ExMIDIWrite(0x4d); //fprintf(fp2,"%c%c%c%c",0x4D,0x54,0x68,0x64);
```

```
  ExMIDIWrite(0x54);
  ExMIDIWrite(0x68);             /* Mthd */
  ExMIDIWrite(0x64);

  ExMIDIWrite(ZERO);        //fprintf(fp2,"%c",zero);
  ExMIDIWrite(ZERO);        //fprintf(fp2,"%c",zero);
  ExMIDIWrite(ZERO);        //fprintf(fp2,"%c",zero);
  ExMIDIWrite(0x06);        //fprintf(fp2,"%c",0x06);/* Always 6 bytes in Mthd header */
  ExMIDIWrite(ZERO);        //fprintf(fp2,"%c",zero);
  ExMIDIWrite(0x01);        //fprintf(fp2,"%c",0x01);/* MIDI Type 1 */
  ExMIDIWrite(ZERO);        //fprintf(fp2,"%c",zero);
  ExMIDIWrite(0x02);        //fprintf(fp2,"%c",0x02);/* Number of Tracks... could change?*/
  ExMIDIWrite(ZERO);        //fprintf(fp2,"%c",zero);
  ExMIDIWrite(0xF0);        //fprintf(fp2,"%c",0xF0); /*240 PPQN always..so dont forget*/

          /*fprintf(fp2,"%c",zero); delta time 00 */

/*******************************************************
*
*   Mtrk Header...
*
*******************************************************/

  ExMIDIWrite(0x4d); //fprintf(fp2,"%c%c%c%c",0x4D,0x54,0x72,0x6B);
  ExMIDIWrite(0x54);
  ExMIDIWrite(0x72);             /* Mtrk */
  ExMIDIWrite(0x6B);

  ExMIDIWrite(ZERO);        //fprintf(fp2,"%c",zero);
  ExMIDIWrite(ZERO);        //fprintf(fp2,"%c",zero);
  ExMIDIWrite(ZERO);        //fprintf(fp2,"%c",zero);
  ExMIDIWrite(0x19);        //fprintf(fp2,"%c",0x19); /* 25 bytes in first track...stays the same */

  ExMIDIWrite(ZERO);        //fprintf(fp2,"%c",zero); /* delta time 00 */

/*****************************************************
*
*  Start at beginning of IMS file to start getting
*  musical information...
*
*****************************************************/

  pex_IMS = IMS;   //fseek(fp1,0,SEEK_SET);
```

```
        /* tempo cammand FF 51 03 xx xx xx = micros / quarter note */

    read_extmem(&temp, pex_IMS, 1);  //fscanf(fp1,"%c",&ch);
        tempo = (temp & 0xFF000000) >> 24;//tempo = (int) ch;
        //tempo = tempo & ~(0xFFFFFFFF << 8);mask off all but the lower byte);

        mspqn = divisor / tempo;

  ExMIDIWrite(0xFF);        //fprintf(fp2,"%c%c%c", 0xFF,0x51,0x03); tempo command
  ExMIDIWrite(0x51);
  ExMIDIWrite(0x03);
/*
ch =(char) ((mspqn  & 0x00FF0000)>>16);
ExIMSWrite(ch);
ch = (mspqn  & 0x0000FF00)>>8;
ExIMSWrite(ch);
ch = (mspqn  & 0x000000FF);
ExIMSWrite(ch);
*/
  ExMIDIWrite((char) ((mspqn  & 0x00FF0000)>>16));//fprintf(fp2,"%c%c%c",*((char *) &mspqn +2),*((char *) &mspqn+1),
*((char *) &mspqn));
  ExMIDIWrite((char) ((mspqn  & 0x0000FF00)>>8));
  ExMIDIWrite((char) (mspqn  & 0x000000FF));

  ExMIDIWrite(ZERO);        //fprintf(fp2,"%c",zero);  /*delta time 00 */

        /* key sig command FF 59 02 xx xx = flats or sharps and major or minor */

        //fscanf(fp1,"%c",&ch);

        //keySig = (temp & 0x00FF0000) >> 16;   //keySig = ch;

        /* logic........for now we are just going to force it
        * to C major no flats we are not sure why we need this...*/

        ExMIDIWrite(0xFF); //fprintf(fp2,"%c%c%c",0xFF,0x59,0x02);
        ExMIDIWrite(0x59);
        ExMIDIWrite(0x02);

        ExMIDIWrite(ZERO); //fprintf(fp2,"%c",zero); /* no flats (C) */
        ExMIDIWrite(ZERO); //fprintf(fp2,"%c",zero); /* major */
        ExMIDIWrite(ZERO); //fprintf(fp2,"%c",zero); /* delta time 00 */

        /* Time Signature command FF 58 04 xx xx xx xx
        * numerator and denominator of time signature
```

```
        * number of MIDI clocks per beat (always 24)
        * number of 32nd notes per quater note (always 8)
        */

        //fscanf(fp1,"%c",&ch);
        timeSigTop = (temp & 0x0000FF00) >> 8; //timeSigTop = ch;

        ExMIDIWrite(0xFF);    //fprintf(fp2,"%c%c%c",0xFF,0x58,0x04);
        ExMIDIWrite(0x58);
        ExMIDIWrite(0x04);

        ExMIDIWrite(timeSigTop);    //fprintf(fp2,"%c",ch);

    timeSigBot = (temp & 0x000000FF);    //fscanf(fp1,"%c",&ch);
        ch = timeSigBot;

    /* denominator must be a negative power of four,
        *  so we check if its a 4 or an 8 and print the corresponding
        * power to the file...
        */

        if ((int) ch == 4){
                ExMIDIWrite(0x02); //fprintf(fp2,"%c",0x02);
        }
        else if ((int) ch == 8){
                ExMIDIWrite(0x03); //fprintf(fp2,"%c",0x03);
        }

        ExMIDIWrite(0x18); //fprintf(fp2,"%c%c",0x18,0x08);
        ExMIDIWrite(0x08); //24 MIDI clocks per beat, 8 32nds per quarter*/


        ExMIDIWrite(ZERO); //fprintf(fp2,"%c",zero);  /* delta time 00 */

        ExMIDIWrite(0xFF);  //fprintf(fp2,"%c%c%c",0xFF,0x2F,0); /* End of Track Marker */
    ExMIDIWrite(0x2F);
    ExMIDIWrite(ZERO);

    //MIDIBuffFlush();

/************************************************************
*
*  NOTE PROCESSING
*
*  Actually deal with the note values and lengths
```

```
*  Figure out how long it is going to be so that the Mtrk
*  header can tell how many bytes to expect, then just
*  process all of the notes two bytes at a time...
*
*********************************************************/


        ExMIDIWrite(0x4D);  //fprintf(fp2,"%c%c%c%c",0x4D,0x54,0x72,0x6B); /* Mtrk */
        ExMIDIWrite(0x54);
        ExMIDIWrite(0x72);
        ExMIDIWrite(0x6B);

        /* So to make this work we have to do an initial search through
        * the file to figure out what is in there...basically to figure out
        * out how many bytes are goiung to be in the file....*/

        //This is going to be alot easier, just check the IMSSz variable to figure out
        // how big the IMS file is, subtract 6 for the header, then divide by two for
        // the number of notes, and multiply by nine for the number of bytes in each
        //note on, note off sequence.

        read_extmem(&temp, (pex_IMS + 4), 1);//fscanf(fp1,"%c",&ch);
        ch = (temp & 0xFF000000) >> 24;
        instr = ch;

        fileMrk = 5;
/*
        while (fscanf(fp1,"%c",&ch) == 1){

                fscanf(fp1,"%c",&ch);
                if (ch == 0x03 || ch == 0x04 || ch == 0x05){
                        numTrkByts += 8;
                }
                else{
                        numTrkByts += 9;
                }
        }
*/
    numTrkByts = (((IMSSz - 6) / 2) * 9);

    //printf("count: %d\n", numTrkByts);

        //fseek(fp1,fileMrk,SEEK_SET);

        numTrkByts += 8; /* + 4 to account for the four bytes to tell
```

```
                                        * what instrument is to be played and +4 for
                                        the end of track bytes*/

//       printf("numTrkByts: %d\n",numTrkByts);
  ExMIDIWrite((char) ((numTrkByts  & 0x00FF0000)>>24));//fprintf(fp2,"%c",*((char *) &numTrkByts + 3));
  ExMIDIWrite((char) ((numTrkByts  & 0x0000FF00)>>16));//fprintf(fp2,"%c",*((char *) &numTrkByts + 2));
  ExMIDIWrite((char) ((numTrkByts  & 0x000000FF)>>8));//fprintf(fp2,"%c",*((char *) &numTrkByts + 1));
  ExMIDIWrite((char) (numTrkByts  & 0x000000FF));       //fprintf(fp2,"%c",*((char *) &numTrkByts));


                 /* actual number of bytes in the coming track */

         /*After we figure out how many bytes there will be in the track,
         * we have to actually process the notes and things...*/

         ExMIDIWrite(ZERO);  //fprintf(fp2,"%c",zero);
         ExMIDIWrite(0xC0);  //fprintf(fp2,"%c%c",0xC0,instr); /*tells what instrument it is ..*/
         ExMIDIWrite(instr);
         ExMIDIWrite(ZERO); //fprintf(fp2,"%c",zero);

         notes[0] = (temp & 0x0000FF00) >> 8;
         notes[1] = (temp & 0x000000FF);
         ctr = 2;

         for (i = 0; i <= (IMSSz - 8) / 4; i++){
             //printf("i: %d\n",i);
             read_extmem(&temp, (pex_IMS + 8 + (4*i)), 1);
            // printf("temp: %x\n",temp);
             for (j = 0; j < 4; j++){
                 notes[j + ctr] = ((temp >> (24 - (8 * j))) & 0x000000FF);
             }
             ctr = ctr + 4;
         }

//       for (i = 0; i < (IMSSz - 6); i++){
//           printf("notes[%d] = %x\n",i,notes[i]);
//       }

         for (i = 0; i < (IMSSz - 6); i++){
                  if (i % 2 == 0){
               noteVal = notes[i];   //while (fscanf(fp1,"%c",&ch) == 1){
                     //noteVal = ch;
                     ExMIDIWrite(0x90);                 //fprintf(fp2,"%c%c%c",0x90,noteVal,0x40);
                     ExMIDIWrite(noteVal);
                     ExMIDIWrite(0x40);
```

```
}
else {
       ch = notes[i];    //fscanf(fp1,"%c",&ch);
       //printf("ch: %X\n",ch);
       if (ch == 0){ /* whole note = 960 pulses */
              ExMIDIWrite(0x87);    //fprintf(fp2,"%c",0x87);
              ExMIDIWrite(0x40);    //fprintf(fp2,"%c",0x40);
       }
       else if (ch == 0x01){ /* half note = 480 pulses */
              ExMIDIWrite(0x83);   // fprintf(fp2,"%c",0x83);
              ExMIDIWrite(0x60);   // fprintf(fp2,"%c",0x60);
       }
       else if (ch == 0x02){ /* quarter note = 240 pulses */
              ExMIDIWrite(0x81);   // fprintf(fp2,"%c",0x81);
              ExMIDIWrite(0x70);   // fprintf(fp2,"%c",0x70);
       }
       else if (ch == 0x03){ /* eigth note = 120 pulses */
              ExMIDIWrite(0x78);   // fprintf(fp2,"%c",0x78);
       }
       else if (ch == 0x04){ /* 16th note = 60 pulses */
              ExMIDIWrite(0x3C);   // fprintf(fp2,"%c",0x3C);
       }
       else if (ch == 0x05){/* 32nd note = 30 pulses */
              ExMIDIWrite(0x1E);   // fprintf(fp2,"%c",0x1E);
       }
       else if (ch == 0x06){/* dotted eigth note = 180 pulses */
              ExMIDIWrite(0x81);  //  fprintf(fp2,"%c",0x81);
              ExMIDIWrite(0x34);  //  fprintf(fp2,"%c",0x34);
       }
       else if (ch == 0x07){ /* dotted quarter note = 360 pulses */
              ExMIDIWrite(0x82);   // fprintf(fp2,"%c",0x82);
              ExMIDIWrite(0x68);    //fprintf(fp2,"%c",0x68);
       }
       else if (ch == 0x08){ /* dotted half note = 720 pulses */
              ExMIDIWrite(0x83);    //fprintf(fp2,"%c",0x83);
         ExMIDIWrite(0x50);    //         fprintf(fp2,"%c",0x50);
       }

       ExMIDIWrite(0x80);    //fprintf(fp2,"%c%c%c",0x80,noteVal,0x40);
       ExMIDIWrite(noteVal);
       ExMIDIWrite(0x40);

  if (i == (IMSSz - 7)){
    //printf("hello\n");
    ExMIDIWrite(0xFF);
```

```
                ExMIDIWrite(0x7F);
                ExMIDIWrite(0xFF);
                ExMIDIWrite(0x2F);
                ExMIDIWrite(ZERO);
                   }
                   else {
                     ExMIDIWrite(ZERO);
                   }

                 }

         }
         //fseek(fp2,-1,SEEK_CUR);

//       fprintf(fp2,"%c",0xFF);
//       fprintf(fp2,"%c",0x7F);
//       fprintf(fp2,"%c%c%c",0xFF,0x2F,0); /* End of Track Marker */

////////////////////////////////
//
//  Check whats in buffer...
//
////////////////////////////
/*
  MIDIBuffFlush();

      read_extmem(checkBuff,MIDI,32);

    for (i = 0; i < 32; i++){
        printf("%8X\n",checkBuff[i]);
    }

*/
//////////////////////////////////////

    //printf("MIDIsize: %d\n",MIDISz);
         return 0;

}



//////////////////////////////////////////
//
//image_processing.c – This module interprets
```

```
//    the sheet music and calls the IMS_Build
//    function.
//
////////////////////////////////////////

void process_image(void)
{
  int bitmap_header[3],staves;
  unsigned int *pex_sheet_hist,*pex_staff_line_loc;
  //extern unsigned int *pex_image;//,ex_image[],ex_sheet_hist[],ex_staff_line_loc[];

/*  if(read_header(bitmap_header))
  {
      printf("ERROR:  File is uncompatable");
  }*/
  pex_image=ex_image;
  pex_sheet_hist=ex_sheet_hist;
  pex_staff_line_loc=ex_staff_line_loc;
  histogram(pex_sheet_hist,bitmap_header[1],0,bitmap_header[2],0,bitmap_header[1],1);
  staves=staff_line_locator(pex_staff_line_loc,pex_sheet_hist,bitmap_header[2],bitmap_header[1]);
  note_finder(pex_sheet_hist,pex_staff_line_loc,staves,bitmap_header[1],bitmap_header[0]);
}

/*************************************************************************
*
*  This function stores all of the bitmap header values to variables and then
*    stores the raw data of the picture.  For a complete list of the data
*    elements of the bitmap file header go to:
*    http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html
*
*************************************************************************/

void read_header(unsigned int *pex_header,int *important_info)
{
  unsigned int temp;

//  read_extmem(&bfType,pimage,1);
//  bfType=0xFFFF&bfType;
//  printf("%d\n",bfType);
//  fseek(fp,2,SEEK_SET);
  read_extmem(&temp,pex_header+8,1);
```

```
  important_info[0]=temp;
  read_extmem(&temp,pex_header+16,1);
  important_info[1]=temp;
  read_extmem(&temp,pex_header+20,1);
  important_info[2]=temp;

//  fread(bitmap_header,4,13,fp);
//  bitmap_header_6a=0xFFFF&bitmap_header[6];
//  for(i=0;i!=6;i++)
//  {
//    printf("%d\n",bitmap_header[i]);//This is data elements 2-8 where elements 3 and 4 are combined
together since they both must be 0 they will equal 0 together.
//  }
//  printf("%d\n",bitmap_header_6a);
//  bitmap_header[6]=(0xFFFF0000&bitmap_header[6])>>16;//This is the 9th data element
//  for(i=6;i!=13;i++)
//  {
//    printf("%d\n",bitmap_header[i]);This is data elements 10-16
//  }
//  if (bfType!=19778)
//  {
//    printf("\nERROR: File %s is not a bitmap file\n",image_name);
//    return(1);
//  }
//  if (bitmap_header[6]!=1)
//  {
//    printf("\nERROR: File %s is not in binary\n",image_name);
//    return(1);
//  }
//  if (bitmap_header[7])
//  {
//    printf("\nERROR: File %s is compressed\n",image_name);
//    return(1);
//  }
//  fclose(fp);
//  important_info[0]=bitmap_header[2];
//  important_info[1]=bitmap_header[4];
//  important_info[2]=bitmap_header[5];
//  return(0);
}
```

```
/*************************************************************************
*
*  This function creates an histogram from the image name given.  It
*    requires the rows and columns lengths as arguements as well as the
*    starting and ending row and column that the histogram will be made of.
*    The offset from where the data starts in the bitmap file is needed as
*    an argument as well.  It also needs to know if the histogram is
*    suppose to be of the rows or the columns.  If it is suppose to be
*    of the rows, the roworcol variable should have a value, but if it
*    is suppose to be of the columns the variable will be zero.  The
*    function grabs 4 bytes of data and stores it in an int.  It then
*    masks every bit so that if the bit is set, then the int will
*    have a value, but if it is not, then the int will be zero.  This is
*    how the function finds how many black pixels are in the int it pulls
*    from the file.
*
*************************************************************************/

void histogram(unsigned int *phist,int columns,int rows_top,int rows_bot,int columns_top,int
columns_bot,int roworcol)
{
  extern unsigned int *pex_image;
  int i,j,k,m,mask,expbase2,col_adj,bytesofhist,bytesofcol,col_offset;
  int count=0,bit_offset;
  unsigned int temp_hist,temp;
  //extern unsigned int ex_image[];

  bytesofhist=(columns_bot-columns_top)/32;
  mask=(columns_bot-columns_top)%32;
  bytesofcol=columns/32;
  bytesofcol*=4;
  col_adj=columns%32;
  col_offset=columns_top/32;
  col_offset*=4;
  if(col_adj)
  {
    col_adj=4;
  }
  bit_offset=columns_top-col_offset*8;
  for(i=rows_top;i<=rows_bot;i++)
  {
```

```
    pex_image=ex_image+i*(bytesofcol+col_adj)+col_offset;
    for(j=0;j<bytesofhist;j++)
    {
      read_extmem(&temp,pex_image,1);
      pex_image+=4;
      for(k=1;k<5;k++)
      {
        for(m=1;m<9;m++)
        {
          if((k==1)&&(m==1))
          {
            expbase2=1;
          }
          else
          {
            expbase2*=2;
          }
          if((temp&expbase2)==0)
          {
            if(roworcol)
            {
              read_extmem(&temp_hist,phist+count,1);
              temp_hist++;
              write_extmem(&temp_hist,phist+count,1);
            }
            else
            {
              read_extmem(&temp_hist,phist+4*(32*j+k*8-m),1);
              temp_hist++;
              write_extmem(&temp_hist,phist+4*(32*j+k*8-m),1);
            }
          }
        }
      }
    }
/*This last part deals with the last int of data in each row*/
    if(mask)
    {
      read_extmem(&temp,pex_image,1);
      pex_image+=4;
      for(k=1;k<5;k++)
```

```
      {
        for(m=1;m<9;m++)
        {
          if((k==1)&&(m==1))
          {
            expbase2=1;
          }
          else
          {
            expbase2*=2;
          }
          if((temp&expbase2)==0)
          {
            if(roworcol)
            {
              if((k*8-m)<mask)
              {
                read_extmem(&temp_hist,phist+count,1);
                temp_hist++;
                write_extmem(&temp_hist,phist+count,1);
              }
            }
            else if(((k*8-m)>=bit_offset)&&((k*8-m)<(bit_offset+columns_bot-columns_top)))
            {
              read_extmem(&temp_hist,phist+4*(32*j+k*8-m),1);
              temp_hist++;
              write_extmem(&temp_hist,phist+4*(32*j+k*8-m),1);
            }
          }
        }
      }
      count+=4;
    }
}
int staff_line_locator(unsigned int *padjusted_loc,unsigned int *pex_sheet_hist,int row,int column)
{
  int last_staff=0,staves=0,half_column,i,j,k,tolerance_of_space=4;
  int staves_dis_1,staves_dis_2,count=0,check_staff=1;
  unsigned int temp_dis[2],temp_sheet;
  unsigned int *pex_backwards_staff_line_loc;
```

```
//extern unsigned int ex_backwards_staff_line_loc[];

pex_backwards_staff_line_loc=ex_backwards_staff_line_loc;
half_column=column/2;
for(i=0;i<row*4;i+=4)
{
  read_extmem(&temp_sheet,pex_sheet_hist+i,1);
  if(temp_sheet>half_column)
  {
    if(last_staff+4!=i)
    {
      write_extmem(&i,pex_backwards_staff_line_loc+staves,1);
      staves+=4;
    }
    last_staff=i;
  }
}
for(i=0,j=0;i<staves-12;i+=4)
{
  read_extmem(temp_dis,pex_backwards_staff_line_loc+i-j,2);
  staves_dis_1=temp_dis[1]-temp_dis[0];
  read_extmem(temp_dis,pex_backwards_staff_line_loc+i-j+4,2);
  staves_dis_2=temp_dis[1]-temp_dis[0];
  if(abs(staves_dis_1-staves_dis_2)<=tolerance_of_space)
  {
    count++;
    check_staff=0;
  }
  else if((!count)&&(check_staff))
  {
    staves-=4;
    for(k=i-j;k<staves;k+=4);
    {
      read_extmem(&temp_sheet,pex_backwards_staff_line_loc+k+4,1);
      write_extmem(&temp_sheet,pex_backwards_staff_line_loc+k,1);
    }
    j+=4;
  }
  else if(count==3)
  {
    count=0;
```

```
    }
    else if(count==0)
    {
      check_staff=1;
    }
  }
  if(abs(staves_dis_1-staves_dis_2)>tolerance_of_space)
  {
    staves-=4;
    temp_sheet=0;
    write_extmem(&temp_sheet,pex_backwards_staff_line_loc+staves,1);
  }
  for(i=0;i<(staves/5);i+=4)
  {
    for(j=0;j<20;j+=4)
    {
      read_extmem(&temp_sheet,pex_backwards_staff_line_loc+((staves/5)-4-i)*5+j,1);
      write_extmem(&temp_sheet,padjusted_loc+i*5+j,1);
    }
  }
  return(staves/4);
}
////////////////////////////////////////////////////////////////////
//
//      This function makes a histogram for each of the staves columns and identifies the notes.
//
////////////////////////////////////////////////////////////////////
void note_finder(unsigned int *pex_sheet_hist,unsigned int *pex_staff_line_loc,int staves,int columns,int
offset)
{
  //extern unsigned int ex_staff_hist[],ex_note_hist[];
  //extern float ex_filt_staff_hist[];
  int i=0,j=0,around_staff,half_columns,staff_line_thickness=1;
  int space_between_lines=0,tolerance_of_note,start_staff,end_staff;
  int c,tolerance_of_intensity,note_thickness=0;
  int note_start=0,measure_line_thickness,count;
  int check_whole_note=0;
  const float gauss_filt[3]={.2741,.4519,.2741};
  unsigned int temp_input[CONVOLVE_TEMP_SIZE];
  float temp_output[CONVOLVE_TEMP_SIZE+2],*pex_filt_staff_hist;
  int note=0,next;
```

```
unsigned int temp,temp2,temp3,temp4;
unsigned int *pex_staff_hist,*pex_note_hist,*pex_measure_test;

pex_staff_hist=ex_staff_hist;
pex_filt_staff_hist=ex_filt_staff_hist;
pex_measure_test=ex_measure_test;
pex_note_hist=ex_note_hist;
if(staves>5)
{
  read_extmem(&around_staff,pex_staff_line_loc,1);
  read_extmem(&temp,pex_staff_line_loc+36,1);
  around_staff-=(int)temp;
  around_staff*=.5;
}
else
{
  read_extmem(&around_staff,pex_staff_line_loc+16,1);
  read_extmem(&temp,pex_staff_line_loc,1);
  around_staff-=(int)temp;
}
half_columns=columns/2;
while(!i)
{
  read_extmem(&temp,pex_staff_line_loc+4,1);
  read_extmem(&temp2,pex_sheet_hist+temp+staff_line_thickness,1);
  if(temp2>half_columns)
  {
    staff_line_thickness++;
  }
  else
  {
    i=1;
  }
}
for(i=0;i<4*staves;i+=20)
{
  for(j=0;j<16;j+=4)
  {
    read_extmem(&temp,pex_staff_line_loc+i+j+4,1);
    read_extmem(&temp2,pex_staff_line_loc+i+j,1);
    space_between_lines=temp-temp2-staff_line_thickness+space_between_lines;
```

```
  }
}
if((space_between_lines%(staves-i/20))>((space_between_lines/(staves-i/20))/2))
{
  space_between_lines=space_between_lines/(staves-i/20)+1;
}
else
{
  space_between_lines=space_between_lines/(staves-i/20);
}
measure_line_thickness=staff_line_thickness+1;
tolerance_of_note=space_between_lines*3;
for(i=0;i<4*staves;i+=20)
{
  read_extmem(&temp,pex_staff_line_loc+i,1);
  start_staff=temp-around_staff;
  read_extmem(&temp,pex_staff_line_loc+i+16,1);
  end_staff=temp+around_staff;
  temp=0;
  for(j=0;j<columns*4;j+=4)
  {
    write_extmem(&temp,pex_staff_hist+j,1);
  }
  histogram(pex_staff_hist,columns,start_staff,end_staff,0,columns,0);
  for(c=0;c<columns*4;c+=CONVOLVE_TEMP_SIZE*4)
  {
    count=0;
    for(j=0;j<CONVOLVE_TEMP_SIZE*4;j+=4)
    {
      if(c+j<columns)
      {
        read_extmem(&temp_input[j],pex_staff_hist+c+j,1);
      }
      else
      {
        count++;
      }
    }
    convolve((const float *)temp_input,CONVOLVE_TEMP_SIZE-count,gauss_filt,3,temp_output);
    for(j=1;j<=(CONVOLVE_TEMP_SIZE-count)*4;j+=4)
    {
```

```
        write_extmem(&temp_output[j],pex_filt_staff_hist+c+j,1);
      }
    }
    for(j=0,tolerance_of_intensity=0;j<columns*4;j+=4)
    {
      read_extmem(&temp,pex_staff_hist+j,1);
      tolerance_of_intensity=tolerance_of_intensity+temp;
    }
    tolerance_of_intensity=tolerance_of_intensity/columns+1;
    for(c=0;c<columns*4;c+=4)
    {
      read_extmem(&temp,pex_staff_line_loc+i+12,1);
      read_extmem(&temp2,pex_staff_line_loc+i,1);
      temp-=temp2;
      read_extmem(&temp2,pex_staff_hist+c,1);
      if(temp2>temp)
      {
        next=1;
        count=0;
        while((next)&&(count!=16))
        {
          for(j=0;j<128;j+=4)
          {
            temp=0;
            write_extmem(&temp,pex_measure_test+j,1);
          }

histogram(pex_measure_test,columns,*(pex_staff_line_loc+i+count),*(pex_staff_line_loc+i+count+4),c,c+1,0);
          for(j=0;j<128;j+=4)
          {
            read_extmem(&temp,pex_measure_test+j,1);
            if(temp)
            {
              read_extmem(&temp3,pex_staff_line_loc+i+count+4,1);
              read_extmem(&temp4,pex_staff_line_loc+i+count,1);
              if((temp3-temp4-temp)>measure_line_thickness)
              {
                next=0;
              }
            }
          }
```

```
          count+=4;
        }
        if(next)
        {
          check_whole_note=0;
          note_thickness=0;
        }
        else
        {
          note_thickness++;
        }
      }
      else if(temp2>tolerance_of_intensity)
      {
        note_thickness++;
      }
      else if(note_thickness>measure_line_thickness)
      {
        if ((note_thickness>space_between_lines)&&(!check_whole_note))
        {
          note++;
//           printf("%d ",c);
          temp=0;
          for(j=0;j<STAFF_MAX_HEIGHT*4;j+=4)
          {
            write_extmem(&temp,pex_note_hist+j,1);
          }
          histogram(pex_note_hist,columns,start_staff,end_staff,c/4-note_thickness-
measure_line_thickness*2,c/4+measure_line_thickness*2,1);

//note_analyzer(note_hist,space_between_lines,start_staff,end_staff,staff_line_loc,i,note_thickness+measure
_line_thickness*4,c+measure_line_thickness*2,measure_line_thickness,staff_line_thickness);
        }
        else if((note_thickness<=space_between_lines)&&(!check_whole_note))
        {
          check_whole_note=c/4;
          note_start=c/4-note_thickness;
        }
        else if(check_whole_note)
        {
          if((c/4-check_whole_note)<tolerance_of_note)
```

```
          {
            note++;
          //  printf("%d ",c);
            note_thickness=c/4-note_start;
            temp=0;
            for(j=0;j<STAFF_MAX_HEIGHT*4;j+=4)
            {
              write_extmem(&temp,pex_note_hist+j,1);
            }
            histogram(pex_note_hist,columns,start_staff,end_staff,c/4-note_thickness-
measure_line_thickness*2,c/4+measure_line_thickness*2,1);

//note_analyzer(note_hist,space_between_lines,start_staff,end_staff,staff_line_loc,i,note_thickness+measure
_line_thickness*4,c+measure_line_thickness*2,measure_line_thickness,staff_line_thickness);
            check_whole_note=0;
          }
          else
          {
            check_whole_note=c/4;
          }
        }
        note_thickness=0;
      }
      else
      {
        note_thickness=0;
      }
    }
    printf("\n");
  }
}
////////////////////////////////////////////////////////////////////
//
//      This function is called by note_finder, and it determines if the note sent to it
//      is in fact a note.  If it is a note it determines what length it is and what value
//      the note is.
//
////////////////////////////////////////////////////////////////////
void note_analyzer(unsigned int *pex_note_hist,int space_between_lines,int start_staff,int
end_staff,unsigned int *pex_staff_loc,int staff_num,int note_thickness,int c,int measure_line_thickness,int
staff_line_thickness)
```

```
{
  //extern unsigned int copy_of_hist[STAFF_MAX_HEIGHT];
  int height,i,j,k,baddown,badup,scan,up,down,max,index,corrected_index;
  int distance_between_lines,not_a_note;
  int up1,down1,index1,note_loc,base_staff_loc,note_top,note_bottom;
  int no_stem_up,no_stem_down,stem,hollow,prev_up,prev_down,filled;
  int current_up,current_down;
  unsigned int temp,*pex_copy_of_hist;

//  printf("%d %d\n",c-note_thickness,c);
  pex_copy_of_hist=ex_copy_of_hist;
  height=end_staff-start_staff;
  for(i=0;i<height*4;i+=4)
  {
    read_extmem(&temp,pex_note_hist+i,1);
    write_extmem(&temp,pex_copy_of_hist+i,1);
  }
  for(i=0;i<height*4;i+=4)
  {
    read_extmem(&temp,pex_note_hist+i,1);
    if(note_thickness-temp<measure_line_thickness*4)
    {
      temp=space_between_lines/2+1;
      write_extmem(&temp,pex_note_hist+i,1);
      read_extmem(&temp,pex_note_hist+i-4,1);
      if(i>0&&temp>space_between_lines)
      {
        temp=space_between_lines/2+1;
        write_extmem(&temp,pex_note_hist+i-4,1);
      }
      read_extmem(&temp,pex_note_hist+i+4,1);
      if(note_thickness-temp>=measure_line_thickness*4)
      {
        if(i<height-1&&temp>space_between_lines)
        {
          temp=space_between_lines/2+1;
          write_extmem(&temp,pex_note_hist+i+4,1);
        }
      }
    }
  }
```

```
distance_between_lines=space_between_lines+staff_line_thickness;
i=0;
while(!i)
{
  baddown=0;
  badup=0;
  scan=0;
  up=0;
  down=0;
  for(k=0,max=0,index=0;k<height*4;k+=4)
  {
    read_extmem(&temp,pex_note_hist+k,1);
    if(max<temp)
    {
      max=temp;
      index=k;
    }
  }
  j=0;
  if(max<=space_between_lines/2+1)
  {
    j=1;
    i=1;
    printf("not a note ");
  }
  while(!j)
  {
    scan+=4;
    read_extmem(&temp,pex_note_hist+index+scan,1);
    if(temp>space_between_lines/2)
    {
      down++;
    }
    else
    {
      baddown++;
    }
    read_extmem(&temp,pex_note_hist+index-scan,1);
    if(temp>space_between_lines/2)
    {
      up++;
```

```
    }
    else
    {
      badup++;
    }
    if(up+down==space_between_lines)
    {
      j=1;
      i=1;
      printf("%d ",index);
    }
    if(baddown==space_between_lines)
    {
      j=1;
      temp=space_between_lines/2+1;
      write_extmem(&temp,pex_note_hist+index,1);
    }
    else if(badup==space_between_lines)
    {
      j=1;
      temp=space_between_lines/2+1;
      write_extmem(&temp,pex_note_hist+index,1);
    }
  }
}
if(up<measure_line_thickness)
{
  corrected_index=4*(baddown+down/2+down%2)+index;
}
else if(down<measure_line_thickness)
{
  corrected_index=4*(-badup-up/2-up%2)+index;
}
else
{
  corrected_index=(down-up)*2+index;
}
not_a_note=0;
temp=0;
write_extmem(&temp,pex_note_hist+corrected_index,1);
for(i=1;i<=4*(distance_between_lines/2+distance_between_lines%2);i+=4)
```

```
{
  write_extmem(&temp,pex_note_hist+corrected_index+i,1);
  write_extmem(&temp,pex_note_hist+corrected_index-i,1);
}
i=0;
while(!i)
{
  baddown=0;
  badup=0;
  scan=0;
  up1=0;
  down1=0;
  for(k=0,max=0,index=0;k<height*4;k+=4)
  {
    read_extmem(&temp,pex_note_hist+k,1);
    if(max<temp)
    {
      max=temp;
      index1=k;
    }
  }
  j=0;
  if(max<=space_between_lines/2+1)
  {
    j=1;
    i=1;
  }
  while(!j)
  {
    scan++;
    read_extmem(&temp,pex_note_hist+index1+scan,1);
    if(temp>space_between_lines/2)
    {
      down1++;
    }
    else
    {
      baddown++;
    }
    read_extmem(&temp,pex_note_hist+index1-scan,1);
    if(temp>space_between_lines/2)
```

```
    {
      up1++;
    }
    else
    {
      badup++;
    }
    if(up1+down1==space_between_lines)
    {
      j=1;
      i=1;
      not_a_note=1;
      printf("too big for a note ");
    }
    if(baddown==space_between_lines)
    {
      j=1;
      temp=space_between_lines/2+1;
      write_extmem(&temp,pex_note_hist+index1,1);
    }
    else if(badup==space_between_lines)
    {
      j=1;
      temp=space_between_lines/2+1;
      write_extmem(&temp,pex_note_hist+index1,1);
    }
  }
}
if(!not_a_note)
{
  i=0;
  note_loc=0;
  read_extmem(&temp,pex_staff_loc+20*staff_num,1);
  base_staff_loc=temp-start_staff+staff_line_thickness/2-1;
  while(i==0)
  {
    if(abs(base_staff_loc-corrected_index)>distance_between_lines)
    {
      if(base_staff_loc>corrected_index)
      {
        note_loc-=2;
```

```
    base_staff_loc-=distance_between_lines;
  }
  else
  {
    note_loc+=2;
    base_staff_loc+=distance_between_lines;
  }
}
else if(abs(base_staff_loc-corrected_index)<=distance_between_lines/3-2)
{
  i=1;
}
else if(abs(base_staff_loc-corrected_index)<=distance_between_lines/3+1)
{
  if(base_staff_loc>corrected_index)
  {
    if(down>up)
    {
      i=1;
    }
    else if(up==down)
    {
      if(abs(base_staff_loc-corrected_index)<=distance_between_lines/3)
      {
        i=1;
      }
    }
    else
    {
      note_loc--;
      base_staff_loc-=distance_between_lines/2;
    }
  }
  else
  {
    if(up>down)
    {
      i=1;
    }
    else if(up==down)
    {
```

```
            if(abs(base_staff_loc-corrected_index)<=distance_between_lines/3)
            {
              i=1;
            }
          }
          else
          {
            note_loc++;
            base_staff_loc+=distance_between_lines/2;
          }
        }
        i=1;
      }
      else if(abs(base_staff_loc-
corrected_index)<=((2*distance_between_lines)%3!=0)+2*distance_between_lines/3-1)
      {
        if(base_staff_loc>corrected_index)
        {
          note_loc--;
          base_staff_loc-=distance_between_lines/2;
        }
        else
        {
          note_loc++;
          base_staff_loc+=distance_between_lines/2;
        }
        i=1;
      }
      else if(abs(base_staff_loc-
corrected_index)<=((2*distance_between_lines)%3!=0)+2*distance_between_lines/3+2)
      {
        if(base_staff_loc>corrected_index)
        {
          if(down>up)
          {
            note_loc--;
            base_staff_loc-=distance_between_lines/2;
          }
          else if(down==up)
          {
```

```
            if(abs(base_staff_loc-
corrected_index)<=((2*distance_between_lines)%3!=0)+2*distance_between_lines/3)
            {
              note_loc--;
              base_staff_loc-=distance_between_lines/2;
            }
          }
          else
          {
            note_loc-=2;
            base_staff_loc-=distance_between_lines;
          }
        }
        else
        {
          if(up>down)
          {
            note_loc++;
            base_staff_loc+=distance_between_lines/2;
          }
          else if(up==down)
          {
            if(abs(base_staff_loc-
corrected_index)<=((2*distance_between_lines)%3!=0)+2*distance_between_lines/3)
            {
              note_loc++;
              base_staff_loc+=distance_between_lines/2;
            }
          }
          else
          {
            note_loc+=2;
            base_staff_loc+=distance_between_lines;
          }
        }
        i=1;
      }
      else
      {
        if(base_staff_loc>corrected_index)
        {
```

```
      note_loc-=2;
      base_staff_loc-=distance_between_lines;
    }
    else
    {
      note_loc+=2;
      base_staff_loc+=distance_between_lines;
    }
    i=1;
  }
}
printf("*%d* ",note_loc);
note_top=base_staff_loc-distance_between_lines/2-distance_between_lines%2;
note_bottom=base_staff_loc+distance_between_lines/2+distance_between_lines%2;
no_stem_up=0;
no_stem_down=0;
stem=0;
for(scan=0;scan<=12*distance_between_lines;scan+=4)
{
  if(note_top-3*distance_between_lines>0)
  {
    read_extmem(&temp,pex_note_hist+4*note_top-scan,1);
    if(!temp)
    {
      no_stem_up++;
    }
  }
  else
  {
    no_stem_up=3*distance_between_lines;
  }
  if(note_bottom+3*distance_between_lines<height)
  {
    read_extmem(&temp,pex_note_hist+4*note_bottom+scan,1);
    if(!temp)
    {
      no_stem_down++;
    }
  }
  else
  {
```

```
    no_stem_down=3*distance_between_lines;
  }
}
if((no_stem_up<distance_between_lines)||(no_stem_down<distance_between_lines))
{
  stem=1;
}
if(stem)
{
  if(!(note_loc%2))
  {
    max=0;
    for(i=note_top*4;i<=note_bottom*4;i+=4)
    {
      read_extmem(&temp,pex_copy_of_hist+i,1);
      if(max<temp)
      {
        max=temp;
        index=i;
      }
    }
    index+=note_top*4;
    note_top=index-2*distance_between_lines-4*distance_between_lines%2;
    note_bottom=index+distance_between_lines*2+4*distance_between_lines%2;
    hollow=0;
    read_extmem(&temp,pex_copy_of_hist+note_top,1);
    prev_up=temp;
    read_extmem(&temp,pex_copy_of_hist+note_bottom,1);
    prev_down=temp;
    for(scan=0;scan<note_bottom-index;scan+=4)
    {
      read_extmem(&temp,pex_copy_of_hist+note_top+scan,1);
      current_up=temp;
      if((current_up<=prev_up)&&(prev_up>measure_line_thickness))
      {
        hollow++;
      }
      prev_up=current_up;
      read_extmem(&temp,pex_copy_of_hist+note_bottom-scan,1);
      current_down=temp;
      if((current_down<=prev_down)&&(prev_down>measure_line_thickness))
```

```
        {
          hollow++;
        }
        prev_down=current_down;
      }
      if(hollow<space_between_lines/3)
      {
        stem=2;
      }
    }
    else
    {
      filled=0;
      read_extmem(&temp,pex_copy_of_hist+corrected_index,1);
      prev_up=temp;
      prev_down=temp;
      for(scan=4;scan/2<=space_between_lines;scan+=4)
      {
        read_extmem(&temp,pex_copy_of_hist+corrected_index-scan,1);
        current_up=temp;
        if((current_up<=prev_up)&&(current_up<note_thickness))
        {
          filled++;
        }
        prev_up=current_up;
        read_extmem(&temp,pex_copy_of_hist+corrected_index+scan,1);
        current_down=temp;
        if((current_down<=prev_down)&&(current_down<note_thickness))
        {
          filled++;
        }
        prev_down=current_down;
      }
      if(filled>distance_between_lines/2+distance_between_lines%2)
      {
        stem=2;
      }
    }
  }
  printf("%d ",stem);
}
```