December 17, 2002

Dr. Andrew Rawicz
School of Engineering Science
Simon Fraser University
Burnaby, B.C., V5A 1S6

Re: ENSC340 Project Post Morterm Document for SmileyBaby Mobile

Dear Dr. Rawicz:

The following document, Post Morterm for SmileyBaby Mobile, lists the current and future states of the product, the deviation, the budget and time constraints and inter-personal experiences.

LittleFellows is comprised of four fun loving but focused energetic 3rd year engineering science students: Shona Huang, Marjan Houshmand, Farnam Mohasseb, and Farhood Hashmi. If you have any questions concerning our proposal, please feel free to contact us by email at ensc340-group@sfu.ca

Sincerely,

Marjan Houshmand

Marjan Houshamnd
Chief Executive Officer
LittleFellows Inc.

Enclosure: Post Morterm for SmileyBaby Mobile

Post Morterm for SmileyBaby Mobile

Team Project: *Farnam Mohasseb*
Marjan Houshmand
Shona Huang
Farhud Hashemian

Submitted to: *Dr. Andrew Rawicz*
Steve Whitmore
School of engineering science

Issued date: *December 17, 2002*

Table of Contents

Introduction

Officially, we have been working on our project, SmileyBaby Mobile, for the past thirteen weeks. Not only, we have gained technical, and non-technical skills, but also an amazing group work dynamic has been established amongst us. Truly, we did choose an ambitious project; however it is very rewarding and exciting to be able to demonstrate the functionality of our product. This report reveals both the current and future states of SmileyBaby, deviations and constraints, and our personal experiences.

Current State of the Device

As stated in our project proposal, SmileyBaby Mobile constantly monitors the nursery, and differentiates the baby's crying pattern from other common-household sounds and noises such as talking, music, pets, and the occasional car driving by. In the situation where the baby cries, our device switches on the mobile, which also plays a selection of music determined by the parents with the intent of comforting the child. The basic idea has been shown in Figure 1.

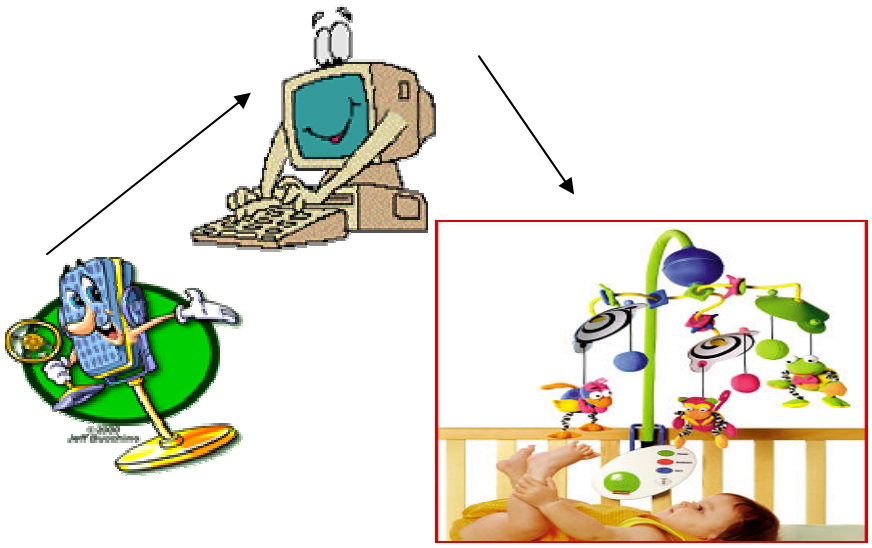

Figure 1, The Basic Overview of the System

In this section, we explain the current state of the device by referring to the current state of each blocks of Figure 1. The first stage is obtaining the input. For our project, we do not use a microphone. Instead, we use lining to receive sounds though a CD-player. This idea can be better explained using Figure 2.

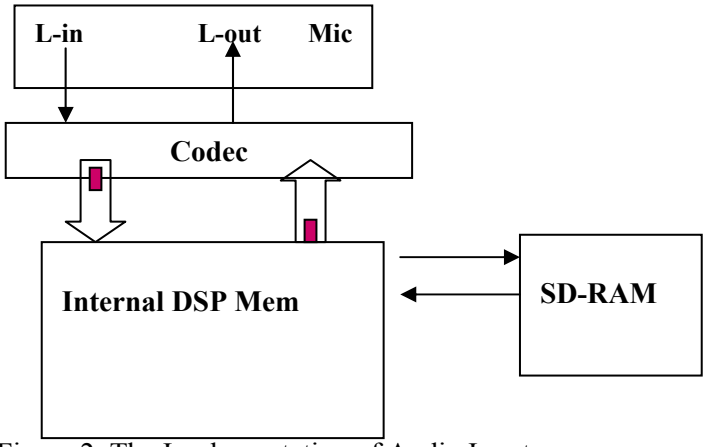

Figure 2, The Implementation of Audio Input

The second stage contains the main process, detecting the sleeping rabbit. For this purpose, the data will go through a low pass filter, envelope detector, average moving filter, and finally the pattern detector. The sleeping rabbit detector looks for the first peak and stores its information. Next, after finding the second peak, it compares the information of both peaks to determine the pattern. Once the pattern is recognized, it passes on the necessary data to the decision maker. At this point, the decision maker will turn on/off the mobile. Figure 3 will illustrate the software-hardware interfacing.

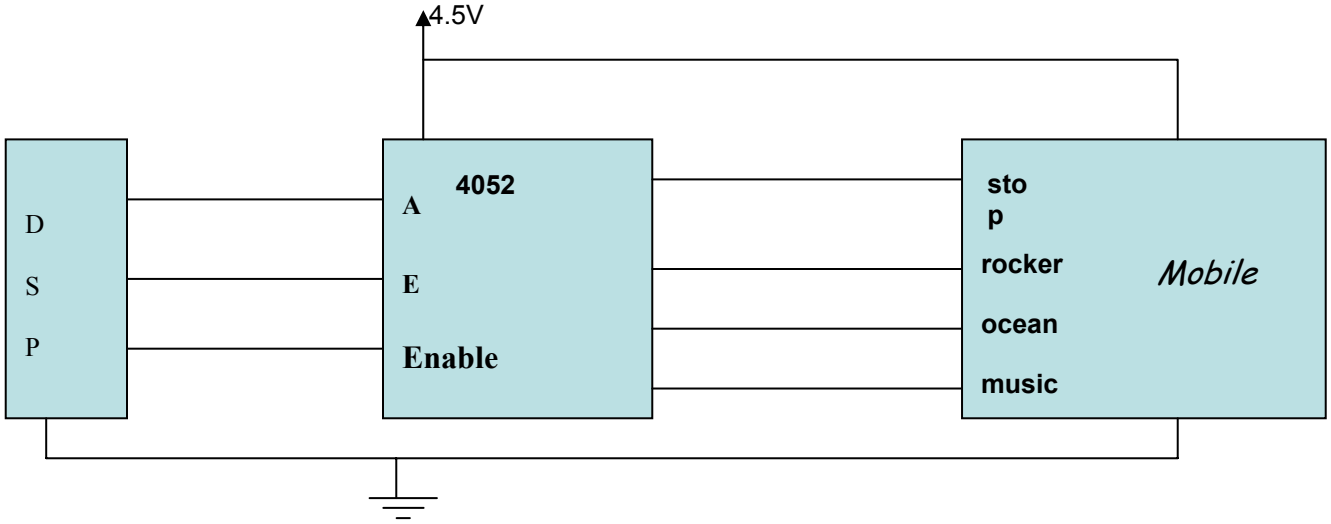

Figure 3, The Hardware-Software Interface

Deviation of the Device

Overall System

We achieved what we planned as far as functionality is concerned. We did not package our device because of time limitations. A more general interface unit may be possible in order to more easily adapt to a wider variety of OEM baby mobile products and other toys. At the moment, our prototype consists of an input device (a CD player), DSP evaluation board, and an interface board connected to an OEM baby mobile. Details of the deviations of each part of our project will be discussed in the following sections.

Signal Acquisition

We originally planned to use a dynamic microphone and place it directionally pointing toward the baby with the crib, strategically hidden; then connect this input signal to the general-purpose audio in jack on the DSP evaluation board. However, the input is not filtered against spurious noise that creates problems for the software in distinguishing desired audio content from broadband noise. The attempt to use a dynamic microphone created an excessive amount of noise, preventing the A/D converter from correct operation. To prove the concept of our technology, we decided to connect the line out from a CD player directly to the audio input of the evaluation board. Even so, the short connection wires pick up a lot of noise from the surrounding atmosphere; but we managed to filter most of the noise out via software and obtained acceptable data for further processing. A solution would be to use an electrets condenser microphone with filtering, prior to passing signal into the DSP. Successful interpretation of input data must be free of noise prior to inputting to the DSP. Failure to do so will prevent the DSP from correctly processing the information.

System Powering

At the current state, the three major parts of our system obtained power from separate sources. The DSP evaluation board is plugged into a computer mother board and receive power there. The DSP board by TI does not supply DC voltage to the OEM baby product. The Mobile and the interface circuit operate on three C type batteries. In this case, signal logic levels are compatible without level translation. Furthermore, the input sound is provided by a CD player which operates on batteries inside the player.

Signal Processing

In order to implement our signal processing we had to consider many algorithms since we had to minimize the required computation power yet not decrease our hit rate. We achieved what was originally planed and managed to distinguish all the cries form other surrounding noise.

Interface Circuit

We originally planned to use a relay to control the OEM baby mobile by disabling through our DSP output the on/off switch. Relays draw excessive current and would deplete the DC battery supply in the OEM unit. We chose to enable the on/off functions within the OEM unit with a simple logic level voltage implemented through a CMOS 4052 dual 1 of 4 switches.

We chose a CMOS family because it is more tolerant of different supply voltage (3-18V) and because it has lower power demands. We decided that two output signals from the DSP chip were sufficient to select the songs available for the OEM mobile. The connection to the evaluation board with single wires is convenient but messy. The female type connecter that we required to plug into the DSP board was not available in town, and due to funding limitation we did not order it from on-line stores. Our SmileyBaby Mobile also presents some connection problems when connecting to some OEM baby mobile products because we don't have enough information on circuitry inside OEM and power supply voltage levels.

Future Plans

SmileBaby Mobile is capable of being improved a lot. There are several possibilities we are considering for its future plan. First, we will have two different models: the Classic SmileyBaby and Advanced SmileyBaby. Our current state is our classic model. The Advanced SmileyBaby includes all the classic mode's functionalities plus additional features and abilities. Inside the digital signal processor, after being recognized as a baby's cry, the signal goes on for further processes. Then, the system tries to match the cry with the samples stored in its memory to decide what type of cry it is. Next, through LCD or any other user interface devices, the chip displays its decision regarding the cry for parents to see.

Furthermore, parents are capable of programming the chip to play the child's favorite music at a specific time. This will be adapted using the internal timer of the chip.

After the time has expired, an interrupt signal will go on, and as a consequence the music will be played.

In addition, we are hoping with a little research to make SmileyBaby act as a bio-feedback mechanism. This ability will help the parents to train the baby to sleep through the night without crying. As a whole, SmileyBaby will be designed in such a way to comfort both parents and the babies.Budgetary and Time Constraints

Budget

Table 1 indicates the estimated cost and the cost up to December 16th, 2002.

Required Material	*Estimated Cost*	*Actual Cost*
Two digital to analog converters (D/A)	**$80**	**No need**
Two analog to digital converters (A/D)	**$80**	**No need**
Two bus controllers	**$140**	**No need**
Three digital signal processing chip	**$240**	**Borrowed**
Two microphones	**$10**	**Borrowed**
Two headphones	**$10**	**Borrowed**
Two mobiles	**$120**	**$60**
Cables/Wires	**$30**	**Borrowed**
Miscellaneous	**$100**	**$30**
Total	**$810**	**$90**

Table 1: Estimated Cost vs. Actual Cost

As the above Table illustrates, our estimated cost was a lot more than the actual amount we spent to develop SmileyBaby. This is because we borrowed an advanced DSP chip, and it contained most of the components we needed. This DSP chip contained a D/A, an A/D and a bus controller. Consequently, there was no need to purchase these items separately. However, we were also able to borrow the PCB (hardware) design and fabrication equipment, a microphone, a headphone and cables as well. In other words, the only thing that we were required to purchase was the actual mobile. The estimated budget for the miscellaneous category was also much less because we overestimated this cost in our proposal.

Time

The Grant Chart below indicates the start and end dates of various tasks we perform at different stages of SmileyBaby development process. The blue print indicates the expected time of completion and the red print indicates the slippage time.

ID	Task name	Week of September				Week of October				Week of November				Week of December			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	Research																
2	Proposal																
3	Functional Specification																
4	Design Specification																
5	Implementation/Integration																
6	Debugging/Modification																
7	Documentation/Website																
8	Process Report																

Table 2: Grant Chart

Since we were the pioneers at this application, we had to do an extensive research and data gathering before we could initiate any analysis. As the above table illustrates, the research took a great deal of time and it interfered with other documentations. In fact, our design specification could not have been completed until the third week of November. Moreover, the implementation process had to be halted until we gathered enough data. In addition, the DSP chip was quite advanced, and we had underestimated the complexity of working with it. This also caused some delays in the implementation process.

However, as much frustrating, time consuming and difficult the tasks appeared, we all kept our focus and composer to complete this project on time. We did our best to adhere with our schedule, but the R&D was underestimated and took longer than we expected. Consequently, we were behind in our schedule most of the time. Nonetheless, we did finish our project in early December and were able to demo it on Monday, December 16th, 2002.

Inter-Personal and Technical Experiences

Marjan Houshmand

I enjoyed working with this group tremendously. Not only we have accomplished what we intended to, but also a strong group bound has been created, which will help us in the future. I believe, as a group, we stand a very good chance in succeeding as a company.

During this semester, I have gained a lot of personal and technical experience. I have learned about the DSP technology and how powerful but unfriendly sometimes they are! It took us a while, in order to be able to work with the chip. In addition, I exercised my C programming skill. Most of our time was spent on debugging the code or trying to understand how to use the DSP chip.

Besides the technical implementation of the product, a great deal of research was involved, specially in the beginning of the project. These tasks were more fun comparing to staring at the computer screen for hours and hours. We had to obtain various baby's crying samples from different sources. And that itself was a quite challenge. Personally, I take pleasure in dealing with people and fortunately this project included both business and technical aspects of the product. It was a pleasant experience being a part of LittleFellows Inc.

Farhud Hashemian

I believe this project provided the best opportunity for me to learn about DSPs, perform advance C programming and baby-cry signal analysis. By programming the Texas Instrument xC60 DSP chip, I learn how a digital signal processing system communicates with other component on a circuit, and how it performs its computations. Moreover, since I was dealing with an advanced DSP chip, I had to raise my C programming skills and write more sophisticated codes. The coding style that we had to follow for this application was very different from the regular coding I did for my other engineering courses. In addition, this project helped me increase my knowledge about baby-cry signals and their characteristics. It was also a tangible proof for me that baby-cry signals are different and exhibit certain unique patterns.

Beside the technical knowledge that I gained by developing SmileyBaby, I really enjoyed working with my group members. I believe this was one the finest groups I have worked with for the past few years. Even though we went through some rough and frustrating

periods during the cycle of development, all of us kept our focus and were determined that we could get this done. And in the end, we came up on top and successful.

Shona Huang

The last 13 weeks must be a flash-forward to the life of real engineer working on a team-designed project. This project surely must be a precursor to the interpersonal and technical challenges and frustrations every engineer faces when attempting to produce a viable prototype in a given amount of time. So what did I gain from this experience?!

I learned that no matter how tired my body and mind were feeling, how daunting the technical problem, how endless the documentation, I could always find from somewhere within me the energy to smile and continue. In addition, I am now somewhat more familiar with A/D conversion, Fast Fourier Transform algorithms and the importance a DSP plays in design. Even though the TI evaluation kit is not the friendliest of devices for a beginner, I am content to have had the opportunity to use a DSP. The digital signal processing algorithms our project employed were simple by comparison to those used in commercial products but I can see advantages a DSP has over a microprocessor when signal processing is involved. Some things that were murky prior to beginning this venture are now coming into focus. The RISC like architecture, parallel processing, internal memory organization and specialized units such as multipliers /accumulators allow the DSP to be 2 to 3 times faster than a microprocessor. Why? Digital signal processing requires a large amount of real –time calculations. For example, a 386 processor would require about 3mS to complete a Fast Fourier Transform for N=1024. The same Transform with a DSP can be completed in less than 1.5mS. Multiplication is the major task of a DSP. It seems that the most common operation in digital signal processing is the sum of products calculation $S = \sum a\ b$. Addition needs 3 clock cycles in a 8086 micro, while multiplication requires 130-160 clock cycles. To increase speed, a DSP has many specialized arithmetic logic units within such as multipliers and accumulators in order to complete both multiplication and addition in one clock cycle. This feature of the DSP was needed for LittleFellows.

Noise became a persistent irritant for me in this endeavor. Noise had to be reduced both at the i/p to the DSP and the o/p interface to the OEM baby mobile. I was unable to achieve the type of i/p signal needed from the microphone, but now know what the level of the signal must be in order that the A/D can make sense of the audio content. It must have a dynamic range of 70 dB yet provide a gain of 60 dB. This would require a high performance preamp similar to those manufactured by TDL Technology.

The connection to the OEM mobile created it's own set of noise problems. The songs played by the drop chip within the mobile unit became superimposed on the logic level signals needed to control the functions and were very difficult to remove without altering the original content of the music. It would have been faster and more productive to record higher fidelity music on digital storage devices rather than use the electronics supplied in the mobile.

Working in a group meant compromise, patience, and understanding. I experienced learning to sort through different opinions, recording/writing styles and seeking accountability for meeting deadlines. Problems are problems and there will always be someone to help work them out. Problems are more easily solved with many heads together rather than one head alone. Working with a group was a good experience for me.

Farnam Mohasseb

This was the most challenging project I ever experienced. Unlike all other project I've done in the past, there were no certain paths or procedures to follow. There were points in the project that we countered problems that seemed to have no feasible solutions. Yet we did not give up and continued. I believe, it was the group dynamic that played an extraordinary role in the project's success. It was assuring to see your group member still smiling after working on frustrating problem overnight. Without such a n amazing group dynamic, the project would have been impossible.

This project provides me with an excellent opportunity to learn. Since we are the pioneer in this field we had to do a lot research to achieve this project. As result I learned a lot about everything not just C programming skills. I learned more about baby's psychologies, much more advance C programming and various signal processing concepts. Dealing with Texas instrument C6000 series DSP, helped me to understand and learn and implement many algorithm that is being used in the industry. Completing this project was not an easy task, but the learning experience was worth the effort, and I would have done the same if I was starting this project again.

```
/*****************************************************/
// add this to command file
/*
        .my_sect1:  {} > SDRAM1
        .my_sect2:  {} > SDRAM1
*/
/*****************************************************/
/* "@(#) DSP/BIOS 4.51.0 05-23-01 (barracuda-i10)" */
/*
 *  ======== audio.c ========
 */

#include <std.h>
#include <stdio.h>
#include <pip.h>
#include <log.h>
#include <trc.h>
#include <hst.h>
#include <mem.h>

#include <string.h>     /* memset() */
#include <stdarg.h>     /* var arg stuff */

#include "dss.h"
#include "audio.h"
#include "ProjectCons.h"



Int PHASE = 0;              /* number of samples of phase difference */
Uns ProbePointer[32];

int NewDataFlag;

Uns LuckyArray[32];

#pragma DATA_SECTION(bufferin, ".my_sect1");
signed short bufferin[TheBufferSize];

#pragma DATA_SECTION(bufferout, ".my_sect2");
signed short bufferout[TheBufferSize];

static far unsigned long RxBufInPtr;
static far unsigned long RxBufOutPtr;
static far unsigned long TxBufInPtr;
static far unsigned long TxBufOutPtr;
static far int RequestProcess = 0;
static int my_start_up;
static unsigned long MyGlobalCounter;
static int SldListFlag;
static int SldListPtr;
static int MobileStatus;
unsigned long SldListArray[3];


static Uns allocBuf(PIP_Obj *out, Uns **buf);
static Void freeBuf(PIP_Obj *out);
static Uns getBuf(PIP_Obj *in, Uns **buf);
static Void process(Uns *src, Int size, Uns *dst);
static Void putBuf(PIP_Obj *out, Uns size);
static Void writeBuf(PIP_Obj *out, Uns *buf, Uns size);
//static Void MyForceRestart();
static void mytest();

static Int loadVal = 0;
static PIP_Obj *hostPipe = NULL;


void mytest()
{
    int k;
    bufferout[0] = 0;
    bufferout[1] = 1;
    bufferout[2] = 2;
    bufferout[3] = 3;
    bufferout[4] = 4;
    for(k=0;k<100; k++)
```

```
    {
        bufferout[(k+4) % BufferSize] = k+4;
    }
    /*
    while (1)
    {
    }*/
}

/*
 *  ======== main ========
 */
Void main()
{
    Int size;
    Uns *buf;

/*****farhud****/
/*initialize
call sld function

while(1)
{
};

/*****end****/


/*    while(1)
    {
        mobile_on();
        mobile_off();
    };
*/


#if AUDIO_USEHST
    hostPipe = HST_getpipe(&hostOutputProbe);
#endif

   LuckyArray[0] = 0x00004000;
   LuckyArray[1] = 0x00000000;
   LuckyArray[2] = 0x00000000;
   LuckyArray[3] = 0x00000000;
   LuckyArray[4] = 0x00000000;
   LuckyArray[5] = 0x00000000;
   LuckyArray[6] = 0x00000000;
   LuckyArray[7] = 0x00000000;
   LuckyArray[8] = 0x00004000;
   LuckyArray[9] = 0x00000000;
   LuckyArray[10] = 0x00000000;
   LuckyArray[11] = 0x00000000;
   LuckyArray[12] = 0x00000000;
   LuckyArray[13] = 0x00000000;
   LuckyArray[14] = 0x00000000;
   LuckyArray[15] = 0x00000000;
   LuckyArray[16] = 0x00004000;
   LuckyArray[17] = 0x00000000;
   LuckyArray[18] = 0x00000000;
   LuckyArray[19] = 0x00000000;
   LuckyArray[20] = 0x00000000;
   LuckyArray[21] = 0x00000000;
   LuckyArray[22] = 0x00000000;
   LuckyArray[23] = 0x00000000;
   LuckyArray[24] = 0x00004000;
   LuckyArray[25] = 0x00000000;
   LuckyArray[26] = 0x00000000;
   LuckyArray[27] = 0x00000000;
   LuckyArray[28] = 0x00000000;
   LuckyArray[29] = 0x00000000;
   LuckyArray[30] = 0x00000000;
   LuckyArray[31] = 0x00000000;

    RxBufInPtr = 32;
    RxBufOutPtr = 32;
    TxBufInPtr = 0;
    TxBufOutPtr = 0;
    my_start_up = 1;
```

```
    MyGlobalCounter = 0;
    MobileStatus = 0;

    SldListPtr = 0;
    SldListFlag = 0;
    SldListArray[0] = 0;
    SldListArray[1] = 0;
    SldListArray[2] = 0;
    myfilter_init();
    envelope_init();
    sld_init();
    DSS_init();     /* Initialize codec and serial port */

    //Filter_init();//Initilize lowpass filter
/*
    LOG_printf(&trace, "Audio example started!!\n");

    LOG_printf(&trace, "Short signed is %i\n", sizeof(signed short));
    LOG_printf(&trace, "int is %i\n", sizeof(int));
    LOG_printf(&trace, "Uns is %i\n", sizeof(Uns));


    LOG_printf(&trace, "first part is %i\n", (short signed) LuckyArray[24]);

    for (size = 0; size<4; size++)
    {
        temp1[size] = (short signed) LuckyArray[24];
    }

    temp_value =  (Uns) temp1[1];
    temp_value &= 0x0000FFFF;
    LOG_printf(&trace, "reconstruct part is %i\n", temp_value);

*/
    /* Prime output buffers with silence */
    size = allocBuf(&DSS_txPipe, &buf);
    memset(buf, 0, size * sizeof (Uns));
    putBuf(&DSS_txPipe, size);

    size = allocBuf(&DSS_txPipe, &buf);
    memset(buf, 0, size * sizeof (Uns));
    putBuf(&DSS_txPipe, size - PHASE);

    /* Fall into BIOS idle loop */
    return;
}

/*
Void MyForceRestart()
{
    RxBufInPtr = 32;
    RxBufOutPtr = 32;
    TxBufInPtr = 0;
    TxBufOutPtr = 0;
    my_start_up = 1;
    MyGlobalCounter = 0;

    SldListPtr = 0;
    SldListFlag = 0;
    SldListArray[0] = 0;
    SldListArray[1] = 0;
    SldListArray[2] = 0;
    myfilter_init();
    envelope_init();
    sld_init();
}
*/

/*
 *  ======== audio ========
 */
Void audio(PIP_Obj *in, PIP_Obj *out)
{
    Uns *src, *dst;
    Uns size;

    if (PIP_getReaderNumFrames(in) == 0 || PIP_getWriterNumFrames(out) == 0) {
        error("Error: audio signal falsely triggered!");
```

```
    }

    /* get input data and allocate output buffer */
    size = getBuf(in, &src);
    allocBuf(out, &dst);

    /* process input data in src to output buffer dst */
    process(src, size, dst);

    /* check for real-time error */
    if (DSS_error != 0) {
        LOG_printf(&trace,
            "Error: DSS missed real-time! (DSS_error = 0x%x)", DSS_error);
        loadVal -= 1;
        DSS_error = 0;
    }

    /* output data and free input buffer */
    putBuf(out, size);
    freeBuf(in);
}

/*
 *  ======== error ========
 */
Void error(String msg, ...)
{
    va_list va;
    va_start(va, msg);

    LOG_error(msg, va_arg(va, Arg));    /* write error message to sys log */
    LOG_disable(LOG_D_system);          /* stop system log */

    for (;;) {
        ;                               /* loop for ever */
    }
}


void myIDLLoop()
{
    int i;
    unsigned long incremente;
    far unsigned long temp;
    signed short temp_array[64];

    unsigned long diff1 = 0;
    unsigned long diff2 = 0;

    if(RequestProcess == 1)
    {

        if ((my_start_up == 1) && (TxBufInPtr > 0x2000))//BufferSize))
        {
        // copy data
        //  WriteDaFile(&bufferout[0]);
            //mytest();
//          temp_array[0] = 0x1111;
//          LOG_printf(&trace, "buffer value is %i", bufferout[TxBufInPtr % BufferSize]);
//          LOG_printf(&trace, "pointer value is %i", (TxBufInPtr % BufferSize));
//          LOG_printf(&trace, "buffer[0] value is %i", bufferout[0 % BufferSize]);
//          LOG_printf(&trace, "buffer[1] value is %i", bufferout[1 % BufferSize]);
//          LOG_printf(&trace, "buffer[2] value is %i", bufferout[2 % BufferSize]);
//          LOG_printf(&trace, "pointer to b[2,BS] is %x", &bufferout[(2 % BufferSize)]);
//          LOG_printf(&trace, "pointer to b[2] is %x", &bufferout[2]);
//          LOG_printf(&trace, "pointer to b[3,BS] is %x", &bufferout[(3 % BufferSize)]);
//          LOG_printf(&trace, "pointer to b[3] is %x", &bufferout[3]);
//          LOG_printf(&trace, "[3, BS] is %x", (3 % BufferSize));
//                      LOG_printf(&trace, "[3, 64K] is %x", (3 % (64*1024)));
//          LOG_printf(&trace, "3 is %x", (3));
//          for(i=0; i< 0x2000; i++)
//          {
//              temp_array[0] = bufferout[(i%BufferSize)];
//              temp_array[0] += 1;
//              bufferout[(i)] = temp_array[0];
//          }
//          temp_array[0] &= 0xFFFF;
            //bufferout[1] = temp_array[0];
```

```
        my_start_up = 0;
    }

    MyGlobalCounter++;

    if ((MobileStatus ==1) && (SldListPtr > 2) && (MyGlobalCounter - SldListArray[(SldListPtr-2)%3
    {
        MobileStatus = 0;
        mobile_off();
        LOG_printf(&trace, "turning off the mobile");
    }


    for (i = 31; i >= 0; i--)
    {
        incremente = envelope(bufferin[RxBufOutPtr++ % BufferSize], &bufferout[0], TxBufInPtr);
/*      if(incremente == 0xffff) // error require force restart
        {
            MyForceRestart();
            RequestProcess = 0;
            return;
        }
*/
        TxBufInPtr = TxBufInPtr + incremente;
        myfilter(incremente, &bufferout[0], TxBufInPtr);
        if (sld(incremente, &bufferout[0], TxBufInPtr) == 1)
        {
            SldListFlag++;
            SldListArray[SldListPtr++%3] = MyGlobalCounter;

            //WriteDaFile(&bufferout[0]);
            LOG_printf(&trace, "Global counter is %u", MyGlobalCounter);

            if((SldListFlag >= 3) && (MobileStatus == 0))
            {
                diff1 = SldListArray[(SldListPtr-1)%3] - SldListArray[(SldListPtr-2)%3];
                diff2 = SldListArray[(SldListPtr-2)%3] - SldListArray[(SldListPtr-3)%3];
                /*
                LOG_printf(&trace," diff1 ,%u", diff1);
                LOG_printf(&trace," diff2 ,%u", diff2);
                LOG_printf(&trace," sldp0 ,%u", SldListArray[SldListPtr%3]);
                LOG_printf(&trace," sldp1 ,%u", SldListArray[(SldListPtr-1)%3]);
                LOG_printf(&trace," sldp2 ,%u", SldListArray[(SldListPtr-2)%3]);
                LOG_printf(&trace," sld0 ,%u", SldListArray[0]);
                LOG_printf(&trace," sld1 ,%u", SldListArray[1]);
                LOG_printf(&trace," sld2 ,%u", SldListArray[2]);
                */
                if((diff1 <= 4500) && (diff2 <= 4500))
                {
                    MobileStatus = 1;
                    LOG_printf(&trace," TURNING THE MOBILE ON");
                    mobile_on();
                }
            }
            //LOG_printf(&trace, "TXBufOutPtr %x", )
            //LOG printf(&trace, "TXBufOutPtr %x", (0x00420000 + ((TxBufInPtr%BufferSize)*2)-0x200
            //LOG_printf(&trace, "SLD DETECTED !!!!");
            //LOG_printf(&trace, "start probing!");
            //LOG_printf(&trace, "probing done");
            //mobile_on();
        }
    }



/*
    for (i = 31; i >= 0; i--)
    {
        RxBufOutPtr++;
        bufferout[TxBufInPtr++ % BufferSize] = (bufferin[RxBufOutPtr % BufferSize]);//x2/3)+(buffe
        //bufferout[TxBufInPtr++ % BufferSize] = bufferin[RxBufOutPtr++ % BufferSize];
        //temp_array[i+32] = bufferout[TxBufInPtr % BufferSize];
    }
*/
    temp = RxBufOutPtr;
    temp = temp - 32;
    for (i= 1; i<64 ; i++)
    {
```

```
            temp_array[i] = bufferout[temp++ % BufferSize];
        }


        RequestProcess = 0;
    }
}

/*
 *  ======== allocBuf ========
 */
static Uns allocBuf(PIP_Obj *out, Uns **buf)
{
    PIP_alloc(out);
    *buf = PIP_getWriterAddr(out);;
    return (PIP_getWriterSize(out));
}

/*
 *  ======== freeBuf ========
 */
static Void freeBuf(PIP_Obj *out)
{
    PIP_free(out);
}

/*
 *  ======== getBuf ========
 */
static Uns getBuf(PIP_Obj *in, Uns **buf)
{
    PIP_get(in);
    *buf = PIP_getReaderAddr(in);;
    return (PIP_getReaderSize(in));
}

/*
 *  ======== process ========
 */
static Void process(Uns *src, Int size, Uns *dst)
{
    Int i;
    signed short test_array[32];
    //Uns temp_value;
    //local_max = filter(src, size, dst);
    //LOG_printf(&trace, " %u local_max", local_max);
    //envelope(local_max);
    //src = &LuckyArray[0];
    for (i = size - 1; i >= 0; i--)
    {
        //dst[i] = src[i];
        //dst[i] = LuckyArray[i];
        test_array[i] = (signed short) src[i];
        bufferin[RxBufInPtr++ % BufferSize] = test_array[i];

        //temp_value =  (Uns) bufferin[RxBufInPtr % BufferSize];
        //temp_value &= 0x0000FFFF;
        //dst[i] = temp_value;
//        test_array[i] = bufferin[RxBufInPtr % BufferSize];
        //ProbePointer[i] = src[i];
    }

    for (i = size - 1; i >= 0; i--)
    {
        dst[i] = LuckyArray[i];

/*      temp_value = (Uns) bufferout[TxBufOutPtr++ % BufferSize];
        temp_value &= 0x0000FFFF;
        dst[i] = temp_value;*/
    }

    RequestProcess = 1;

    /* if hostPipe is non-NULL write data to this pipe */
    if (hostPipe != NULL) {
        writeBuf(hostPipe, dst, size);
    }
}
```

```
/*
 *  ======== putBuf ========
 */
static Void putBuf(PIP_Obj *out, Uns size)
{
    PIP_setWriterSize(out, size);
    PIP_put(out);
}

/*
 *  ======== writeBuf ========
 */
static Void writeBuf(PIP_Obj *out, Uns *buf, Uns size)
{
    Uns dstSize;
    Uns *dst;
    Int i;

    /* allocate outBuffer for output */
    dstSize = allocBuf(out, &dst);

    /* copy data to output buffer (but not more than output buffer size) */
    for (i = (dstSize >= size) ? size : dstSize; i > 0; i--) {
        *dst++ = *buf++;
    }

    /* put buffer to output pipe */
    putBuf(out, size);
}
```

```
#include <std.h>
#include <trc.h>
#include <log.h>
#include <swi.h>

//#include "sldcfg.h"
#include "sld.h"

extern far LOG_Obj trace;
const far unsigned long BufferSize4 = 64*1024;

Uns timer_value[2];
static signed short peak1, peak2;//, dumby_timer, noiseAmmunity;
static Uns timer;
//static int SLD_flag;
static int start_flag, count, low_counter1, low_counter2;

int sld(int num_new_memebers, signed short *buffer, unsigned long index);
int sld_process(signed short member);
int sld_calculate();
void sld_init();
void sld_reset();


//extern void c_put(Uns var);
//extern Uns c_get();

/*void main()
{
    int k;
    sld_init();
    LOG_printf(&trace, "In main");
    //initialize all members to 45!
    for(k=0; k < ECHOFRAMELEN; k++)
        c_put(45);

    //input 0,2,4,3,1
    for(k = 0; k < 3; k++)
    {
        c_put(2 * k);
    }
    for(k = 2; k > 0; k--)
    {
        c_put(2 * k - 1);
    }

    sld(5);

    //input 11 zeros
    for(k=0; k < 11; k++)
        c_put(0);

    sld(11);

    //input 0,2,4,6,8,7,3,1
    for(k = 0; k < 5; k++)
    {
        c_put(2 * k);
    }

    for(k = 4; k > 0; k--)
    {
        c_put(2 * k - 1);
    }

    sld(9);

    //input 7 zeros
    for(k=0; k < 7; k++)
        c_put(0);

    sld(7);

    //input a good SLD data : 0,2,4,6,8,7,5,3,1,0,2,4,3,1
    for(k = 0; k < 5; k++)
    {
        c_put(2 * k);
    }
```

```
    for(k = 4; k > 0; k--)
    {
        c_put(2 * k - 1);
    }

    sld(9);

    for(k = 0; k < 3; k++)
    {
        c_put(2 * k);
    }
    for(k = 2; k > 0; k--)
    {
        c_put(2 * k - 1);
    }

    sld(5);

    return;
}

void c_put(Uns var)
{
    c_buffer[write_index++ % ECHOFRAMELEN] = var;
    counter++;
}

Uns c_get()
{   Uns temp;
    if(counter == 0)
    {
        LOG_printf(&trace, "Circular Buffer is empty. There is nothing to return.");
        return 1;
    }
    else
    {
        counter--;
        temp = c_buffer[read_index++ % ECHOFRAMELEN];
        //LOG_printf(&trace, "%u member", temp);
        return  temp;
    }
}*/

void sld_init()
{
    start_flag=0;
    sld_reset();
}

void sld_reset()
{
    low_counter1 = 0;
    low_counter2 = 0;
    timer = 0;
    timer_value[0] = 0;
    timer_value[1] = 0;
    count = 0;
    peak1 = 0;
    peak2 = 0;
    //dumby_timer = 0;
    //noiseAmmunity = 0;
}


int sld(int num_new_memebers, signed short *buffer, unsigned long index)
{
    int sld_detected = 0;
    index = index - num_new_memebers;
    while(num_new_memebers)
    {
        if (sld_process(buffer[index++ % BufferSize4]) == 1)
            sld_detected = 1;
        num_new_memebers--;
    }
    return sld_detected;
}
```

```
int sld_process(signed short member)
{
    //LOG_printf(&trace, "%u member", member);

/*  if(dumby_timer >= dumby_timer_ther)
    {
        LOG_printf(&trace, "dummy timer ran out!! start_flag %d member %d", start_flag, member);
        sld_init();
    }*/

    if(start_flag == 0)
    {
        if ((member >= low_ther) && (low_counter1 >= 100))
        {
            start_flag = 1;
            low_counter1 = 0;
            //noiseAmmunity = 0;
            //LOG_printf(&trace, "starting");
        }
        else if(member >= low_ther)
        {
            low_counter1++;
        }
    }

    else
    {
        //low_counter1 = 0;
        if (member <= low_ther)
        {
            /*if (noiseAmmunity < 100)
            {
                noiseAmmunity++;
            }*/
            //else if (timer <= 500)
            //{
            //  dumby_timer++;
                //noiseAmmunity = 0;
        //  }

            /*if (noiseAmmunity < 100)
            {
                noiseAmmunity++;
            }
            else if(!peak2 && !count)
            {
                timer_value[count]=timer;
                timer=0;
                //count++;
                dumby_timer=0;
                noiseAmmunity=0;
            }*/
        /*  else if((timer < 500) && !peak1)
            {
                sld_init();
            }*/
            if(low_counter2 <= 200)
            {
                low_counter2++;
            }
            /*else if(peak <= high_ther)
            {
                sld_init();
            }*/
            //low_counter2 = 0;
            else if (((peak1 >= high_ther) && !count) ||
                     ((peak2 >= high_ther) &&  count))
            {
                //low_counter2 = 0;
                //LOG_printf(&trace, "before going to sld_calcutate %d", peak1);
                return sld_calculate();
            }
            else
            {
                sld_init();
            }
        }
```

```
        else if (member >= low_ther)
        {
            /*dumby_timer = 0;
            if (noiseAmmunity < 100)
            {
                noiseAmmunity++;
            }
            if(low_counter3 <= 100)
            {
                low_counter1++;
            }*/
            //else
            //{
                //low_counter1 = 0;
                low_counter2 = 0;
                timer++;
            //  noiseAmmunity = 0;
                if(!count && (member >= high_ther) && (peak1 < member))
                    {
                        peak1 = member;
                        //dumby_timer = 0;
                    }
                else if(count && (member >= high_ther) && (peak2 < member))
                    {
                        peak2 = member;
                        //dumby_timer = 0;
                    }
        //  LOG_printf(&trace, "%d state", state);
            /*  if(member >= high_ther)
                {
                //Store the max
                    if((count == 0) && (peak1 < member))
                    {
                        peak1 = member;
                    }
                else if(
                    if((count == 1) &&  (peak2 < member))
                    {
                        peak2 = member;
                    }
                }*/
            }
        //}
    }
    return 0;
}

int sld_calculate()
{
    //LOG_printf(&trace, "timer %u and count %d in sld_calculate", timer, count);
    //LOG_printf(&trace, "peak1 %d peak2 %d in sld_calculate", peak1, peak2);
    timer_value[count] = timer;
    timer = 0;
    start_flag = 0;
    count++;
    //noiseAmmunity = 0;
    //dumby_timer = 0;
    if(count == 2)
    {
        if((peak1 >= high_ther2) && (peak2 >= high_ther) &&
           ((timer_value[0] >= (2*timer_value[1])) && (timer_value[0] <= (12*timer_value[1]))) &&
           ((peak1 >= peak2) && (peak1 <= (6*peak2)))) /*&&
           ((timer_value[0] / timer_value[1]) <= time_ther2)*/
        {
            //SLD_flag = 1;
            LOG_printf(&trace, "%d peak1", peak1);
            LOG_printf(&trace, "%d peak2", peak2);
            LOG_printf(&trace, "%u timer_count[0]", timer_value[0]);
            LOG_printf(&trace, "%u timer_count[1]", timer_value[1]);
            LOG_printf(&trace, "SLD_flag set!");
            sld_init();
            return 1;
        }

        else
        {
            LOG_printf(&trace, "switching");
            //LOG_printf(&trace, "in switching peak1 %d is peak2 is%d", peak1, peak2);
```

```
            timer_value[0] = timer_value[1];
            peak1 = peak2;
            peak2 = 0;
            timer_value[1] = 0;
            count = 1;
        }
    }
    return 0;
}
```

```
#include <std.h>
#include <stdarg.h>
#include <log.h>
#include <trc.h>
#include "myfilter.h"

static int BufferInPtr;
//static int BufferOutPtr;
signed short buffer[filterLen];

static short signed filter_process(signed short new_value);


void myfilter_init()
{
    int i;
    BufferInPtr = 0;
    for (i= 0; i<filterLen; i++)
    {
        buffer[i] = 0;
    }

}

void myfilter(int num_new_members, signed short *buffer, unsigned long index)
{
    index = index - num_new_members;
    while(num_new_members)
    {
        buffer[index++ % BufferSize3] = filter_process(buffer[index % BufferSize3]);
        num_new_members--;
    }
}

short signed filter_process(signed short new_value)
{
    int i;
    signed short avg;
    buffer[BufferInPtr++ % filterCons] = new_value;
    for (i= 0; i<filterLen; i++)
    {
        avg += buffer[i]/filterCons;
    }
    if (avg>2500)
    {
        return 0;
    }
    return avg;
}
```

```
#include "timer.h"
#include "regs.h"

void mobile_on ();
void mobile_off ();

void mobile_on ()
{
    static int musicNum = 0;

    musicNum++;
    if ((musicNum%3) == 0)
    {
        TOUT_VAL(0,1);
        TOUT_VAL(1,0);
    }

    if ((musicNum%3) == 1)
    {
        TOUT_VAL(0,1);
        TOUT_VAL(1,1);
    }

    if ((musicNum%3) == 2)
    {
        TOUT_VAL(0,0);
        TOUT_VAL(1,1);
    }
}

void mobile_off ()
{
    TOUT_VAL(0,0);
    TOUT_VAL(1,0);
}
```

```
//#include <stdio.h>
#include <std.h>
#include <stdarg.h>
#include <log.h>
#include <trc.h>
#include "envelope_dec.h"
//#include "ProjectCons.h"
//#include "meowcfg.h"

//#define BufferSize2 64*1024

int i;
int temp;
signed short peak_value[Len];
signed short Last_Max;
signed short New_Max;
//extern signed short bufferout[BufferSize];
extern far unsigned long TxBufInPtr;
extern far LOG_Obj trace;



short signed sorted_array[Len];


static void env_rest();
static void interpolate();

/*
void main()
{
    int k;
    Uns testArray[10];
    testArray[0] = 10;
    testArray[1] = 0;
    testArray[2] = 50;
    testArray[3] = 0;
    testArray[4] = 10;
    testArray[5] = 30;
    testArray[6] = 0;
    testArray[7] = 20;
    testArray[8] = 0;
    testArray[9] = 10;
    temp =0;
    envelope_init(); //

    for (k = 0; k<9; k++)
    {
        envelope(testArray[k]);
    }

    for (k = 0; k<9; k++)
    {
        sorted_array[k] = c_get(9-k);
    }
    return;
}
*/
void envelope_init()
{
    Last_Max = 0;
    peak_value[0] = Last_Max;
    peak_value[1] = 1;
    i = 1;
    return;
}
/*
void send_data()
{
    int k;
    // send data (the first i-1 data are ready to go)
    for(k = 1; k<i; k++)
    {
//      c_put(peak_value[k]);
//      test_array[temp] = peak_value[k];
//      temp++;
//      LOG_printf(&trace, "Audio example started!!\n");
//      printf("location %i, has value %u", temp++, peak_value[k]);
```

```
    }

    Last_Max = New_Max;
    peak_value[0] = Last_Max;
    peak_value[1] = peak_value[i];
    i = 1;
    return;
}
*/
void interpolate()
{
    int j = 0;
    signed short average;

/*
    int temp_flag = 0;

    signed short temp_local[100];
    for (j = 0; j <= (i); j++)
    {
        temp_local[j] = peak_value[j];
        if (temp_local[j] < 0)
        ;
    }

    if (temp_flag == 1)
    {
        for (j = 0; j <= (i); j++)
        {
            LOG_printf(&trace,"location %i, value is %x", j, peak_value[j]);
        }
    }
*/


    New_Max = peak_value[i-1];


    if (New_Max > Last_Max)
    {
        average = (New_Max - Last_Max)/ (i-1);

        for (j = 1; j <= (i-1); j++)
        {
            peak_value[j] = Last_Max + (j * average);
        }
    }
    else
    {
        average = (Last_Max - New_Max)/ (i-1);

        for (j =1; j <= (i-1); j++)
        {
            peak_value[j] = Last_Max - (j * average);
        }
    }

/*  if (temp_flag == 1)
    {
        for (j = 0; j <= (i); j++)
        {
            LOG_printf(&trace,"location %i, value is %x", j, peak_value[j]);
        }
    }
    */
    /*
    for (j = 0; j <= (i); j++)
    {
        temp_local[j] = peak_value[j];
        if (temp_local[j] < 0)
        ;
    }*/
       //send_data();
}

void env_rest()
{
    Last_Max = New_Max;
```

```
    peak_value[0] = Last_Max;
    peak_value[1] = peak_value[i];
    i = 1;
}

unsigned long envelope(signed short local_value, signed short *buffer, unsigned long index)
{
    //static int MyResetFlag = 0;
    int k,l;
    unsigned long tempy=0;
    signed short filtered_value;
    int b=0;
    signed short average;

    signed short temp_array2[201];
    i++;
    peak_value[i] = local_value;

    if ((i-10) > Len)
    {
    //  for (b=0; b<200; b++)
    //      temp_array2[b] = peak_value[b];
        if (peak_value[i-1] < 0)
        {
            peak_value[i-1] = 0-peak_value[i-1];
        }
        //peak_value[i-1] = 10;
        peak_value[i] = peak_value[i-1] -1;
        peak_value[i-2] = peak_value[i-1] -1;

        //MyResetFlag =1;
        LOG_printf(&trace,"look at me later on");
        //return 0;
    }

    if (peak_value[i-1] <= 0)
    {
        return 0;
    }

    //checks to see if the middle number is the peak
    if ((peak_value[i-1]  >= peak_value[i]) &&
        (peak_value[i-1] >= peak_value[i-2]))
    {
        // calculate teh midpoints
        interpolate();
        //LOG_printf(&trace, "tempy is %i", tempy);
        //LOG_printf(&trace, "index is %i", index);
        //LOG_printf(&trace, "buffer value is %i", buffer[index % BufferSize]);
        // update the main array
        for(k = 1; k<i; k++)
        {
            // filtering process
            buffer[index++ % BufferSize2] = peak_value[k];
            tempy++;
        }

        // get ready for the  next peak
        //LOG_printf(&trace, "tempy is %i", tempy);
        //LOG_printf(&trace, "index is %i", index);
        //LOG_printf(&trace, "buffer value is %i", buffer[(index-k) % BufferSize]);
        env_rest();
        return tempy;
    }
    return 0;
}
```

```
/*
 *  Copyright 2001 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *  U.S. Patent Nos. 5,283,900  5,392,448
 */
/* "@(#) DSP/BIOS 4.51.0 05-23-01 (barracuda-i10)" */
/*
 *  ======== dss_evm62.c ========
 */

#include <std.h>

#include "dss.h"
#include "dss_priv.h"

#define RXINT_BIT 0x0100      /* assume DMA0 rx interrupt on INT 8 */
#define TXINT_BIT 0x0200      /* assume DMA1 tx interrupt on INT 9 */

/* CPLD (Complex Programmable Logic Device) register */
#define CPLD (*(volatile unsigned int *)0x01780000)

/* Codec Memory Mapped Registers */
#define IAR  (*(volatile unsigned int *)0x01720000) /* Index Address Reg */
#define IDR  (*(volatile unsigned int *)0x01720004) /* Indexed Data Reg */
#define SR   (*(volatile unsigned int *)0x01720008) /* Status Reg */
#define PIO  (*(volatile unsigned int *)0x0172000c) /* PIO Data Reg */

/* Codec Indirect Mapped Registers */
#define DAC_LEFT_CNTL   0x06  /* I6  - Left DAC Output Control */
#define DAC_RIGHT_CNTL  0x07  /* I7  - Right DAC Output Control */
#define FS_PDF_CNTL     0x48  /* I8  - FS & Playback Data Control (MCE set) */
#define PIO_CNTL        0x49  /* I9  - Interface Control (MCE set) */
#define MODE_CNTL       0x0c  /* I12 - Mode and ID Control */
#define SP_CNTL         0x50  /* I16 - Alternate Feature enable 1 (MCE set) */
#define LINE_LEFT_CNTL  0x12  /* I18 - Left Line Input Control */
#define LINE_RIGHT_CNTL 0x13  /* I19 - Right Line Input Control */
#define CDF_CNTL        0x5c  /* I28 - Capture Data Format (MCE set)*/

/* Codec Commands */
#define DAC_UNMUTE    0x00  /* Unmute DAC-to-mixer */
#define MODE2_ENABLE  0x40  /* Enable MODE 2 */
#define INIT_DONE     0x80  /* Initialization bit */
#define SP_32_ENABLE  0x0a  /* Serial Port (32 bits) */
#define SP_64E_ENABLE 0x02  /* Serial Port (64 bits enhanced) */
#define SP_64_ENABLE  0x06  /* Serial Port (64 bits enhanced) */
#define CDF_S_L16     0x50  /* CDF; Stereo, linear, 16-bit */
#define FS8_S_L16     0x50  /* FS: 8kHz, PDF; Stereo, linear, 16-bit */
#define FS44_S_L16    0x5c  /* FS: 44kHz, PDF; Stereo, linear, 16-bit */
#define PIO_ENABLE    0xc3  /* Enable PIO (capture and playback) */
#define MCE_RESET     0xbf  /* Reset Mode Control Bit (MCE) */

/* CPLD commands */
#define CODEC_DISABLE  0xf8
#define CODEC_ENABLE   0x04

/* Macros */
#define SETBITMASKCODEC(i,n)\
              IAR = (i);\
              IDR |= (n);

#define SETREGCODEC(i,n)\
              IAR = (i);\
              IDR = (n);

static void codecInit(void);  /* Configure Codec */

/*
 * ======== DSS_init ========
 */
void DSS_init(void)
{
    /* Configure codec */
    codecInit();

    /* Enable Transmit and Receive bits */
    MCBSP_enableRcv(DSS_hMcbsp0);
```

```
    MCBSP_enableXmt(DSS_hMcbsp0);

    /* Enable DMA-to-CPU interrupts */
    IER |= RXINT_BIT;   /* INT 8 */
    IER |= TXINT_BIT;   /* INT 9 */
}

/*
 * ======== codecInit ========
 */
static void codecInit(void)
{
    /* Reset and enable codec */
    CPLD &= CODEC_DISABLE;
    CPLD |= CODEC_ENABLE;

    /* wait for codec to finish initialization */
    while(IAR & INIT_DONE);

    /* Enable mode 2 */
    SETBITMASKCODEC(MODE_CNTL, MODE2_ENABLE);

    /* Set codec Serial port format */
    SETREGCODEC(SP_CNTL, SP_32_ENABLE);

    /* Set capture format */
    SETREGCODEC(CDF_CNTL, CDF_S_L16);

    /* Set playback format and sample rate */
    SETREGCODEC(FS_PDF_CNTL, FS8_S_L16);//FS44_S_L16);

    /* Enable bits in PIO (programmed I/O) */
    SETREGCODEC(PIO_CNTL, PIO_ENABLE);

    /* Reset Mode Control Bit (MCE) */
    IAR &= MCE_RESET;

    /* Unmute left/right DAC-to-mixer */
    SETREGCODEC(DAC_LEFT_CNTL, DAC_UNMUTE);
    SETREGCODEC(DAC_RIGHT_CNTL, DAC_UNMUTE);
}

/*
 *  ======== DSS_spWrite ========
 */
Void DSS_spWrite(Uns data)
{
    while ((MCBSP_RGETH(DSS_hMcbsp0, SPCR) & 0x20000) == 0);

    MCBSP_RSETH(DSS_hMcbsp0, DXR, data);
}

/*
 *  ======== DSS_spRead ========
 */
Uns DSS_spRead(Void)
{
    while ((MCBSP_RGETH(DSS_hMcbsp0, SPCR) & 0x2) == 0);

    return (MCBSP_RGETH(DSS_hMcbsp0, DRR));
}
```

```
/*
 *  Copyright 2001 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *  U.S. Patent Nos. 5,283,900  5,392,448
 */
/* "@(#) DSP/BIOS 4.51.0 05-23-01 (barracuda-i10)" */
/*
 *  ======== dss_dmacisr.c ========
 */

#include <std.h>
#include <pip.h>

#define _DMA_   1

#include "dss.h"
#include "dss_priv.h"

/*
 *  ======== DSS_dmaRxIsr ========
 */
void DSS_dmaRxIsr(void)
{
    PIP_put(&DSS_rxPipe);
    DSS_rxCnt = 0;

    DMA_RSETH(DSS_hDmaRint0, SECCTL, 0x08);
    DSS_rxPrime(TRUE);
}

/*
 *  ======== DSS_dmaTxIsr ========
 */
void DSS_dmaTxIsr(void)
{
    PIP_free(&DSS_txPipe);
    DSS_txCnt = 0;

    DMA_RSETH(DSS_hDmaXint0, SECCTL, 0x08);
    DSS_txPrime(TRUE);
}
```

```
/*
 *  Copyright 2001 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *  U.S. Patent Nos. 5,283,900  5,392,448
 */
/* "@(#) DSP/BIOS 4.51.0 05-23-01 (barracuda-i10)" */
/*
 *  ======== dss.c ========
 *  DSS provides a pipe(PIP) interface to the serial port audio data.
 *  DSS export two pipes: DSS_rxPipe and DSS_txPipe.  DSS_rxPipe can be
 *  read by the client to receive data from the serial port and
 *  DSS_txPipe can be written to by the client to send data to the
 *  serial port.
 *
 *  The input pipe (DSS_rxPipe) operates as follows:
 *
 *    Startup:
 *      BIOS startup
 *      ------------
 *      notifyWriter()
 *          |__DSS_rxPrime(0)
 *
 *
 *    Steady state:
 *      ISR(writer)              client(reader)
 *      -----------              --------------
 *      PIP_put()
 *          |__notifyReader()--> audio()
 *                                 |
 *      DSS_rxPrime(1)             |
 *          |__PIP_alloc()         |
 *             start DMA           |
 *                                 |__PIP_get()
 *                                    `process data`
 *                                    PIP_free()
 *                                        |__notifyWriter()
 *                                               |__DSS_rxPrime(0)
 *
 *  The output pipe (DSS_txPipe) operates as follows:
 *
 *    Startup:
 *      BIOS startup
 *      ------------
 *      notifyWriter()
 *          |__DSS_txPrime(0)
 *
 *
 *    Steady state:
 *      ISR(reader)              client(writer)
 *      -----------              --------------
 *      PIP_free()
 *          |__notifyWriter()--> audio()
 *                                 |
 *      DSS_txPrime(1)             |
 *          |__PIP_get()           |
 *             start DMA           |
 *                                 |__PIP_alloc()
 *                                    `write data`
 *                                    PIP_put()
 *                                        |__notifyReader()
 *                                               |__DSS_txPrime(0)
 */

#include <std.h>
#include <pip.h>
#include <log.h>
#include <trc.h>
#include <clk.h>

#include "dss.h"
#include "dss_priv.h"

#ifdef _EDMA_
#include <csl_cache.h>
#endif
```

```
Int     DSS_error = 0;

Int     DSS_rxCnt = 0;
Int     DSS_txCnt = 0;

Int     *DSS_rxPtr = NULL;
Int     *DSS_txPtr = NULL;

DSS_Obj DSS_config = {
        0                       /*  enable tracing */
};

/*
 *  ======= DSS_txPrime ========
 *  Called when DSS_txPipe has a full buffer to be transmitted
 *  (i.e., when notifyReader() is called) and when the DSS ISR
 *  is ready for more data.
 */
void DSS_txPrime(Bool calledByISR)
{
    PIP_Obj     *txPipe = &DSS_txPipe;
    static int delay = 1;       /* or 2, 3, etc. */
    static Int nested = 0;

    LOG_message("DSS_txPrime(): %u", CLK_gethtime());

#if !(defined(_EDMA_) || defined(_DMA_))
    if (calledByISR) {
        DSS_computePhase(PIP_getWriterSize(txPipe));
    }
#endif

    if (nested) {       /* prohibit recursive call via PIP_get() */
        return;
    }

    if (delay) {      /* ensure that output does not start too soon */
        delay--;
        return;
    }

    nested = 1;

    if (DSS_txCnt == 0 && PIP_getReaderNumFrames(txPipe) > 0) {
        Int count, i;
        MdInt *dst;
        PIP_get(txPipe);

        /* must set 'Ptr' before 'Cnt' to synchronize with isr() */
        DSS_txPtr = PIP_getReaderAddr(txPipe);

        /* ensure bit 0 of DAC word is 0; AIC uses bit 0 to request control */
        count = (sizeof(Int) / sizeof(MdInt)) * PIP_getReaderSize(txPipe);
        for (i = count, dst = (MdInt *)DSS_txPtr; i > 0; i--) {
            *dst++ = 0xfffe & *dst;
        }

        DSS_txCnt = count;

#if defined(_EDMA_) || defined(_DMA_)
        DSS_dmaTxStart(DSS_txPtr, DSS_txCnt);
#endif
    }
    else if (calledByISR && (DSS_error & DSS_TXERR) == 0) {
        if (DSS_config.enable) {
            TRC_disable(TRC_GBLTARG);
        }
        LOG_error("transmit buffer underflow: %u", CLK_gethtime());
        DSS_error |= DSS_TXERR;
    }

    nested = 0;
}

/*
 *  ======= DSS_rxPrime ========
 *  Called when DSS_rxPipe has an empty buffer to be filled;
 *  e.g., when notifyWriter() is called) and when the DSS ISR
```

```
 *  is ready to fill another buffer.
 */
void DSS_rxPrime(Bool calledByISR)
{
    PIP_Obj     *rxPipe = &DSS_rxPipe;
    static Int  nested = 0;

    LOG_message("DSS_rxPrime(): %u", CLK_gethtime());

#if !(defined(_EDMA_) || defined(_DMA_))
    if (calledByISR) {
        DSS_computePhase(PIP_getWriterSize(rxPipe));
    }
#endif

    if (nested) {       /* prohibit recursive call via PIP_alloc() */
        return;
    }
    nested = 1;

    if (DSS_rxCnt == 0 && PIP_getWriterNumFrames(rxPipe) > 0) {
        PIP_alloc(rxPipe);

        /* must set 'Ptr' before 'Cnt' to synchronize with isr() */
        DSS_rxPtr = PIP_getWriterAddr(rxPipe);
        DSS_rxCnt = (sizeof(Int) / sizeof(MdInt)) * PIP_getWriterSize(rxPipe);

#if defined(_EDMA_) || defined(_DMA_)
        DSS_dmaRxStart(DSS_rxPtr, DSS_rxCnt);
#endif
    }
    else if (calledByISR && (DSS_error & DSS_RXERR) == 0) {
        if (DSS_config.enable) {
            TRC_disable(TRC_GBLTARG);
        }
        LOG_error("receive buffer overflow: %u", CLK_gethtime());
        DSS_error |= DSS_RXERR;
    }

    nested = 0;
}

#ifdef _EDMA_
/*
 *  ======== DSS_dmaInit ========
 *  Initialise EDMA Controller
 */
Void DSS_dmaInit(Void)
{
    LOG_message("DSS_dmaInit(): %u", CLK_gethtime());

    /* General EDMA Initialization */
    EDMA_RSET(EER, 0x0000);     /* Disable all events */
    EDMA_RSET(ECR, 0xffff);     /* Clear all pending events */
    EDMA_RSET(CIER, 0x0000);    /* Disable all events to Interrupt */
    EDMA_RSET(CIPR, 0xffff);    /* Clear all pending Queued EDMA ints */

    /* Enable Rx/Tx DMA Complete Interrupts to the CPU */
    EDMA_RSET(CIER, DSS_RXDONE | DSS_TXDONE);
}

/*
 *  ======= DSS_dmaRxStart ========
 */
Void DSS_dmaRxStart(Void *dst, Int nsamps)
{
    LOG_message("DSS_dmaRxstart(): %u", CLK_gethtime());

    CACHE_clean(CACHE_L2, dst, nsamps);

    /* Reconfig EDMA channel for next receive buffer */
    EDMA_RSETH(DSS_hEdmaRint0, CNT, (Uns) nsamps);
    EDMA_RSETH(DSS_hEdmaRint0, DST, (Uns) dst);

    EDMA_RSET(EER, 0x3000);     /* Enable McBSP0 Rx/Tx Events to the DMA */
}

/*
```

```
 *  ======= DSS_dmaTxStart ========
 */
Void DSS_dmaTxStart(Void *src, Int nsamps)
{
    static Int startup = 1;

    LOG_message("DSS_dmaTxstart(): %u", CLK_gethtime());

    CACHE_flush(CACHE_L2, src, nsamps);

    if (startup) {
        startup = 0;
        DSS_spWrite(0); DSS_spWrite(0);
        /* EDMA_RSET(ECR, 0x3000);    /Clear any previous McBSP Events */
    }

    /* Reconfig EDMA channel for next transmit buffer */
    EDMA_RSETH(DSS_hEdmaXint0, SRC, (Uns) src);
    EDMA_RSETH(DSS_hEdmaXint0, CNT, (Uns) nsamps);

    EDMA_RSET(EER, 0x3000);     /* Enable McBSP0 Rx/Tx Events to the DMA */
}
#endif

#ifdef _DMA_
/*
 *  ======= DSS_dmaRxStart ========
 */
void DSS_dmaRxStart(void *dst, Int nsamps)
{
    DMA_RSETH(DSS_hDmaRint0, SRC,(Int)(&DSS_hMcbsp0->baseAddr[_MCBSP_DRR_OFFSET]));
    DMA_RSETH(DSS_hDmaRint0, DST, (Uns)dst);
    DMA_RSETH(DSS_hDmaRint0, XFRCNT, (0x1 << 16) + nsamps / (sizeof(Int) / sizeof(MdInt)));
    DMA_start(DSS_hDmaRint0);
}

/*
 *  ======= DSS_dmaTxStart ========
 */
void DSS_dmaTxStart(void *src, Int nsamps)
{
    static Int startup = 1;

    if (startup) {
        startup = 0;
        DSS_spWrite(0); DSS_spWrite(0);
    }
    DMA_RSETH(DSS_hDmaXint0, SRC, (Uns)src);
    DMA_RSETH(DSS_hDmaXint0, DST,(Int)(&DSS_hMcbsp0->baseAddr[_MCBSP_DXR_OFFSET]));
    DMA_RSETH(DSS_hDmaXint0, XFRCNT, (0x1 << 16) + nsamps / (sizeof(Int) / sizeof(MdInt)));
    DMA_start(DSS_hDmaXint0);
}
#endif
```

```
/*   Do *not* directly modify this file.  It was    */
/*   generated by the Configuration Tool; any  */
/*   changes risk being overwritten.                */

/* INPUT audio.cdb */

/* MODULE PARAMETERS */
GBL_USERINITFXN = _FXN_F_nop;

MEM_MALLOCSEG = MEM_NULL;

CLK_TIMEFXN = CLK_F_getshtime;
CLK_HOOKFXN = HWI_F_dispatch;

PRD_THOOKFXN = FXN_F_nop;

RTDX_DATAMEMSEG = IDRAM;

HST_DSMBUFSEG = IDRAM;

SWI_EHOOKFXN = GBL_NULL;
SWI_IHOOKFXN = GBL_NULL;
SWI_EXECFXN = SWI_F_iexec;
SWI_RUNFXN = SWI_F_run;

TSK_STACKSEG = MEM_NULL;
TSK_CREATEFXN = _FXN_F_nop;
TSK_DELETEFXN = _FXN_F_nop;
TSK_EXITFXN = _FXN_F_nop;

IDL_CALIBRFXN = IDL_F_calibrate;

SYS_ABORTFXN = _error;
SYS_ERRORFXN = _error;
SYS_EXITFXN = _error;
SYS_PUTCFXN = _FXN_F_nop;

/* OBJECT ALIASES */
_IPRAM = IPRAM;
_SBSRAM = SBSRAM;
_SDRAM0 = SDRAM0;
_SDRAM1 = SDRAM1;
_IDRAM = IDRAM;
_PRD_clock = PRD_clock;
_RTA_fromHost = RTA_fromHost;
_RTA_toHost = RTA_toHost;
_HWI_RESET = HWI_RESET;
_HWI_NMI = HWI_NMI;
_HWI_RESERVED0 = HWI_RESERVED0;
_HWI_RESERVED1 = HWI_RESERVED1;
_HWI_INT4 = HWI_INT4;
_HWI_INT5 = HWI_INT5;
_HWI_INT6 = HWI_INT6;
_HWI_INT7 = HWI_INT7;
_HWI_INT8 = HWI_INT8;
_HWI_INT9 = HWI_INT9;
_HWI_INT10 = HWI_INT10;
_HWI_INT11 = HWI_INT11;
_HWI_INT12 = HWI_INT12;
_HWI_INT13 = HWI_INT13;
_HWI_INT14 = HWI_INT14;
_HWI_INT15 = HWI_INT15;
_audioSwi = audioSwi;
_LNK_dataPump = LNK_dataPump;
_RTA_dispatcher = RTA_dispatcher;
_IDL_cpuLoad = IDL_cpuLoad;
_IDL0 = IDL0;
_LOG_system = LOG_system;
_trace = trace;
_DSS_rxPipe = DSS_rxPipe;
_DSS_txPipe = DSS_txPipe;
_IDL_busyObj = IDL_busyObj;
_DSS_ioPhase = DSS_ioPhase;

/* MODULE GBL */

SECTIONS {
   .vers (COPY): {} /* version information */
```

```
}

-llnkrtdx.a62
-ldrivers.a62          /* device drivers support */
-lbiosi.a62            /* DSP/BIOS support */
-lrtdx.lib             /* RTDX support */
-lcsl6201.lib
-lrtsbios.a62          /* C and C++ run-time library support */

_GBL_CACHE = GBL_CACHE;

/* MODULE MEM */
-stack 0x800
MEMORY {
   IPRAM       : origin = 0x0,         len = 0x10000
   SBSRAM      : origin = 0x400000,    len = 0x40000
   SDRAM0      : origin = 0x2000000,   len = 0x400000
   SDRAM1      : origin = 0x3000000,   len = 0x400000
   IDRAM       : origin = 0x80000000,  len = 0x10000
}
/* MODULE CLK */
SECTIONS {
   .clk: {
        CLK_F_gethtime = CLK_F_getshtime;
        CLK_A_TABBEG = .;
        *(.clk)
        CLK_A_TABEND = .;
        CLK_A_TABLEN = (. - CLK_A_TABBEG) / 1;
   } > IDRAM
}
_CLK_PRD = CLK_PRD;
_CLK_COUNTSPMS = CLK_COUNTSPMS;
_CLK_REGS = CLK_REGS;
_CLK_USETIMER = CLK_USETIMER;
_CLK_TIMERNUM = CLK_TIMERNUM;
_CLK_TDDR = CLK_TDDR;

/* MODULE PRD */
SECTIONS {
   .prd: {
        PRD_A_TABBEG = .;
        /* no PRD objects */
        PRD_A_TABEND = .;
        PRD_A_TABLEN = (. - PRD_A_TABBEG) / 32;
   } > IDRAM
}

/* MODULE RTDX */
_RTDX_interrupt_mask = 0x0;

/* MODULE HWI */
SECTIONS {
    .hwi_vec: 0x0 {
        HWI_A_VECS = .;
        *(.hwi_vec)
    }
}

/* MODULE SWI */
SECTIONS {
   .swi: {
        SWI_A_TABBEG = .;
        *(.swi)
        SWI_A_TABEND = .;
        SWI_A_TABLEN = (. - SWI_A_TABBEG) / 44;
   } > IDRAM
}

/* MODULE TSK */
_KNL_swi = 0;
/* MODULE IDL */
SECTIONS {
   .idl: {
        IDL_A_TABBEG = .;
        *(.idl)
        IDL_A_TABEND = .;
        IDL_A_TABLEN = (. - IDL_A_TABBEG) / 8;
        IDL_A_CALBEG = .;
```

```
        *(.idlcal)
        IDL_A_CALEND = .;
        IDL_A_CALLEN = (. - IDL_A_CALBEG) / 8;
   } > IDRAM
}



SECTIONS {
        .bss:     {} > IDRAM

        .far:     {} > IDRAM

        .sysdata: {} > IDRAM

        .mem:     {} > IDRAM

        .gblinit:    {} > IDRAM

        .sysregs: {} > IDRAM

        .trcdata:    {} > IDRAM

        .args: fill=0 {
            *(.args)
            . += 0x4;
        } > IDRAM

        .hst: {
             HST_A_TABBEG = .;
            _HST_A_TABBEG = .;
            *(.hst)
            HST_A_TABEND = .;
            _HST_A_TABEND = .;
             HST_A_TABLEN = (. - _HST_A_TABBEG) / 20;
            _HST_A_TABLEN = (. - _HST_A_TABBEG) / 20;
        } > IDRAM

        .cinit:    {} > IDRAM

        .pinit:    {} > IDRAM

        .data:    {} > IDRAM

        .const:    {} > IDRAM

        .switch:    {} > IDRAM

        .cio:     {} > IDRAM

       /* RTA_toHost buffer */
       .hst0: align = 0x4 {} > IDRAM

        .sts: {
             STS_A_TABBEG = .;
            _STS_A_TABBEG = .;
            *(.sts)
            STS_A_TABEND = .;
            _STS_A_TABEND = .;
             STS_A_TABLEN = (. - _STS_A_TABBEG) / 16;
            _STS_A_TABLEN = (. - _STS_A_TABBEG) / 16;
        } > IDRAM

        .sys:     {} > IDRAM

        .stack: fill=0xc0ffee {
            GBL_stackbeg = .;
            *(.stack)
            GBL_stackend = GBL_stackbeg + 0x800 - 1;
            _HWI_STKBOTTOM = GBL_stackbeg + 0x800 - 4 & ~7;
            _HWI_STKTOP = GBL_stackbeg;
        } > IDRAM

        .rtdx_data: {}  > IDRAM

        .log: {
             LOG_A_TABBEG = .;
            _LOG_A_TABBEG = .;
```

```
        *(.log)
        LOG_A_TABEND = .;
        _LOG_A_TABEND = .;
         LOG_A_TABLEN = (. - _LOG_A_TABBEG) / 24;
        _LOG_A_TABLEN = (. - _LOG_A_TABBEG) / 24;
    } > IDRAM

    .dsm: {} > IDRAM

   /* RTA_fromHost buffer */
   .hst1: align = 0x4 {} > IDRAM

    /* DSS_txPipe buffer */
    .pip1: align = 0x80 {} > IDRAM

    .pip: {
         PIP_A_TABBEG = .;
        _PIP_A_TABBEG = .;
        *(.pip)
        PIP_A_TABEND = .;
        _PIP_A_TABEND = .;
         PIP_A_TABLEN = (. - _PIP_A_TABBEG) / 100;
        _PIP_A_TABLEN = (. - _PIP_A_TABBEG) / 100;
    } > IDRAM

    /* DSS_rxPipe buffer */
    .pip0: {} > IDRAM

    .printf (COPY): {} > IDRAM

    /* LOG_system buffer */
    .LOG_system$buf: align = 0x800 fill = 0xffffffff {} > IDRAM

    /* trace buffer */
    .trace$buf: align = 0x80 fill = 0xffffffff {} > IDRAM

    frt:    {} > IPRAM

    .sysinit:    {} > IPRAM

    .rtdx_text: {}  > IPRAM

    .text:    {} > IPRAM

    .hwi: {}  > IPRAM

    .bios:    {} > IPRAM

}
```