FPGA-based MP3 Player

Project Report

Student: Lalu Luka

Advisors:
Dr. Yufeng Lu
Dr. In Soo Ahn

May 10, 2010

Abstract:
Digital design using a Field Programmable Gate Array (FPGA) device is a rapidly evolving field. The increasing density and capacity of these devices make it possible to implement an entire embedded system on a single chip. The goal of the project is to design an MPEG Layer III (MP3) player using a Xilinx Virtex 5 FPGA board. The system will read an MP3 file from a compact flash memory, decode the MP3 bit stream into 16-bit pulse code modulated (PCM) outputs using a standard MP3 decoding algorithm, and play the output through an external speaker. C programming language is used to run on a Microblaze 32-bit processor. Hardware description language such as VHDL is used to drive external peripherals, including the stereo AC97 codec and LCD controller. The AC97 codec converts the digital PCM outputs into an analog sound wave, and the LCD controller displays the title and author information of the selected song. The software and hardware designs are integrated on the Xilinx Embedded Development Kit platform. In this project, data compression techniques are used in MP3 encoding and decoding are explored and tested on hardware.

Table of Contents

Motivation

I can honestly say that I have always had a passion for understanding and building MP3 players. I hope to continue this interest by pursuing a career in an audio engineering industry where similar data compression techniques, as well as hardware-software co-designs, may be used.

Introduction

Digital design using a Field Programmable Gate Array (FPGA) is a rapidly evolving field. A complete embedded system can be built and programmed into a single FPGA chip for digital signal processing applications.

The goals of this project are to (1) gain an in-depth understanding of hardware/software co-design using an FPGA, (2) understand the specifications set by the ISO/IEC 11172-3 standard for encoding and decoding MP3 files., and (3) build a FPGA-based MPEG Layer III (MP3) system, which implements the MP3 decoding algorithm using VHDL and C language on the Embedded Development Kit (EDK) software platform.

System Description

The inputs to the FPGA MP3 player system will be an MP3 bit stream that is preloaded onto a compact flash memory (CFM) and any user interface control input. Using pushbuttons, the user will be enabled scan through the MP3 file list, and then select, play, pause, and/or stop the song. In addition, volume control is triggered by a change in the on-board rotary encoder dial position.

The outputs of the MP3 decoder will be the LCD screen that is used to display MP3 related information (i.e. song title and author) and the reconstructed MP3 audio in digital format, that is, 16-bit pulse code modulated (PCM) outputs and play the audio files through an external speaker. The PCM outputs need to be converted to analog format via the on-board stereo ac97 codec hardware chip before the audio can be heard with an external speaker that can be attached through the audio jack with a 15 mW amplifier. The high level system block diagram is illustrated in Figure 4-1.

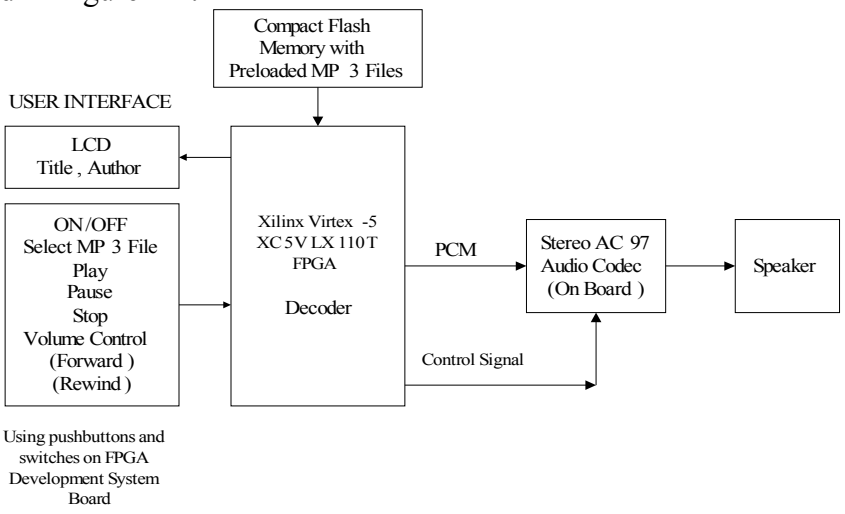


Figure 4-1 High Level System Block Diagram

The system includes:

- **MP3 Decoder**
  A MP3 decoder runs on the Xilinx Virtex-5 XC5V LX110T FPGA that will decode the selected MP3 stream with the sampling frequency specified in the MP3 header. A typical sampling frequency is 44.1 kHz. The software and hardware designs are integrated on the Xilinx Embedded Development Kit platform with C programming language used to run on the Xilinx software-based Microblaze 32-bit processor.

  External Peripherals:
  VHDL is used to drive all external peripherals. Most applications utilize devices by means of high-level device-generic commands. Driver software accept these generic high-level commands and break them into a series of low-level device-specific commands. The peripherals used in the FPGA MP3 player system are the pushbuttons, LCD screen, rotary encoder, compact flash memory, stereo ac97 codec, and the external 512 MB DDRAM

- **User Interface**
  The user interface provides the inputs to control the MP3 player, such as scanning, selecting, playing, pausing, and stopping the MP3 files. It will also allow outputting related information (i.e., title and author of the song) on the LCD. The LCD is capable of displaying 2 lines of 16 characters.

- **Compact Flash Memory Card**
  The 1 Gb capacity ML50X FAT 16 compact flash memory (CFM) card supplies the preloaded MP3 files for the MP3 decoder system in the FPGA. MP3 files are loaded onto to the CFM using a PC and memory card reader.

- **Onboard Stereo Audio AC97 Codec**
  The AC97 codec (AD1981B) is used to convert the PCM format signal from the MP3 decoder into an audio signal, which is fed into an external speaker through an audio jack.

Equipment list:
Table 5-1. Equipment list

Item	Description
Xilinx UPV5-LX110T FPGA	FPGA used to implement decoding process
AD1981B JSTZ	Onboard AC97 Stereo Codec Chip used to convert decoder digital PCM outputs to analog audio sound waves.
1 GB Compact Flash Memory (CFM) and an external memory card reader	The CFM holds preloaded MP3 files to be accessed and decoded by FPGA system. Eventually decoding process software will be run from the CFM, so that an external PC is not required.
Headphones or external speaker	Headphones will be connected through the AC97 audio jack (driven by the audio codec's internal 50-mW amplifier).

System functional requirements and performance specifications:

- **Input MP3 bit stream requirements**
  The MP3 player will decode MP3 inputs with various bit rates (from 128 kbps to 320 kbps) and different sampling frequencies (32 kHz, 44.1 kHz or 48 kHz)

- **Decoding speed**
  The ultimate objective of decoding speed is to process MP3 files in real-time. The execution time of the MP3 decoding will be profiled and measured. If the real-time specification can not be met, further optimization will be needed.

- **LCD display**
  The LCD will list the MP3 information in real-time.

Background information on the MP3 format

The need to reduce the size of audio files without any noticeable quality loss was stated in the 1980ies by the International Organization for Standardization (ISO). A working group within the ISO known as the Moving Pictures Experts Group (MPEG), developed a standard that contained several techniques for both audio and video compression. The audio part of the standard included three modes with increasing complexity and performance, as shown in Figure 6-1. The third mode, called Layer III, manages to compress music by a factor of 12 with almost no audible degradation. This technique is known as MP3 and has become very popular and widely used in applications today. The specifications of MP3 encoding and decoding can be found in the ISO standard document 11172-3 [5].

Complexity ↓

Coding	Ratio	Required bitrate
PCM CD Quality	1:1	1.4 Mbps
Layer I	4:1	384 kbps
Layer II	8:1	192 kbps
Layer III (MP3)	12:1	128 kbps

Figure 6-1. Comparison of audio compression layers

Overview of MP3 Encoding Process:
MP3 encoding involves representing a song as a bit stream (an array of 0's and 1's) that can be recovered by a MP3 decoder (player). The high percentage compression involved in MP3 encoding allows songs to be stored and shared rather easily and quickly on computers and through the internet without losing any perceptible quality. This lossy compression works by first masking inaudible frequency components to the human ear, and then using several data compression techniques that remove data redundancies.

First the analog audio is sampled at a specific sampling rate, typically at 44.1 kHz. This is due to Nyquist frequency's rule, in which the sampling rate must be at least two times greater than the largest possible frequency component present in the data. And since the range of audible frequencies to the human ear is roughly 20 Hz to 22 kHz, this sampling rate is usually chosen. The signals are quantized using pulse code modulation, where each sample amplitude is represented by 16 bits.

To remove redundancies and compress data, frequency analysis techniques are used. The PCM samples are filtered for 32 equal frequency spectrums, called subbands using a polyphase architecture that yields in a higher computational efficiency. A discrete cosine transformation is then applied to remove low energy signals from high frequency components.

Further compression is achieved by using a lossless compression technique known as Huffman encoding that is based on statistical behavior of data. Finally, the bit stream is arranged into frames that the MP3 decoder will analyze to reconstruct the MP3 sound.

MP3 decoding

MP3 decoding is the reverse process of MP3 encoding. Fortunately, decoding is not nearly as complex, since it does not require a psychoacoustic model (a virtual model of the human ear and how it perceives different frequencies). The MP3 decoder's role is to recover the original audio by analyzing certain sections of a frame to gain information about encoding parameters used and then use reverse procedures to reconstruct PCM samples.

Each frame consists of exactly 1152 PCM samples and contains at least two sections:
1) A header section that contains important encoding parameters, such as bit rate and sampling frequency used and an audio data section that holds the encoded bit stream. Some MP3 bit streams contain an optional ID3 tag frame that can be used to store MP3 related information including the title and author of the song.

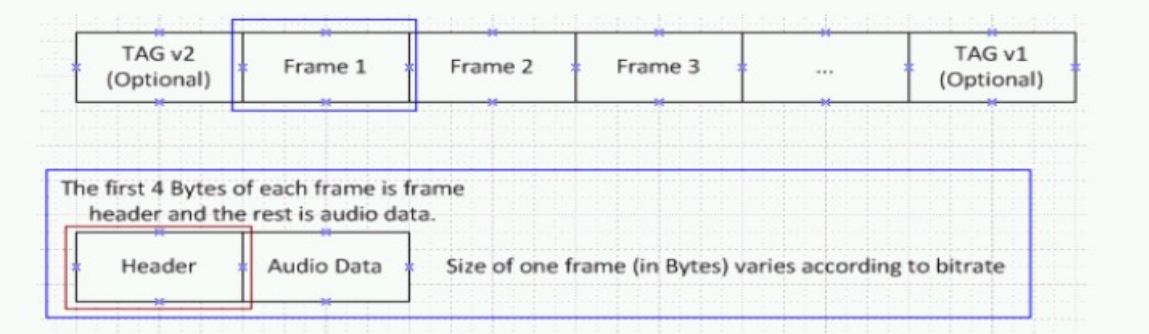


Figure 7-1. MP3 File Structure

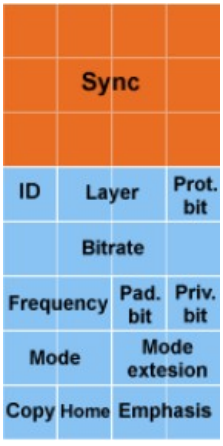


Figure 7-2. 32 byte header section

The frame size in bytes varies from song to song, and in some cases, even within one song (when using variable bit rates). The general equation for calculating the frame size in bytes is found in Equation 8-1.

Equation 8-1
Frame size (in bytes) = (144* bit rate)/ (sampling rate + padding)
Where 144= (1152 PCM/frame) / (8 bits/byte) and where padding is an integer number to ensure that the frame size is an integer number

Bit rate is the rate at which the compressed bit stream is delivered from the storage medium to the input of a decoder while sampling frequency defines the number of samples per second taken from a continuous signal to make a discrete signal
.
For MP3 encoding, there are several allowed bit rates and sampling frequencies that can be used, as illustrated in Table 8-1 and 8-2, respectively. These tables are copied directly from the ISO standard document [5].

Table 8-1 [5]

bitrate_index	bitrate specified (kbits/s)		
	Layer I	Layer II	Layer III
'0000'	free	free	free
'0001'	32	32	32
'0010'	64	48	40
'0011'	96	56	48
'0100'	128	64	56
'0101'	160	80	64
'0110'	192	96	80
'0111'	224	112	96
'1000'	256	128	112
'1001'	288	160	128
'1010'	320	192	160
'1011'	352	224	192
'1100'	384	256	224
'1101'	416	320	256
'1110'	448	384	320
'1111'	forbidden	forbidden	forbidden

Table 8-2 [5]

sampling_frequency	frequency specified (kHz)
'00'	44,1
'01'	48
'10'	32
'11'	reserved

Typically, a sampling rate of 44.1 kHz is used and is known as “CD quality” while 48 kHz is referred to as “DVD quality.”

The MP3 decoding process is shown in Figure 9-1. It includes the following stages: initial reading, Huffman decoding, re-quantization and reordering, stereo decoding, alias reduction, inverse modified discrete cosine transform (IMDCT), and synthesis polyphase filter bank [1-7].

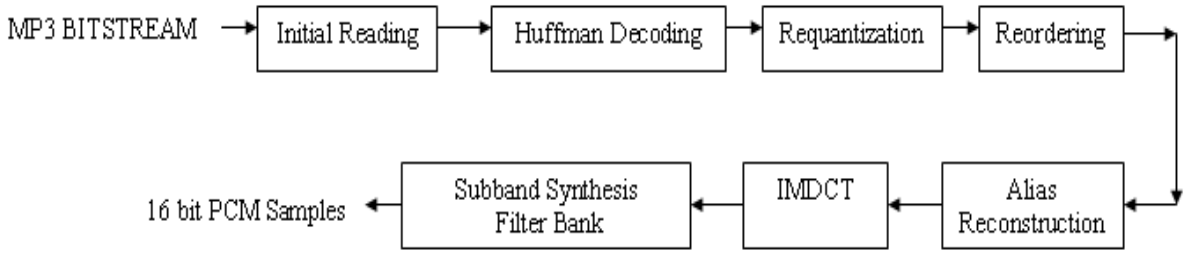


Figure 9-1. High Level Diagram of Decoding Process

1. Initial reading

The incoming data stream is split up into individual frames. The header section of each frame is analyzed to obtain parameters used in the encoding process (i.e. bit rate and sampling frequency). The first action is the synchronization of the decoder to the incoming bit stream by checking if the first 12 bits of the header section are 1's. Scalefactors and Huffman table selection bits are also decoded.

2. Huffman decoding.

The Huffman algorithm is used for lossless data compression. The basic idea of the technique is to assign shorter binary codes to more frequent samples and longer codes to less frequent samples. The Huffman decoding procedure is based on tables that are used to map the Huffman binary codes to the original samples.

3. Re-quantization

During the encoding process, the outputs of the MDCT, or frequency domain samples, were pre-quantized in an attempt to use more precision when needed. It turns out that finer frequency resolution is needed for low volume sounds and larger values are coded with less accuracy. Afterwards, the values were scaled, or multiplied by a scalefactor, a value that is based on the absolute threshold of the human ear (a frequency dependent function). Larger scalefactors are needed if the frequency components are more difficult to hear. So for the decoding process, the values need to be requantized. Afterwards, de-scaling is required

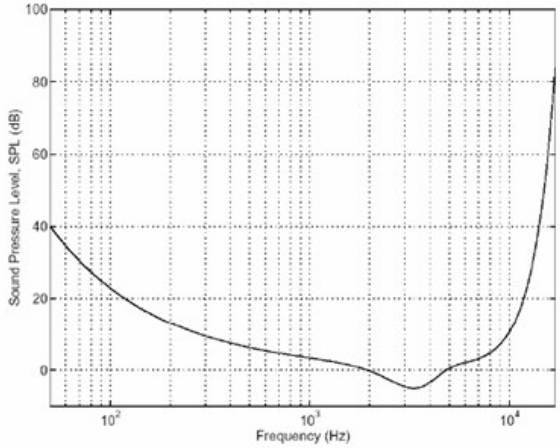


Figure 9-2. Absolute threshold of the human ear [10]

4. **Re-ordering**

In the MP3 encoding process, the use of short windows would generate frequency lines ordered first by subband, then by window and at last by frequency. In order to increase the efficiency of the Huffman coding the frequency lines for the short windows case were reordered into subbands first, then frequency and at last by window, since the samples close in frequency are more likely to have similar values. The reordering block in the MP3 decoding process will re-sort the samples by subbands, then on windows and then on increasing frequency. For a description on windowing, refer to the IMDCT block.

5. Alias reduction block

Aliasing is the overlap of frequency components when energies greater than Nyquist frequency are present. This is the result of the decimation or the reduction of sampling rate in the analysis filter bank process where overlapping of adjacent subband filters is inevitable. In the encoding process, these aliasing effects are removed to reduce the amount of information that needs to be transmitted. This can be achieved by using a series of butterfly computations that add weighted, mirrored versions of adjacent subbands to each other. In the decoding process, aliasing artifacts must be added to the signal again in order to obtain a correct reconstruction of the audio signal. The alias reconstruction calculation consists of eight butterfly calculations for each subband.

6. Inverse Modified Discrete Cosine Transform (IMDCT)

The *Inverse Modified Discrete Cosine Transform* (IMDCT) is the inverse of the modified discrete cosine transform used in MP3 encoding. The MDCT was used to represents signals as a sum of cosine waves, essentially transforming them to the frequency domain. Compared to the DFT and other well-known transforms, the MDCT has a few properties that make it very suitable for audio compression. First of all, the MDCT has the *energy compaction property* common to discrete cosine transforms. This means most of the information in the signal is concentrated to a few output samples with high energy. The term modified is used since there is a 50% overlap. The lower 18 values are added with the higher 18 values from the previous frame, and used as output. The higher 18 values are then stored and used the same way when the next frame is being decoded. This overlapping that avoids sharp discontinuities.

The formal definition of the IMDCT is shown in Equation 10-1, where n=36

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos\left[\frac{\pi}{2n}\left[2i+1+\frac{n}{2}\right](2k+1)\right], \text{ for i=0 to n-1}$$

Equation 10-1. Definition of IMDCT

Windowing

To smoothen frame transition, a 36 point windowing function is applied to the IMDCT outputs. Depending on the degree of stationarity of consecutive frames, two different types of windows can be used. If there is little difference between consecutive (previous and present) frames, then a long window for 36 values is used to enhance the spectral (frequency) resolution. Alternatively, if there is a considerable difference from the previous time frame, then three short overlapped windows are applied. Shorter windows are used for better time resolution, when is there a quick change in pitch or instruments used. The functions for the

long and short windows can be found in the ISO standard document [5] on page 43 and are illustrated in Figure 11-1 and 11-2, respectively.

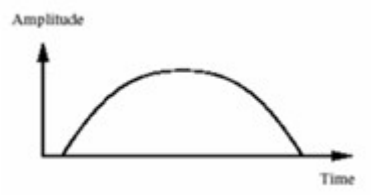


Figure 11-1. Illustration of long window

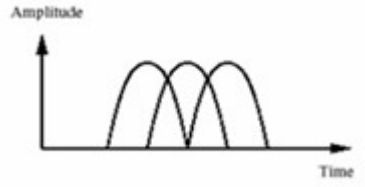


Figure 11-2. Illustration of three short overlapped windows

Frequency Inversion

To achieve the correct phase difference, the encoder compensates the 32 band polyphase filter for its frequency inversion. In decoder, every odd sample of every odd subband is multiplied by (-1).

7. Synthesis polyphase filter bank

The synthesis polyphase filter bank is the final step in the decoding process. It is used to combine the signal energies from all the 32 subbands. The result output for each frame is 1152 16 bit PCM samples. A polyphase architecture is used since the decimation of the sampling rate allows the use of a lower number of filter coefficients, and thus improves computational efficiency. The method recommended by ISO standard for transforming subband samples to the Pulse code modulated format involves shifting, matrixing with a 32 point discrete cosine transform that represent band pass filter coefficients, a 512 point window to improve filter quality, and finally a summation for all the subbands. Various algorithms can be implemented that observe symmetry properties and reduce the number of computations. For example, the DCT function can be calculated using a method known as the fast DCT in a similar manner that a DFT function can be more efficiently computed using the FFT.
The method recommended by the ISO standard document [5] for transforming the subband samples into the PCM format is illustrated in Figure 11-3.

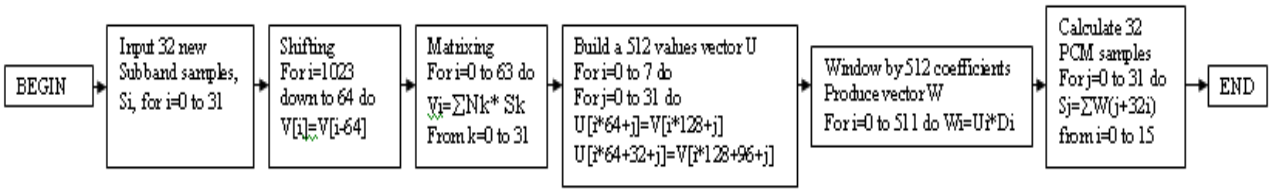


Figure 11-3. Subband synthesis implementation

One by one, the PCM samples are finally sent to the stereo ac97 FIFO register that enables a playback of each frame of the song.

Stereo AC97 Codec

The stereo AC97 codec hardware chip for the Xilinx Virtex 5 XUPV5 development board is the AD1981B. As shown in the functional block diagram of Figure 12-1, the codec has both DAC and ADC capabilities. For this project, only the 20 bit DAC will be used, although only 16 bits will be sent.

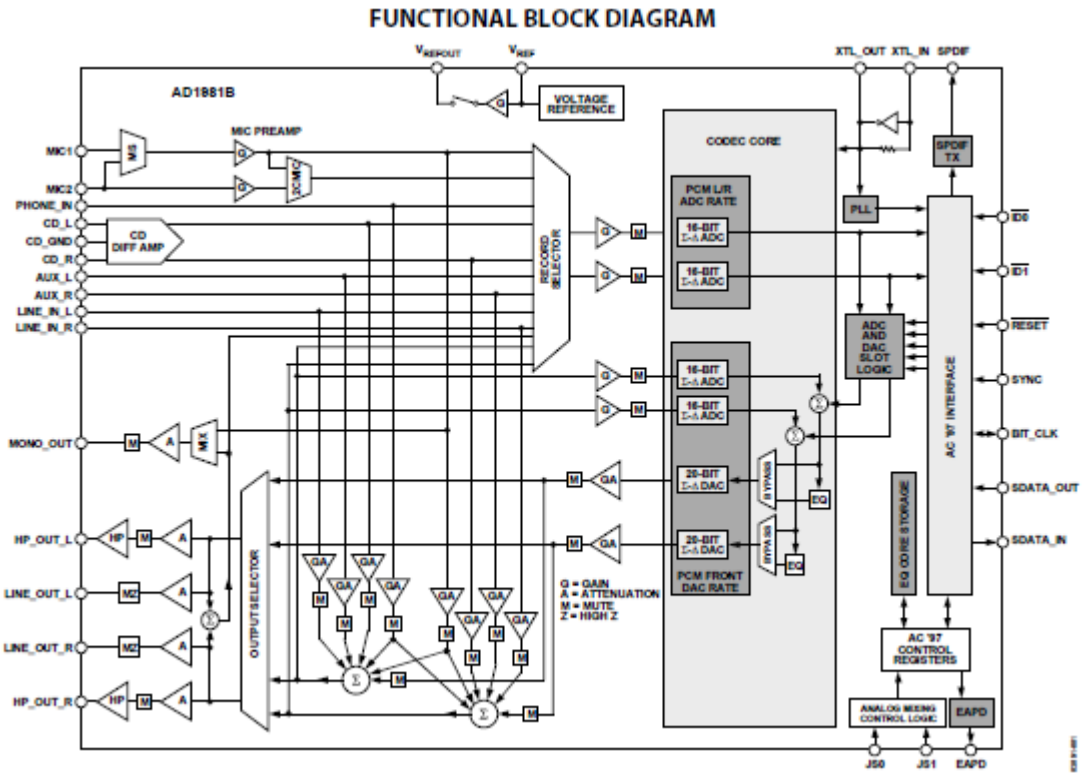


Figure 12-1. Functional block diagram of AD1981B (stereo AC97 codec)

As an external peripheral, the stereo ac97 codec was driven by .vhdl files that can be found in the VHDL folder of the provided data disc. As described in the debugging procedures section of this report, modifying the depth of the In_FIFO register (used for playback) did not affect the sound quality of the output audio. As of now, the depth is set to the default value of 16, implying that 16 PCM samples can be loaded onto the FIFO before being replaced by a new set of PCM samples.

In the Mpeg_Audio_decoder function (called within play_song function), the synthesized PCM samples are sent to the stereo ac97 codec FIFO, which enables the playback of each frame. This is done by XAC97_WriteFifo (XPAR_AUDIO_CODEC_BASEADDR, SampleLR);
where SampleLR is a PCM sample and XPAR_AUDIO_CODEC_BASEADDR is the defined beginning memory location for the opb_ac97_controller_ref_0 peripheral on EDK.

The XAC97_WriteFIFO is defined in the Xac97_l.c file, as shown in this report below as well as on the provided data disc. Other functions including the initial setup of the codec and volume control operation are also found in this file. It is important to note that register definitions had to be modified to work for the Xilinx Virtex 5's ac97 codec chip, the AD1981B. Failing to do so results in no audio. For more details on hardware electrical characteristics, as well as register definitions and pin locations, please consult the Stereo_AC97_Codec datasheet in the "Datasheets" folder of the provided data disc.

Xac97_l.c

//including comments, 280 lines of code

```
#include <stdio.h>
```

```
#include <math.h>

//Lalu Luka removed xac97_l.h and placed contents in this .c file

#include "xio.h"
#include "sleep.h"
#include "xparameters.h"
#include <xbasic_types.h>

int c;

#define MY_AC97_BASEADDR XPAR_OPB_AC97_CONTROLLER_REF_0_BASEADDR

//The following register definitions are exclusive to the Virtex 5 XUPV5's AC97 codec (AD…)
//Virtex 2 will have different constant offsets
#define AC97_InFIFO             MY_AC97_BASEADDR
#define AC97_OutFIFO            MY_AC97_BASEADDR + 0x4
#define AC97_FIFO_Status        MY_AC97_BASEADDR + 0x8
#define AC97_Control            MY_AC97_BASEADDR + 0xC
#define AC97_RegAddr            MY_AC97_BASEADDR + 0x10
#define AC97_RegRead            MY_AC97_BASEADDR + 0x14
#define AC97_RegWrite           MY_AC97_BASEADDR + 0x18

#define AC97_IN_FIFO_OFFSET         0x0
#define AC97_STATUS_OFFSET          0x8

#define AC97_InFIFO_Full        0x01
#define AC97_InFIFO_Half_Full   0x02
#define AC97_OutFIFO_Full       0x04
#define AC97_OutFIFO_Empty      0x08
#define AC97_Reg_Access_Finished 0x10
#define AC97_CODEC_RDY          0x20
#define AC97_REG_ACCESS         0x40
#define AC97_Enable_In_Intr     0x01


// AC97 CODEC Registers
#define AC97_Reset              0x00
#define AC97_MasterVol          0x02
#define AC97_HeadphoneVol       0x04
#define AC97_MasterVolMono      0x06
#define AC97_Reserved0x08       0x08
#define AC97_PCBeepVol          0x0A
#define AC97_PhoneInVol         0x0C
#define AC97_MicVol             0x0E
#define AC97_LineInVol          0x10
#define AC97_CDVol              0x12
#define AC97_VideoVol           0x14
#define AC97_AuxOutVol          0x16
#define AC97_PCMOutVol          0x18
#define AC97_RecordSelect       0x1A
#define AC97_RecordGain         0x1C
#define AC97_Reserved0x1E       0x1E
#define AC97_GeneralPurpose     0x20
#define AC97_3DControl          0x22
#define AC97_PowerDown          0x26
#define AC97_ExtendedAudioID    0x28
```

```
#define AC97_ExtendedAudioStat  0x2A
#define AC97_PCM_DAC_Rate       0x2C //from xac97_l.h
#define AC97_PCM_ADC_Rate       0x32 //from xac97_l.h
#define AC97_PCM_DAC_Rate0      0x78
#define AC97_PCM_DAC_Rate1      0x7A
#define AC97_Reserved0x34       0x34
#define AC97_JackSense          0x72
#define AC97_SerialConfig       0x74
#define AC97_MiscControlBits    0x76
#define AC97_VendorID1          0x7C
#define AC97_VendorID2          0x7E

// Volume Constants
#define AC97_VolMute     0x8000
#define AC97_VOL_MIN      0x1f1f //0x3F3F
#define AC97_VOL_MID    0x1010// 0x0a0a //0x1010
#define AC97_VOL_MAX     0x0000

// Macros for reading/writing AC97 core registers
#define XAC97_mGetRegister (BaseAddress, offset)  XIo_In32((BaseAddress + offset))

#define XAC97_mSetInFifoData (BaseAddress, value) XIo_Out32 ((BaseAddress) +
AC97_IN_FIFO_OFFSET, (value))

#define XAC97_mGetOutFifoData (BaseAddress) XIo_In32 ((BaseAddress + AC97_OUT_FIFO_OFFSET))

#define XAC97_mGetStatus (BaseAddress) XIo_In32 ((BaseAddress + AC97_STATUS_OFFSET))

#define XAC97_mSetControl (BaseAddress, value) XIo_Out32 ((BaseAddress) + AC97_CONTROL_OFFSET,
(value))

#define XAC97_mSetAC97RegisterAccessCommand (BaseAddress, value) \
XIo_Out32 ((BaseAddress) + AC97_REG_CONTROL_OFFSET, (value))

#define XAC97_mGetAC97RegisterData (BaseAddress) XIo_In32 ((BaseAddress +
AC97_REG_READ_OFFSET))

#define XAC97_mSetAC97RegisterData (BaseAddress, value) XIo_Out32 ((BaseAddress) +
AC97_REG_WRITE_OFFSET, (value))

// Status register macros
#define XAC97_isInFIFOFull (BaseAddress) (XAC97_mGetStatus (BaseAddress) & AC97_IN_FIFO_FULL)

#define XAC97_isInFIFOEmpty (BaseAddress) (XAC97_mGetStatus (BaseAddress) &
AC97_IN_FIFO_EMPTY)

#define XAC97_isOutFIFOEmpty (BaseAddress) (XAC97_mGetStatus (BaseAddress) &
AC97_OUT_FIFO_EMPTY)

#define XAC97_isOutFIFOFull (BaseAddress) (XAC97_mGetStatus (BaseAddress) &
AC97_OUT_FIFO_FULL)

#define XAC97_isRegisterAccessFinished (BaseAddress) \
        ((XAC97_mGetStatus (BaseAddress) & AC97_REG_ACCESS_BUSY) == 0)
 //         (XAC97_mGetStatus (BaseAddress) & AC97_REG_ACCESS_FINISHED))

#define XAC97_isRegisterAccessError (BaseAddress) \
```

```
        ((XAC97_mGetStatus (BaseAddress) & AC97_REG_ACCESS_ERROR) > 0)

#define XAC97_isCodecReady (BaseAddress) \
        (XAC97_mGetStatus (BaseAddress) & AC97_CODEC_RDY)

#define XAC97_isInFIFOUnderrun (BaseAddress) (XAC97_mGetStatus (BaseAddress) &
AC97_IN_FIFO_UNDERRUN)

#define XAC97_isOutFIFOOverrun (BaseAddress)   (XAC97_mGetStatus (BaseAddress) &
AC97_OUT_FIFO_UNDERRUN)

#define XAC97_getInFIFOLevel (BaseAddress) \
    ((XAC97_mGetStatus (BaseAddress) & AC97_IN_FIFO_LEVEL) >> AC97_IN_FIFO_LEVEL_RSHFT)


void XAC97_Delay (Xuint32 value) {
  while (value-- > 0);
}

#define AC97_CLEAR_IN_FIFO             0x1
#define AC97_CLEAR_OUT_FIFO            0x2
#define AC97_ENABLE_IN_FIFO_INTERRUPT   0x4
#define AC97_ENABLE_OUT_FIFO_INTERRUPT 0x8
#define AC97_ENABLE_RESET_AC97         0x10
#define AC97_DISABLE_RESET_AC97        0x0
#define AC97_CLEAR_FIFOS AC97_CLEAR_IN_FIFO | AC97_CLEAR_OUT_FIFO

//#define XAC97_mSetControl (BaseAddress, value) \
//          XIo_Out32 ((BaseAddress) + AC97_CONTROL_OFFSET, (value))
//
#define AC97_CONTROL_OFFSET      0xC             //works if C, E or F...does not otherwise


void XAC97_ClearFifos (Xuint32 BaseAddress)
{
            Xuint32 i;
            XAC97_mSetControl (BaseAddress, AC97_CLEAR_FIFOS);
            for (i = 0; i < 512; i++)
              XAC97_mSetInFifoData (BaseAddress, 0);
}


void WriteAC97Reg (int reg_addr, int value)
{
            XIo_Out32 (AC97_RegWrite, value);
            XIo_Out32 (AC97_RegAddr, reg_addr);
            //while ((XIo_In32 (AC97_FIFO_Status) & AC97_Reg_Access_Finished) == 0);
            usleep (10);
}

int ReadAC97Reg (int reg_addr
(
            XIo_Out32 (AC97_RegAddr, reg_addr | 0x80);
           // while ((XIo_In32 (AC97_FIFO_Status) & AC97_Reg_Access_Finished) == 0);
            usleep (10);
            return XIo_In32 (AC97_RegRead);   }
```

```
#define AC97_IN_FIFO_FULL        0x01

void XAC97_WriteFifo (Xuint32 BaseAddress, Xuint32 sample)
            {
             while (XAC97_isInFIFOFull (BaseAddress));
             XAC97_mSetInFifoData (BaseAddress, sample);
            }

int SetupAC97 (int samplerate) {

//-------------------------------------------
   WriteAC97Reg (AC97_Reset, 0);
   while (! (XIo_In32 (AC97_FIFO_Status) & AC97_CODEC_RDY)) {};
   XAC97_ClearFifos (MY_AC97_BASEADDR); /** Clear FIFOs **/
 //-----------------------------------------------------

  //xil_printf("-- Set DAC rate to %d Hz \r\n",samplerate);
   WriteAC97Reg (AC97_PCM_DAC_Rate, samplerate);
   WriteAC97Reg (AC97_PCM_DAC_Rate0, samplerate);

   // xil_printf ("-- Volume settings initialized \r\n");
  //need these settings, especially PCMoutVol...otherwise no audio heard
 //-------------------------------------------------------------------------------------
            WriteAC97Reg (AC97_MasterVol,     AC97_VOL_MAX);
            WriteAC97Reg (AC97_HeadphoneVol, AC97_VOL_MAX);
            WriteAC97Reg (AC97_MasterVolMono, AC97_VOL_MAX);
            WriteAC97Reg (AC97_PCBeepVol,     AC97_VolMute);
            WriteAC97Reg (AC97_PhoneInVol,    AC97_VolMute);
            WriteAC97Reg (AC97_CDVol,         AC97_VolMute);
            WriteAC97Reg (AC97_VideoVol,      AC97_VolMute);
            WriteAC97Reg (AC97_AuxOutVol,     AC97_VolMute);
            WriteAC97Reg (AC97_PCMOutVol,     AC97_VOL_MAX);
            WriteAC97Reg (AC97_RecordGain, AC97_VolMute); //added
            WriteAC97Reg (AC97_PowerDown,   0x0100); //added
            WriteAC97Reg (AC97_LineInVol,   AC97_VolMute); //added
//-------------------------------------------------------------------------------------

}

//these variables must be initialized to zero outside volume_control function
//otherwise, incorrect results

int q=0;
Xuint32 VI_old=0;

void volume_control (int VI)
            {

//initial method for checking encoder position and updating volume register value
//works, but only 4 different settings AND encoder values (VI) are not predictable
//one would think that values would increase by some constant for every increment in encoder position
//However, VI values are very random for the 360 degree rotation
//Majority of positions are VI==0
//      if (VI==1) {WriteAC97Reg (AC97_HeadphoneVol, AC97_VOL_MAX) ;}
//      if (VI==2) {WriteAC97Reg (AC97_HeadphoneVol, AC97_VOL_MIN) ;}
//      if (VI==3) {WriteAC97Reg (AC97_HeadphoneVol, AC97_VOL_MID) ;}
//      if (VI==4) {WriteAC97Reg (AC97_HeadphoneVol, AC97_VolMute) ;}
```

```
int volume_setting;

//xil_printf("VI:%d \n\r", VI);


if(VI!=VI_old)
{
  q++;
        volume_setting=8000 - (1000*q);
        //1000 is an arbitrary number, but very good choice for noticeable difference in volume
        //xil_printf ("vol: %s\n\r", volume_setting); //--for debugging purposes
        if (volume_setting<1) {q=0 ;}

        WriteAC97Reg (AC97_HeadphoneVol, volume_setting);
}
VI_old=VI;
//xil_printf ("VI_old:%d \n\r", VI_old); //--for debugging purposes


}
```

Cache

Cache is a block of local memory used to transparently store data so that future requests (from the DDRAM) for that data can be served faster. For the finalized project, the size of cache was set to about 128 kB. As also explained in the Observations and debugging procedures section of this report, modifying the size of the cache did not have a noticeable effect on the speed of the MP3 decoding. However, a thorough comparison of execution times for different sizes of cache was not conducted. Nevertheless, it was found from profiling results that there was an approximate 300% improvement in MP3 decoding speed (performance) with cache enabled compared to the program without using cache.

Making sure that the "xcache_l.h" file is included, the cache can be enabled and disabled using the functions as stated below.

```
void enable_cache()
{
 #define ICACHE_SIZE 16384  //* ICACHE_SIZE should be set to C_CACHE_BYTE_SIZE
 #define DCACHE_SIZE 16384  //* DCACHE_SIZE should be set to C_DCACHE_BYTE_SIZE

/* Initialize ICache */
microblaze_disable_icache ();
microblaze_init_icache_range(0, ICACHE_SIZE);
microblaze_enable_icache ();

/* Initialize DCache */
microblaze_disable_dcache ();
microblaze_init_dcache_range(0, DCACHE_SIZE);
microblaze_enable_dcache ();
}
```

```
void disable_cache()
{

        #if XPAR_MICROBLAZE_0_USE_DCACHE
    microblaze_disable_dcache();
    microblaze_invalidate_dcache();
  #endif

  #if XPAR_MICROBLAZE_0_USE_ICACHE
    microblaze_disable_icache();
    microblaze_invalidate_icache();
  #endif
}
```

Flowcharts

The MP3 decoding algorithm described thus far is implemented completed in software using C language. High level flowcharts for the main program, as well as the select_song and play_song functions are illustrated below.

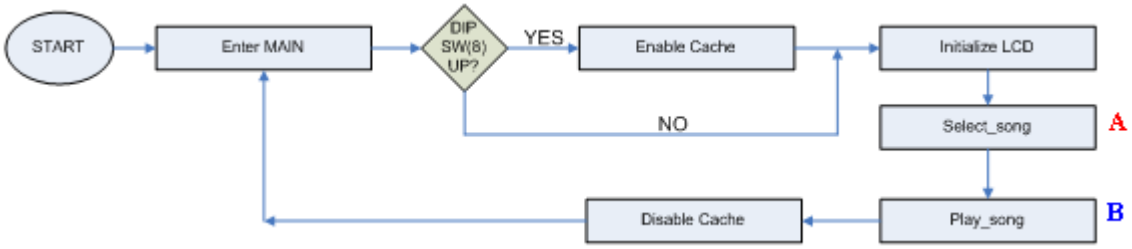


Figure 18-1. High level flowchart for FPGA MP3 player program

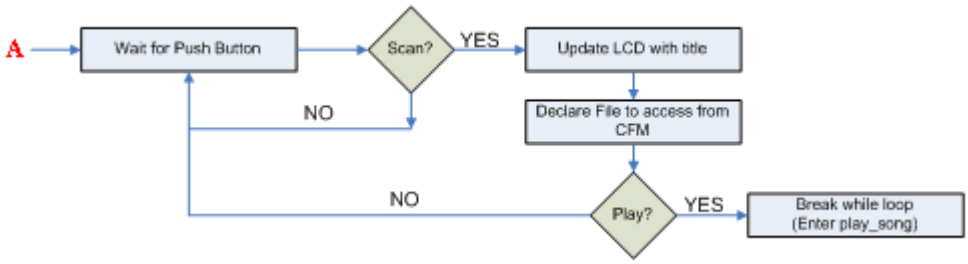


Figure 18-2. Flowchart for select_song function

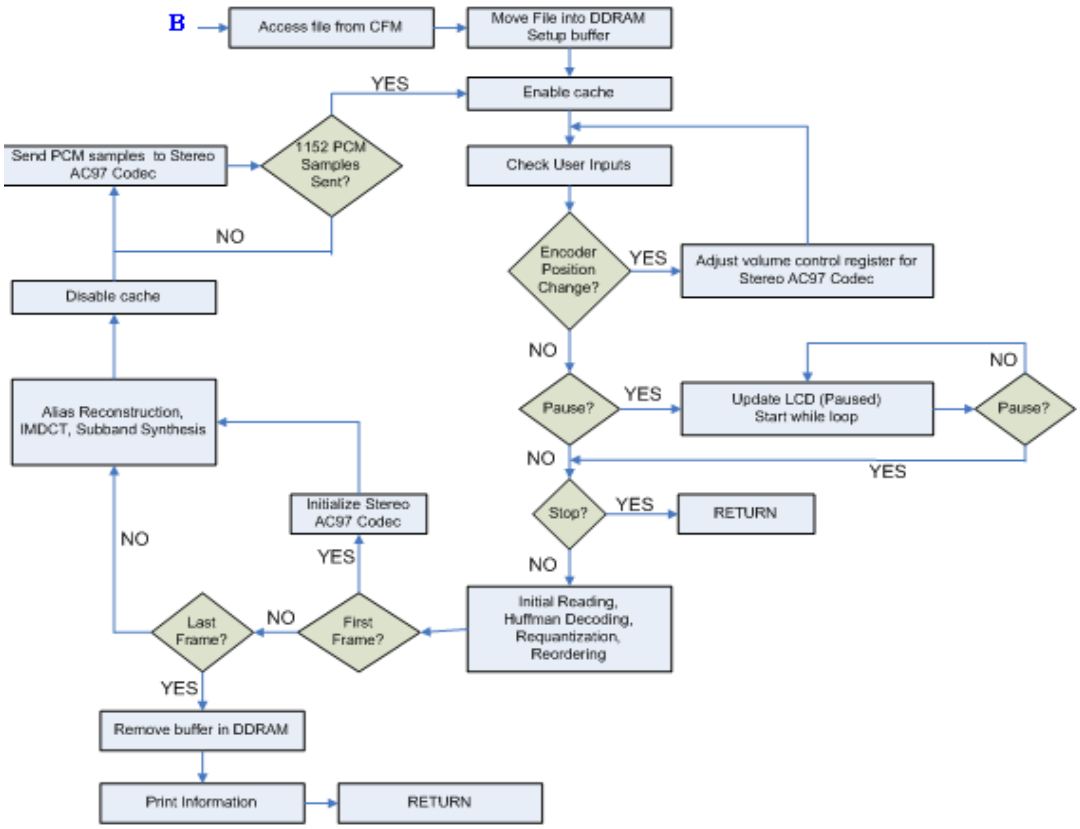


Figure 18-3. Flowchart for play_song function

A short description of the steps used in the play_song function, as illustrated in Figure 18-3 is needed. After the user selects a song, the MP3 file is accessed from the compact flash memory, that is, the "a: /" drive and the sysace_fopen function, that is defined in the Xilinx library, xilfatfs, for the SysAce compact flash controller.

To increase performance, the entire file is stored into the 256 Mb DDR memory. This is done essentially by setting up a software buffer and placing the program code (.text and .data sections) into the DDRAM. As a result, the audio output will be smoother (no gaps or breaks). However, with larger MP3 files (over 4 MB or more than 5 minutes), there will be a short delay (about 5 seconds or less) before decoding actually begins. Fortunately, for demo purposes I have only loaded songs less than this size. The other method (which was not used) is to move a frame of the MP3 bitstream into the DDRAM, decode and sent a frame's worth of PCM samples to the stereo ac97 codec FIFO, and reiterate. Although there would be no latency, breaks in audio output may occur. For more details on this buffering method, please consult the bstdfile.c file, as well the Mpeg_Audio_Decoder function within madlidwithoutOS.c.

The cache is then enabled (if the eighth dipswitch is up). A quick check for a triggering of any user inputs is performed. If there is a change in position for the rotary encoder (dial) peripheral, a user-defined function is called to modify the value of a volume control register for the stereo ac97 codec, essentially altering the voltage level for the stereo ac97.
If the pause button is pressed, the program will wait in a while loop until the same button (pause=play=UP) has been pressed. If the stop button is pressed, the program will break out of this decoding loop and return back to main. The stack and heap values (memory allocations) should be large enough (both currently set to 50 Mb) so that decoding can continue after pressing stop. Otherwise, the MP3 decoder cannot continue, printing out "unrecoverable error: not enough memory."

If no user inputs are triggered, the decoding process will begin with the initial reading, Huffman decoding, requantization, and reordering stages. The program checks the frame count. For the first frame (of 1152 PCM samples), the stereo ac97 codec firsts needs to be initialized with appropriate settings, as can be seen in the setup_ac97 function on page 15 of this report. Most importantly, the sampling rate of the codec is set to the sampling rate of the MP3 input, that is, the sampling rate used in the MP3 encoding process of the input file. This was determined from the initial reading stage based on a look up table for 2 bits (see Table 8-2). The PCM samples are then synthesized using the alias reconstruction, IMDCT, and subband synthesis filter bank blocks. Before sending the PCM samples out to the stereo ac97 FIFO register, the cache is disabled and enabled afterwards. The same process continues until the entire frame (of exactly 1152 PCM samples) has been sent to the ac97 (and heard on the external speaker). The entire decoding loop continues until the last frame (or if the stop button has been pressed). After the last frame, the buffer used to store the MP3 input file is cleared and program returns to main, where the cache is disabled once again.

Building the hardware platform

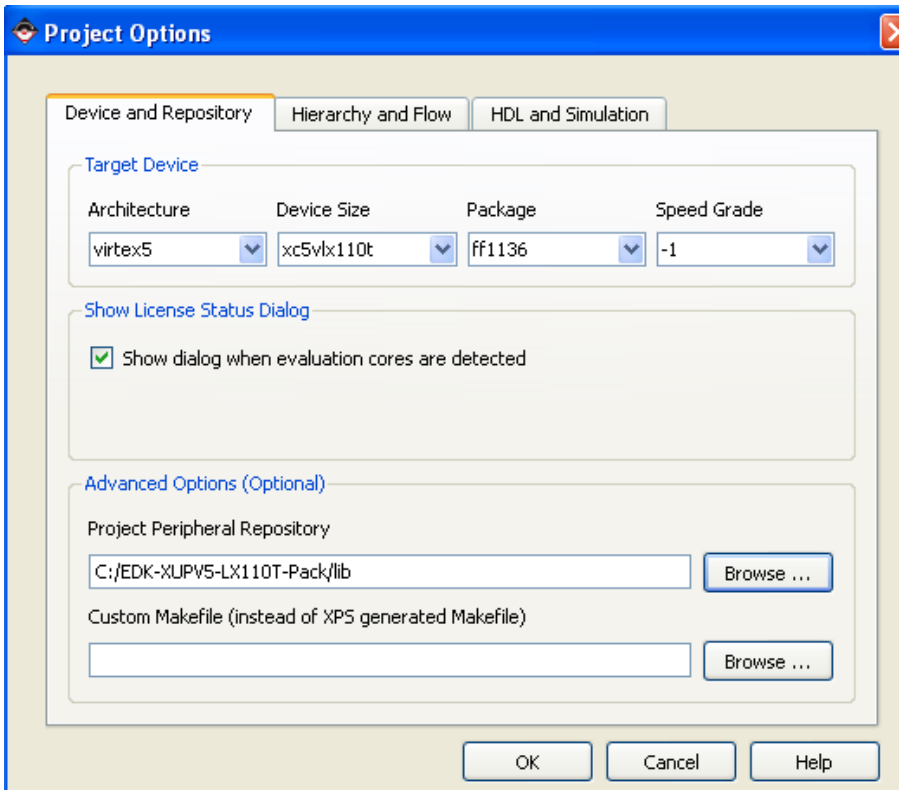
The following steps will guide you through the process of setting up the FPGA based MP3 Player system using Xilinx EDK 11.1.

Step 1
Launch Xilinx EDK 11.1 from the list of programs in the start menu.
Check the option, 'Base System Builder wizard' and click OK

Step 2

You will now be prompted to specify the location of a target folder to store your project. Enter the location of the folder or click on 'browse' to navigate to it. Also, check the option, 'Set Project Peripheral Repositories' and browse to the location of the folder that contains board specific hardware information (EDK-XUPV5-LX110T-Pack/lib => downloaded from Xilinx website).

Choosing a vendor, a board name and its version.



For the next steps, I choose to explain how to configure hardware for external peripherals manually in the .mhs and .mss files, instead of using the GUI provided by the EDK software platform (which involves adding IP from the IP catalog, connect buses, setting up port connections, and finally adding pin locations and description in the user constraints file (.ucf file).

MHS file

```
##############################################################################
# Created by Base System Builder Wizard for Xilinx EDK 11.2 Build EDK_LS2.6
# Thu Feb 25 10:27:17 2010
# Target Board:  Xilinx XUPV5-LX110T Evaluation Platform Rev A
# Family:    virtex5
# Device:    xc5vlx110t
# Package:   ff1136
# Speed Grade:  -1
# Processor number: 1
# Processor 1: microblaze_0
# System clock frequency: 125.0
# Debug Interface: On-Chip HW Debug Module
# ##############################################################################
 PARAMETER VERSION = 2.1.0

PORT fpga_0_LEDs_8Bit_GPIO_IO_pin = fpga_0_LEDs_8Bit_GPIO_IO_pin, DIR = IO, VEC = [0:7]
 PORT fpga_0_LEDs_Positions_GPIO_IO_pin = fpga_0_LEDs_Positions_GPIO_IO_pin, DIR = IO, VEC = [0:4]
 PORT fpga_0_Push_Buttons_5Bit_GPIO_IO_pin = fpga_0_Push_Buttons_5Bit_GPIO_IO_pin, DIR = IO, VEC = [0:4]
 PORT fpga_0_DIP_Switches_8Bit_GPIO_IO_pin = fpga_0_DIP_Switches_8Bit_GPIO_IO_pin, DIR = IO, VEC = [0:7]
 PORT fpga_0_SRAM_Mem_A_pin = fpga_0_SRAM_Mem_A_pin_vslice_7_30_concat, DIR = O, VEC = [7:30]
 PORT fpga_0_SRAM_Mem_CEN_pin = fpga_0_SRAM_Mem_CEN_pin, DIR = O
 PORT fpga_0_SRAM_Mem_OEN_pin = fpga_0_SRAM_Mem_OEN_pin, DIR = O
 PORT fpga_0_SRAM_Mem_WEN_pin = fpga_0_SRAM_Mem_WEN_pin, DIR = O
 PORT fpga_0_SRAM_Mem_BEN_pin = fpga_0_SRAM_Mem_BEN_pin, DIR = O, VEC = [0:3]
 PORT fpga_0_SRAM_Mem_ADV_LDN_pin = fpga_0_SRAM_Mem_ADV_LDN_pin, DIR = O
 PORT fpga_0_SRAM_Mem_DQ_pin = fpga_0_SRAM_Mem_DQ_pin, DIR = IO, VEC = [0:31]
 PORT fpga_0_SRAM_ZBT_CLK_OUT_pin = SRAM_CLK_OUT_s, DIR = O
 PORT fpga_0_SRAM_ZBT_CLK_FB_pin = SRAM_CLK_FB_s, DIR = I, SIGIS = CLK, CLK_FREQ = 125000000
 PORT fpga_0_DDR2_SDRAM_DDR2_Clk_pin = fpga_0_DDR2_SDRAM_DDR2_Clk_pin, DIR = O, VEC = [1:0]
 PORT fpga_0_DDR2_SDRAM_DDR2_Clk_n_pin = fpga_0_DDR2_SDRAM_DDR2_Clk_n_pin, DIR = O, VEC = [1:0]
 PORT fpga_0_DDR2_SDRAM_DDR2_CE_pin = fpga_0_DDR2_SDRAM_DDR2_CE_pin, DIR = O, VEC = [1:0]
 PORT fpga_0_DDR2_SDRAM_DDR2_CS_n_pin = fpga_0_DDR2_SDRAM_DDR2_CS_n_pin, DIR = O, VEC = [1:0]
 PORT fpga_0_DDR2_SDRAM_DDR2_ODT_pin = fpga_0_DDR2_SDRAM_DDR2_ODT_pin, DIR = O, VEC = [1:0]
 PORT fpga_0_DDR2_SDRAM_DDR2_RAS_n_pin = fpga_0_DDR2_SDRAM_DDR2_RAS_n_pin, DIR = O
 PORT fpga_0_DDR2_SDRAM_DDR2_CAS_n_pin = fpga_0_DDR2_SDRAM_DDR2_CAS_n_pin, DIR = O
 PORT fpga_0_DDR2_SDRAM_DDR2_WE_n_pin = fpga_0_DDR2_SDRAM_DDR2_WE_n_pin, DIR = O
 PORT fpga_0_DDR2_SDRAM_DDR2_BankAddr_pin = fpga_0_DDR2_SDRAM_DDR2_BankAddr_pin, DIR = O, VEC = [1:0]
 PORT fpga_0_DDR2_SDRAM_DDR2_Addr_pin = fpga_0_DDR2_SDRAM_DDR2_Addr_pin, DIR = O, VEC = [12:0]
 PORT fpga_0_DDR2_SDRAM_DDR2_DQ_pin = fpga_0_DDR2_SDRAM_DDR2_DQ_pin, DIR = IO, VEC = [63:0]
 PORT fpga_0_DDR2_SDRAM_DDR2_DM_pin = fpga_0_DDR2_SDRAM_DDR2_DM_pin, DIR = O, VEC = [7:0]
 PORT fpga_0_DDR2_SDRAM_DDR2_DQS_pin = fpga_0_DDR2_SDRAM_DDR2_DQS_pin, DIR = IO, VEC = [7:0]
 PORT fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin = fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin, DIR = IO, VEC = [7:0]
 PORT fpga_0_SysACE_CompactFlash_SysACE_MPA_pin = fpga_0_SysACE_CompactFlash_SysACE_MPA_pin, DIR = O, VEC = [6:0]
 PORT fpga_0_SysACE_CompactFlash_SysACE_CLK_pin = fpga_0_SysACE_CompactFlash_SysACE_CLK_pin, DIR = I
 PORT fpga_0_SysACE_CompactFlash_SysACE_MPIRQ_pin = fpga_0_SysACE_CompactFlash_SysACE_MPIRQ_pin, DIR = I
 PORT fpga_0_SysACE_CompactFlash_SysACE_CEN_pin = fpga_0_SysACE_CompactFlash_SysACE_CEN_pin, DIR = O
 PORT fpga_0_SysACE_CompactFlash_SysACE_OEN_pin = fpga_0_SysACE_CompactFlash_SysACE_OEN_pin, DIR = O
 PORT fpga_0_SysACE_CompactFlash_SysACE_WEN_pin = fpga_0_SysACE_CompactFlash_SysACE_WEN_pin, DIR = O
 PORT fpga_0_SysACE_CompactFlash_SysACE_MPD_pin = fpga_0_SysACE_CompactFlash_SysACE_MPD_pin, DIR = IO, VEC = [15:0]
```

```
PORT fpga_0_clk_1_sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
 PORT fpga_0_rst_1_sys_rst_pin = sys_rst_s, DIR = I, SIGIS = RST, RST_POLARITY = 0
 PORT fpga_0_RS232_Uart_1_sin_pin = fpga_0_RS232_Uart_1_sin, DIR = I
 PORT fpga_0_RS232_Uart_1_sout_pin = fpga_0_RS232_Uart_1_sout, DIR = O
 PORT lcd_ip_0_lcd_pin = lcd, DIR = O, VEC = [0:6]
 PORT opb_ac97_controller_ref_0_Bit_Clk_pin = opb_ac97_controller_ref_0_Bit_Clk, DIR = I, SIGIS = CLK
 PORT opb_ac97_controller_ref_0_Sync_pin = opb_ac97_controller_ref_0_Sync, DIR = O
 PORT opb_ac97_controller_ref_0_SData_Out_pin = opb_ac97_controller_ref_0_SData_Out, DIR = O
 PORT opb_ac97_controller_ref_0_SData_In_pin = opb_ac97_controller_ref_0_SData_In, DIR = I
 PORT flash_audio_reset_n = sys_periph_reset_n, DIR = O
 PORT Volume_Dial_GPIO_IO = Volume_Dial_GPIO_IO, DIR = IO, VEC = [0:2]
```

```
BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER C_FAMILY = virtex5
 PARAMETER C_INTERCONNECT = 1
 PARAMETER C_DEBUG_ENABLED = 1
 PARAMETER C_ICACHE_BASEADDR = 0xb0000000
 PARAMETER C_ICACHE_HIGHADDR = 0xbfffffff
 PARAMETER C_CACHE_BYTE_SIZE = 16384
 PARAMETER C_ICACHE_ALWAYS_USED = 1
 PARAMETER C_DCACHE_BASEADDR = 0xb0000000
 PARAMETER C_DCACHE_HIGHADDR = 0xbfffffff
 PARAMETER C_DCACHE_BYTE_SIZE = 16384
 PARAMETER C_DCACHE_ALWAYS_USED = 1
 PARAMETER HW_VER = 7.20.b
 PARAMETER C_USE_ICACHE = 1
 PARAMETER C_USE_DCACHE = 1
 PARAMETER C_ICACHE_LINE_LEN = 4
 PARAMETER C_DCACHE_LINE_LEN = 4
 PARAMETER C_DCACHE_USE_WRITEBACK = 1
 BUS_INTERFACE DPLB = mb_plb
 BUS_INTERFACE IPLB = mb_plb
 BUS_INTERFACE DXCL = microblaze_0_DXCL
 BUS_INTERFACE IXCL = microblaze_0_IXCL
 BUS_INTERFACE DEBUG = microblaze_0_mdm_bus
 BUS_INTERFACE DLMB = dlmb
 BUS_INTERFACE ILMB = ilmb
 PORT MB_RESET = mb_reset
END

BEGIN plb_v46
 PARAMETER INSTANCE = mb_plb
 PARAMETER C_FAMILY = virtex5
 PARAMETER HW_VER = 1.04.a
 PORT PLB_Clk = clk_125_0000MHzPLL0
 PORT SYS_Rst = sys_bus_reset
END
```

```
BEGIN lmb_v10
 PARAMETER INSTANCE = ilmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = clk_125_0000MHzPLL0
 PORT SYS_Rst = sys_bus_reset
END

BEGIN lmb_v10
 PARAMETER INSTANCE = dlmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = clk_125_0000MHzPLL0
 PORT SYS_Rst = sys_bus_reset
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = dlmb_cntlr
 PARAMETER HW_VER = 2.10.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0000ffff
 BUS_INTERFACE SLMB = dlmb
 BUS_INTERFACE BRAM_PORT = dlmb_port
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = ilmb_cntlr
 PARAMETER HW_VER = 2.10.b
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0000ffff
 BUS_INTERFACE SLMB = ilmb
 BUS_INTERFACE BRAM_PORT = ilmb_port
END

BEGIN bram_block
 PARAMETER INSTANCE = lmb_bram
 PARAMETER C_FAMILY = virtex5
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = ilmb_port
 BUS_INTERFACE PORTB = dlmb_port
END
```

```
BEGIN xps_gpio
 PARAMETER INSTANCE = LEDs_8Bit
 PARAMETER C_FAMILY = virtex5
 PARAMETER C_ALL_INPUTS = 0
 PARAMETER C_GPIO_WIDTH = 8
 PARAMETER C_INTERRUPT_PRESENT = 0
 PARAMETER C_IS_DUAL = 0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_BASEADDR = 0x81440000
 PARAMETER C_HIGHADDR = 0x8144ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT GPIO_IO = fpga_0_LEDs_8Bit_GPIO_IO_pin
END

BEGIN xps_gpio
 PARAMETER INSTANCE = LEDs_Positions
 PARAMETER C_FAMILY = virtex5
 PARAMETER C_ALL_INPUTS = 0
 PARAMETER C_GPIO_WIDTH = 5
 PARAMETER C_INTERRUPT_PRESENT = 0
 PARAMETER C_IS_DUAL = 0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_BASEADDR = 0x81420000
 PARAMETER C_HIGHADDR = 0x8142ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT GPIO_IO = fpga_0_LEDs_Positions_GPIO_IO_pin
END

BEGIN xps_gpio
 PARAMETER INSTANCE = Push_Buttons_5Bit
 PARAMETER C_FAMILY = virtex5
 PARAMETER C_ALL_INPUTS = 1
 PARAMETER C_GPIO_WIDTH = 5
 PARAMETER C_INTERRUPT_PRESENT = 0
 PARAMETER C_IS_DUAL = 0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_BASEADDR = 0x81400000
 PARAMETER C_HIGHADDR = 0x8140ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT GPIO_IO=fpga_0_Push_Buttons_5Bit_GPIO_IO_pin
END

BEGIN xps_gpio
 PARAMETER INSTANCE = DIP_Switches_8Bit
 PARAMETER C_FAMILY = virtex5
 PARAMETER C_ALL_INPUTS = 1
 PARAMETER C_GPIO_WIDTH = 8
 PARAMETER C_INTERRUPT_PRESENT = 0
 PARAMETER C_IS_DUAL = 0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_BASEADDR = 0x81460000
 PARAMETER C_HIGHADDR = 0x8146ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT GPIO_IO= fpga_0_DIP_Switches_8Bit_GPIO_IO_pin
END
```

```
BEGIN xps_mch_emc
 PARAMETER INSTANCE = SRAM
 PARAMETER C_FAMILY = virtex5
 PARAMETER C_NUM_BANKS_MEM = 1
 PARAMETER C_NUM_CHANNELS = 0
 PARAMETER C_MEM0_WIDTH = 32
 PARAMETER C_MAX_MEM_WIDTH = 32
 PARAMETER C_INCLUDE_DATAWIDTH_MATCHING_0 = 0
 PARAMETER C_SYNCH_MEM_0 = 1
 PARAMETER C_TCEDV_PS_MEM_0 = 0
 PARAMETER C_TAVDV_PS_MEM_0 = 0
 PARAMETER C_THZCE_PS_MEM_0 = 0
 PARAMETER C_THZOE_PS_MEM_0 = 0
 PARAMETER C_TWC_PS_MEM_0 = 0
 PARAMETER C_TWP_PS_MEM_0 = 0
 PARAMETER C_TLZWE_PS_MEM_0 = 0
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_MEM0_BASEADDR = 0x20100000
 PARAMETER C_MEM0_HIGHADDR = 0x201fffff
 BUS_INTERFACE SPLB = mb_plb
 PORT RdClk = clk_125_0000MHzPLL0
 PORT Mem_A = 0b0000000 &
fpga_0_SRAM_Mem_A_pin_vslice_7_30_concat & 0b0
 PORT Mem_CEN = fpga_0_SRAM_Mem_CEN_pin
 PORT Mem_OEN = fpga_0_SRAM_Mem_OEN_pin
 PORT Mem_WEN = fpga_0_SRAM_Mem_WEN_pin
 PORT Mem_BEN = fpga_0_SRAM_Mem_BEN_pin
 PORT Mem_ADV_LDN = fpga_0_SRAM_Mem_ADV_LDN_pin
 PORT Mem_DQ = fpga_0_SRAM_Mem_DQ_pin
END

BEGIN xps_sysace
 PARAMETER INSTANCE = SysACE_CompactFlash
 PARAMETER C_MEM_WIDTH = 16
 PARAMETER C_FAMILY = virtex5
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x83600000
 PARAMETER C_HIGHADDR = 0x8360ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT SysACE_MPA =
fpga_0_SysACE_CompactFlash_SysACE_MPA_pin
 PORT SysACE_CLK =
fpga_0_SysACE_CompactFlash_SysACE_CLK_pin
 PORT SysACE_MPIRQ =
fpga_0_SysACE_CompactFlash_SysACE_MPIRQ_pin
 PORT SysACE_CEN =
fpga_0_SysACE_CompactFlash_SysACE_CEN_pin
 PORT SysACE_OEN =
fpga_0_SysACE_CompactFlash_SysACE_OEN_pin
 PORT SysACE_WEN =
fpga_0_SysACE_CompactFlash_SysACE_WEN_pin
 PORT SysACE_MPD =
fpga_0_SysACE_CompactFlash_SysACE_MPD_pin
END
```

```
BEGIN mpmc
 PARAMETER INSTANCE = DDR2_SDRAM
 PARAMETER C_FAMILY = virtex5
 PARAMETER C_NUM_PORTS = 2
 PARAMETER C_NUM_IDELAYCTRL = 3
 PARAMETER C_IDELAYCTRL_LOC = IDELAYCTRL_X0Y6-
IDELAYCTRL_X0Y2-IDELAYCTRL_X0Y1
 PARAMETER C_MEM_PARTNO = mt4htf3264h-53e
 PARAMETER C_MEM_CLK_WIDTH = 2
 PARAMETER C_MEM_ODT_WIDTH = 2
 PARAMETER C_MEM_CE_WIDTH = 2
 PARAMETER C_MEM_CS_N_WIDTH = 2
 PARAMETER C_DDR2_DQSN_ENABLE = 1
 PARAMETER C_PIM0_BASETYPE = 1
 PARAMETER C_PIM1_BASETYPE = 1
 PARAMETER HW_VER = 5.02.a
 PARAMETER C_MPMC_BASEADDR = 0xb0000000
 PARAMETER C_MPMC_HIGHADDR = 0xbfffffff
 BUS_INTERFACE XCL0 = microblaze_0_IXCL
 BUS_INTERFACE XCL1 = microblaze_0_DXCL
 PORT MPMC_Clk0 = clk_125_0000MHzPLL0
 PORT MPMC_Clk0_DIV2 = clk_62_5000MHzPLL0
 PORT MPMC_Clk90 = clk_125_0000MHz90PLL0
 PORT MPMC_Clk_200MHz = clk_200_0000MHz
 PORT MPMC_Rst = sys_periph_reset
 PORT DDR2_Clk = fpga_0_DDR2_SDRAM_DDR2_Clk_pin
 PORT DDR2_Clk_n = fpga_0_DDR2_SDRAM_DDR2_Clk_n_pin
 PORT DDR2_CE = fpga_0_DDR2_SDRAM_DDR2_CE_pin
 PORT DDR2_CS_n = fpga_0_DDR2_SDRAM_DDR2_CS_n_pin
 PORT DDR2_ODT = fpga_0_DDR2_SDRAM_DDR2_ODT_pin
 PORT DDR2_RAS_n = fpga_0_DDR2_SDRAM_DDR2_RAS_n_pin
 PORT DDR2_CAS_n = fpga_0_DDR2_SDRAM_DDR2_CAS_n_pin
 PORT DDR2_WE_n = fpga_0_DDR2_SDRAM_DDR2_WE_n_pin
 PORT DDR2_BankAddr = fpga_0_DDR2_SDRAM_DDR2_BankAddr_pin
 PORT DDR2_Addr = fpga_0_DDR2_SDRAM_DDR2_Addr_pin
 PORT DDR2_DQ = fpga_0_DDR2_SDRAM_DDR2_DQ_pin
 PORT DDR2_DM = fpga_0_DDR2_SDRAM_DDR2_DM_pin
 PORT DDR2_DQS = fpga_0_DDR2_SDRAM_DDR2_DQS_pin
 PORT DDR2_DQS_n = fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin
END

BEGIN plbv46_opb_bridge
 PARAMETER INSTANCE = plbv46_opb_bridge_0
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_NUM_ADDR_RNG = 1
 PARAMETER C_RNG0_BASEADDR = 0xffff8000
 PARAMETER C_RNG0_HIGHADDR = 0xffff80ff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE MOPB = opb_v20_0
END

BEGIN util_vector_logic
 PARAMETER INSTANCE = util_vector_logic_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_OPERATION = not
 PARAMETER C_SIZE = 1
 PORT Op1 = sys_periph_reset
 PORT Res = sys_periph_reset_n
END
```

```
BEGIN clock_generator
 PARAMETER INSTANCE = clock_generator_0
 PARAMETER C_CLKIN_FREQ = 100000000
 PARAMETER C_CLKFBIN_FREQ = 125000000
 PARAMETER C_CLKOUT0_FREQ = 125000000
 PARAMETER C_CLKOUT0_PHASE = 90
 PARAMETER C_CLKOUT0_GROUP = PLL0
 PARAMETER C_CLKOUT0_BUF = TRUE
 PARAMETER C_CLKOUT1_FREQ = 125000000
 PARAMETER C_CLKOUT1_PHASE = 0
 PARAMETER C_CLKOUT1_GROUP = PLL0
 PARAMETER C_CLKOUT1_BUF = TRUE
 PARAMETER C_CLKOUT2_FREQ = 200000000
 PARAMETER C_CLKOUT2_PHASE = 0
 PARAMETER C_CLKOUT2_GROUP = NONE
 PARAMETER C_CLKOUT2_BUF = TRUE
 PARAMETER C_CLKOUT3_FREQ = 62500000
 PARAMETER C_CLKOUT3_PHASE = 0
 PARAMETER C_CLKOUT3_GROUP = PLL0
 PARAMETER C_CLKOUT3_BUF = TRUE
 PARAMETER C_CLKFBOUT_FREQ = 125000000
 PARAMETER C_CLKFBOUT_BUF = TRUE
 PARAMETER HW_VER = 3.01.a
 PORT CLKIN = dcm_clk_s
 PORT CLKFBIN = SRAM_CLK_FB_s
 PORT CLKOUT0 = clk_125_0000MHz90PLL0
 PORT CLKOUT1 = clk_125_0000MHzPLL0
 PORT CLKOUT2 = clk_200_0000MHz
 PORT CLKOUT3 = clk_62_5000MHzPLL0
 PORT CLKFBOUT = SRAM_CLK_OUT_s
 PORT RST = net_gnd
 PORT LOCKED = Dcm_all_locked
END

BEGIN mdm
 PARAMETER INSTANCE = debug_module
 PARAMETER HW_VER = 1.00.f
 PARAMETER C_MB_DBG_PORTS = 1
 PARAMETER C_USE_UART = 1
 PARAMETER C_UART_WIDTH = 8
 PARAMETER C_BASEADDR = 0x84400000
 PARAMETER C_HIGHADDR = 0x8440ffff
 BUS_INTERFACE SPLB = mb_plb
 BUS_INTERFACE MBDEBUG_0 = microblaze_0_mdm_bus
 PORT Debug_SYS_Rst = Debug_SYS_Rst
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = proc_sys_reset_0
 PARAMETER C_EXT_RESET_HIGH = 0
 PARAMETER HW_VER = 2.00.a
 PORT Slowest_sync_clk = clk_125_0000MHzPLL0
 PORT Ext_Reset_In = sys_rst_s
 PORT MB_Debug_Sys_Rst = Debug_SYS_Rst
 PORT Dcm_locked = Dcm_all_locked
 PORT MB_Reset = mb_reset
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT Peripheral_Reset = sys_periph_reset
END
BEGIN opb_v20
 PARAMETER INSTANCE = opb_v20_0
 PARAMETER HW_VER = 1.10.c
 PORT OPB_Clk = clk_125_0000MHzPLL0
 PORT SYS_Rst = net_gnd
END
```

```
BEGIN xps_uart16550
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_IS_A_16550 = 1
 PARAMETER C_BASEADDR = 0x83e20000
 PARAMETER C_HIGHADDR = 0x83e2ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT sin = fpga_0_RS232_Uart_1_sin
 PORT sout = fpga_0_RS232_Uart_1_sout
END

BEGIN lcd_ip
 PARAMETER INSTANCE = lcd_ip_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xcf400000
 PARAMETER C_HIGHADDR = 0xcf40ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT lcd = lcd
END

BEGIN opb_ac97_controller_ref
 PARAMETER INSTANCE = opb_ac97_controller_ref_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_RECORD = 0
 PARAMETER C_BASEADDR = 0xffff8000
 PARAMETER C_HIGHADDR = 0xffff80ff
 BUS_INTERFACE SOPB = opb_v20_0
 PORT Bit_Clk = opb_ac97_controller_ref_0_Bit_Clk
 PORT Sync = opb_ac97_controller_ref_0_Sync
 PORT SData_Out = opb_ac97_controller_ref_0_SData_Out
 PORT SData_In = opb_ac97_controller_ref_0_SData_In
END

BEGIN xps_gpio
 PARAMETER INSTANCE = Volume_Dial
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_INTERRUPT_PRESENT = 1
 PARAMETER C_GPIO_WIDTH = 3
 PARAMETER C_BASEADDR = 0x40000000
 PARAMETER C_HIGHADDR = 0x4000ffff
 BUS_INTERFACE SPLB = mb_plb
 PORT GPIO_IO = Volume_Dial_GPIO_IO
END

BEGIN xps_timer
 PARAMETER INSTANCE = xps_timer_0
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BASEADDR = 0x20000000
 PARAMETER C_HIGHADDR = 0x2000ffff
 BUS_INTERFACE SPLB = mb_plb
END
```

END of MHS file

MSS File

```
PARAMETER VERSION = 2.2.0

BEGIN OS
 PARAMETER OS_NAME = standalone
 PARAMETER OS_VER = 2.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
 PARAMETER STDIN = debug_module
 PARAMETER STDOUT = debug_module
END


BEGIN PROCESSOR
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.12.a
 PARAMETER HW_INSTANCE = microblaze_0
 PARAMETER COMPILER = mb-gcc
 PARAMETER ARCHIVER = mb-ar
END


BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = dlmb_cntlr
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = bram
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = ilmb_cntlr
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = lmb_bram
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = gpio
 PARAMETER DRIVER_VER = 2.13.a
 PARAMETER HW_INSTANCE = LEDs_8Bit
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = gpio
 PARAMETER DRIVER_VER = 2.13.a
 PARAMETER HW_INSTANCE = LEDs_Positions
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = plbv46_opb_bridge_0
END
```

```
BEGIN DRIVER
 PARAMETER DRIVER_NAME = gpio
 PARAMETER DRIVER_VER = 2.13.a
 PARAMETER HW_INSTANCE = Push_Buttons_5Bit
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = gpio
 PARAMETER DRIVER_VER = 2.13.a
 PARAMETER HW_INSTANCE = DIP_Switches_8Bit
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = emc
 PARAMETER DRIVER_VER = 2.00.a
 PARAMETER HW_INSTANCE = SRAM
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = mpmc
 PARAMETER DRIVER_VER = 3.00.a
 PARAMETER HW_INSTANCE = DDR2_SDRAM
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = sysace
 PARAMETER DRIVER_VER = 1.12.a
 PARAMETER HW_INSTANCE = SysACE_CompactFlash
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = clock_generator_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.14.a
 PARAMETER HW_INSTANCE = debug_module
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = proc_sys_reset_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = opbarb
 PARAMETER DRIVER_VER = 1.02.a
 PARAMETER HW_INSTANCE = opb_v20_0
END
```

```
BEGIN DRIVER
 PARAMETER DRIVER_NAME = uartns550
 PARAMETER DRIVER_VER = 1.11.a
 PARAMETER HW_INSTANCE = RS232_Uart_1
 PARAMETER CLOCK_HZ = 125000000
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = lcd_ip
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = lcd_ip_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE =
opb_ac97_controller_ref_0
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = gpio
 PARAMETER DRIVER_VER = 2.13.a
 PARAMETER HW_INSTANCE = Volume_Dial
END

BEGIN DRIVER
 PARAMETER DRIVER_NAME = tmrctr
 PARAMETER DRIVER_VER = 1.11.a
 PARAMETER HW_INSTANCE = xps_timer_0
END

BEGIN LIBRARY
 PARAMETER LIBRARY_NAME = xilfatfs
 PARAMETER LIBRARY_VER = 1.00.a
 PARAMETER PROC_INSTANCE = microblaze_0
END
```

END of MSS file

UCF File

```
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<0> LOC = AE24 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<1> LOC = AD24 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<2> LOC = AD25 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<3> LOC = G16 | IOSTANDARD=LVCMOS25 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<4> LOC = AD26 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<5> LOC = G15 | IOSTANDARD=LVCMOS25 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<6> LOC = L18 | IOSTANDARD=LVCMOS25 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_8Bit_GPIO_IO_pin<7> LOC = H18 | IOSTANDARD=LVCMOS25 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_Positions_GPIO_IO_pin<0> LOC=E8 | IOSTANDARD=LVCMOS33 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_Positions_GPIO_IO_pin<1> LOC=AF23 | IOSTANDARD=LVCMOS33 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_Positions_GPIO_IO_pin<2> LOC=AG12 | IOSTANDARD=LVCMOS33 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_Positions_GPIO_IO_pin<3> LOC=AG23 | IOSTANDARD=LVCMOS33 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_LEDs_Positions_GPIO_IO_pin<4> LOC=AF13 | IOSTANDARD=LVCMOS33 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_Push_Buttons_5Bit_GPIO_IO_pin<0> LOC = AJ6 | IOSTANDARD=LVCMOS33 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_Push_Buttons_5Bit_GPIO_IO_pin<1> LOC = AJ7 | IOSTANDARD=LVCMOS33 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_Push_Buttons_5Bit_GPIO_IO_pin<2> LOC = V8 | IOSTANDARD=LVCMOS33 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_Push_Buttons_5Bit_GPIO_IO_pin<3> LOC = AK7 | IOSTANDARD=LVCMOS33 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_Push_Buttons_5Bit_GPIO_IO_pin<4> LOC = U8 | IOSTANDARD=LVCMOS33 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<0> LOC=U25 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<1> LOC=AG27 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<2> LOC=AF25 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<3> LOC=AF26 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<4> LOC=AE27 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<5> LOC=AE26 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<6> LOC=AC25 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
Net fpga_0_DIP_Switches_8Bit_GPIO_IO_pin<7> LOC=AC24 | IOSTANDARD=LVCMOS18 | PULLDOWN | SLEW=SLOW | DRIVE=2;
```

```
## SYSACE Compact Flash
Net fpga_0_SysACE_CompactFlash_SysACE_MPA_pin<0> LOC=G5 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPA_pin<1> LOC=N7 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPA_pin<2> LOC=N5 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPA_pin<3> LOC=P5 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPA_pin<4> LOC=R6 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPA_pin<5> LOC=M6 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPA_pin<6> LOC=L6 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_CLK_pin LOC=AH17 | IOSTANDARD = LVCMOS33 | PERIOD = 30000 ps;
Net fpga_0_SysACE_CompactFlash_SysACE_MPIRQ_pin LOC=M7 | IOSTANDARD = LVCMOS33 | TIG;
Net fpga_0_SysACE_CompactFlash_SysACE_CEN_pin LOC=M5 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_OEN_pin LOC=N8 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_WEN_pin LOC=R9 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<0> LOC=P9 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<1> LOC=T8 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<2> LOC=J7 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<3> LOC=H7 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<4> LOC=R7 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<5> LOC=U7 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<6> LOC=P7 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<7> LOC=P6 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<8> LOC=R8 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<9> LOC=L5 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<10> LOC=L4 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<11> LOC=K6 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<12> LOC=J5 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<13> LOC=T6 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<14> LOC=K7 | IOSTANDARD = LVCMOS33;
Net fpga_0_SysACE_CompactFlash_SysACE_MPD_pin<15> LOC=J6 | IOSTANDARD = LVCMOS33;
```

```
## SRAM
Net fpga_0_SRAM_Mem_A_pin<30> LOC=K12 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<29> LOC=K13 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<28> LOC=H23 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<27> LOC=G23 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<26> LOC=H12 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<25> LOC=J12 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<24> LOC=K22 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<23> LOC=K23 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<22> LOC=K14 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<21> LOC=L14 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<20> LOC=H22 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<19> LOC=G22 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<18> LOC=J15 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<17> LOC=K16 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<16> LOC=K21 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<15> LOC=J22 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<14> LOC=L16 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<13> LOC=L15 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<12> LOC=L20 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<11> LOC=L21 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<10> LOC=AE23 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<9> LOC=AE22 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<8> LOC=AE12 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_A_pin<7> LOC=AE13 | SLEW = FAST | DRIVE = 8 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_CEN_pin LOC=J10 | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_OEN_pin LOC=B12 | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_WEN_pin LOC=AF20 | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_BEN_pin<3> LOC=J11 | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_BEN_pin<2> LOC=K11 | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_BEN_pin<1> LOC=D10 | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_BEN_pin<0> LOC=D11 | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_ADV_LDN_pin LOC=H8 | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<0> LOC=AG22 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<1> LOC=AH22 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<2> LOC=AH12 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<3> LOC=AG13 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<4> LOC=AH20 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<5> LOC=AH19 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<6> LOC=AH14 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<7> LOC=AH13 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<8> LOC=AF15 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<9> LOC=AE16 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<10> LOC=AE21 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<11> LOC=AD20 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<12> LOC=AF16 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<13> LOC=AE17 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<14> LOC=AE19 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<15> LOC=AD19 | PULLDOWN | SLEW = FAST | DRIVE = 12 | IOSTANDARD=LVCMOS33;
Net fpga_0_SRAM_Mem_DQ_pin<16> LOC=J9 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<17> LOC=K8 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<18> LOC=K9 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<19> LOC=B13 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<20> LOC=C13 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<21> LOC=G11 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<22> LOC=G12 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<23> LOC=M8 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<24> LOC=L8 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<25> LOC=F11 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<26> LOC=E11 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<27> LOC=M10 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<28> LOC=L9 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<29> LOC=E12 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<30> LOC=E13 | PULLDOWN | IOSTANDARD=LVDCI_33;
Net fpga_0_SRAM_Mem_DQ_pin<31> LOC=N10 | PULLDOWN | IOSTANDARD=LVDCI_33;
```

```
Net fpga_0_DDR2_SDRAM_DDR2_Addr_pin<3> LOC=P29 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_Addr_pin<4> LOC=K31 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_Addr_pin<5> LOC=L31 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_Addr_pin<6> LOC=P31 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_Addr_pin<7> LOC=P30 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_Addr_pin<8> LOC=M31 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_Addr_pin<9> LOC=R28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_Addr_pin<10> LOC=J31 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_Addr_pin<11> LOC=R29 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_Addr_pin<12> LOC=T31 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<0> LOC=AF30 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<1> LOC=AK31 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<2> LOC=AF31 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<3> LOC=AD30 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<4> LOC=AJ30 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<5> LOC=AF29 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<6> LOC=AD29 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<7> LOC=AE29 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<8> LOC=AH27 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<9> LOC=AF28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<10> LOC=AH28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<11> LOC=AA28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<12> LOC=AG25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<13> LOC=AJ26 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<14> LOC=AG28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<15> LOC=AB28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<16> LOC=AC28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<17> LOC=AB25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<18> LOC=AC27 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<19> LOC=AA26 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<20> LOC=AB26 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<21> LOC=AA24 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<22> LOC=AB27 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<23> LOC=AA25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<24> LOC=AC29 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<25> LOC=AB30 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<26> LOC=W31 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<27> LOC=V30 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<28> LOC=AC30 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<29> LOC=W29 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<30> LOC=V27 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<31> LOC=W27 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<32> LOC=V29 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<33> LOC=Y27 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<34> LOC=Y26 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<35> LOC=W24 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<36> LOC=V28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<37> LOC=W25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<38> LOC=W26 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<39> LOC=V24 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<40> LOC=R24 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<41> LOC=P25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<42> LOC=N24 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<43> LOC=P26 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<44> LOC=T24 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<45> LOC=N25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<46> LOC=P27 | IOSTANDARD = SSTL18_II;
```

```
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<47> LOC=N28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<48> LOC=M28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<49> LOC=L28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<50> LOC=F25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<51> LOC=H25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<52> LOC=K27 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<53> LOC=K28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<54> LOC=H24 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<55> LOC=G26 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<56> LOC=G25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<57> LOC=M26 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<58> LOC=J24 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<59> LOC=L26 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<60> LOC=J27 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<61> LOC=M25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<62> LOC=L25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQ_pin<63> LOC=L24 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<0> LOC=AJ31 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<1> LOC=AE28 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<2> LOC=Y24 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<3> LOC=Y31 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<4> LOC=V25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<5> LOC=P24 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<6> LOC=F26 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DM_pin<7> LOC=J25 | IOSTANDARD = SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<0> LOC=AA29 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<1> LOC=AK28 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<2> LOC=AK26 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<3> LOC=AB31 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<4> LOC=Y28 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<5> LOC=E26 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<6> LOC=H28 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_pin<7> LOC=G27 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin<0> LOC=AA30 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin<1> LOC=AK27 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin<2> LOC=AJ27 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin<3> LOC=AA31 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin<4> LOC=Y29 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin<5> LOC=E27 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin<6> LOC=G28 | IOSTANDARD = DIFF_SSTL18_II;
Net fpga_0_DDR2_SDRAM_DDR2_DQS_n_pin<7> LOC=H27 | IOSTANDARD = DIFF_SSTL18_II;

Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100000 kHz;
Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin | LOC = AH15 | IOSTANDARD=LVCMOS33;
Net fpga_0_rst_1_sys_rst_pin TIG;
Net fpga_0_rst_1_sys_rst_pin LOC = E9 | IOSTANDARD=LVCMOS33 | PULLUP;

###### DDR2_SDRAM
# Define multicycle paths - these paths may take longer because additional
# time allowed for logic to settle in calibration/initialization FSM

# MUX Select for either rising/falling CLK0 for 2nd stage read capture
INST "*/u_phy_calib_0/gen_rd_data_sel*.u_ff_rd_data_sel" TNM = "TNM_RD_DATA_SEL";
TIMESPEC "TS_MC_RD_DATA_SEL" = FROM "TNM_RD_DATA_SEL" TO FFS
  "TS_sys_clk_pin" * 2;
# MUX select for read data - optional delay on data to account for byte skews
```

```
#INST "*/u_usr_rd_0/gen_rden_sel_mux*.u_ff_rden_sel_mux" TNM = "TNM_RDEN_SEL_MUX";
#TIMESPEC "TS_MC_RDEN_SEL_MUX" = FROM "TNM_RDEN_SEL_MUX" TO FFS   "TS_sys_clk_pin"
* 2;
# Calibration/Initialization complete status flag (for PHY logic only)
INST "*/u_phy_init_0/u_ff_phy_init_data_sel" TNM = "TNM_PHY_INIT_DATA_SEL";
TIMESPEC "TS_MC_PHY_INIT_DATA_SEL_0" = FROM "TNM_PHY_INIT_DATA_SEL" TO
 FFS "TS_sys_clk_pin" * 2;
TIMESPEC "TS_MC_PHY_INIT_DATA_SEL_90" = FROM "TNM_PHY_INIT_DATA_SEL" TO
  FFS "TS_sys_clk_pin" * 2;
# Select (address) bits for SRL32 shift registers used in stage3/stage4
# calibration
INST "*/u_phy_calib_0/gen_gate_dly*.u_ff_gate_dly" TNM = "TNM_GATE_DLY";
TIMESPEC "TS_MC_GATE_DLY" = FROM "TNM_GATE_DLY" TO FFS "TS_sys_clk_pin" * 2;
INST "*/u_phy_calib_0/gen_rden_dly*.u_ff_rden_dly" TNM = "TNM_RDEN_DLY";
TIMESPEC "TS_MC_RDEN_DLY" = FROM "TNM_RDEN_DLY" TO FFS "TS_sys_clk_pin" * 2;
INST "*/u_phy_calib_0/gen_cal_rden_dly*.u_ff_cal_rden_dly"
  TNM = "TNM_CAL_RDEN_DLY";
TIMESPEC "TS_MC_CAL_RDEN_DLY" = FROM "TNM_CAL_RDEN_DLY" TO FFS
  "TS_sys_clk_pin" * 2;

# LOC placement of DQS-squelch related IDDR and IDELAY elements
# Each circuit can be located at any of the following locations:
#  1. Ununsed "N"-side of DQS diff pair I/O
#  2. DM data mask (output only, input side is free for use)
#  3. Any output-only site

INST "*/gen_dqs[0].u_iob_dqs/u_iddr_dq_ce"  LOC = "ILOGIC_X0Y96";
INST "*/gen_dqs[0].u_iob_dqs/u_iodelay_dq_ce"  LOC = "IODELAY_X0Y96";
INST "*/gen_dqs[1].u_iob_dqs/u_iddr_dq_ce"  LOC = "ILOGIC_X0Y58";
INST "*/gen_dqs[1].u_iob_dqs/u_iodelay_dq_ce"  LOC = "IODELAY_X0Y58";
INST "*/gen_dqs[2].u_iob_dqs/u_iddr_dq_ce"  LOC = "ILOGIC_X0Y62";
INST "*/gen_dqs[2].u_iob_dqs/u_iodelay_dq_ce"  LOC = "IODELAY_X0Y62";
INST "*/gen_dqs[3].u_iob_dqs/u_iddr_dq_ce"  LOC = "ILOGIC_X0Y100";
INST "*/gen_dqs[3].u_iob_dqs/u_iodelay_dq_ce"  LOC = "IODELAY_X0Y100";
INST "*/gen_dqs[4].u_iob_dqs/u_iddr_dq_ce"  LOC = "ILOGIC_X0Y102";
INST "*/gen_dqs[4].u_iob_dqs/u_iodelay_dq_ce"  LOC = "IODELAY_X0Y102";
INST "*/gen_dqs[5].u_iob_dqs/u_iddr_dq_ce"  LOC = "ILOGIC_X0Y256";
INST "*/gen_dqs[5].u_iob_dqs/u_iodelay_dq_ce"  LOC = "IODELAY_X0Y256";
INST "*/gen_dqs[6].u_iob_dqs/u_iddr_dq_ce"  LOC = "ILOGIC_X0Y260";
INST "*/gen_dqs[6].u_iob_dqs/u_iodelay_dq_ce"  LOC = "IODELAY_X0Y260";
INST "*/gen_dqs[7].u_iob_dqs/u_iddr_dq_ce"  LOC = "ILOGIC_X0Y262";
INST "*/gen_dqs[7].u_iob_dqs/u_iodelay_dq_ce"  LOC = "IODELAY_X0Y262";

#### Module RS232_Uart_1 constraints
Net fpga_0_RS232_Uart_1_sin_pin LOC = AG15;
Net fpga_0_RS232_Uart_1_sin_pin IOSTANDARD=LVCMOS33;
Net fpga_0_RS232_Uart_1_sout_pin LOC = AG20;
Net fpga_0_RS232_Uart_1_sout_pin IOSTANDARD=LVCMOS33;
```

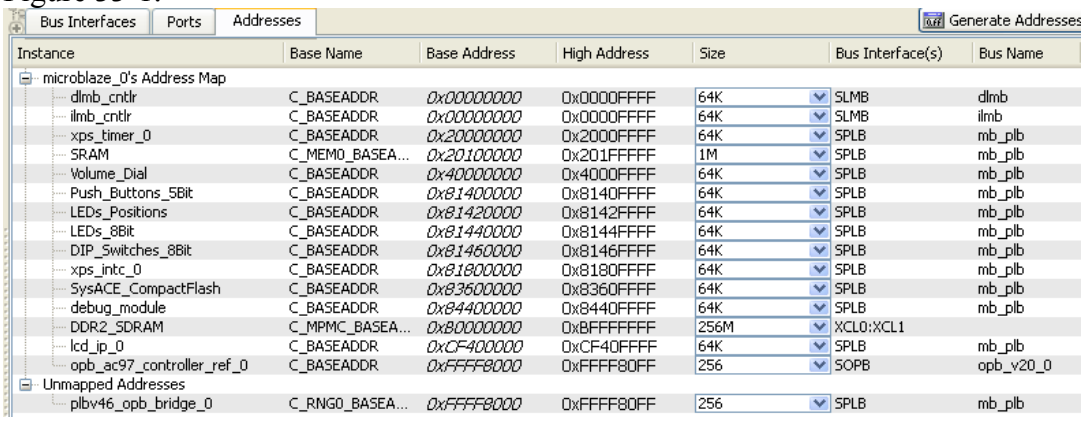
```
### LCD User constraints
NET lcd_ip_0_lcd_pin<0> LOC = AC9 | IOSTANDARD = LVCMOS33 | DRIVE = 2 | SLEW = SLOW; # LCD_E
NET lcd_ip_0_lcd_pin<1> LOC = J17 | IOSTANDARD = LVCMOS25  | DRIVE = 2 | SLEW = SLOW; # LCD_RS
NET lcd_ip_0_lcd_pin<2> LOC = AC10 | IOSTANDARD = LVCMOS33  | DRIVE = 2 | SLEW = SLOW; # LCD_RW
NET lcd_ip_0_lcd_pin<3> LOC = T11 | IOSTANDARD = LVCMOS33  | DRIVE = 2 | SLEW = SLOW; # LCD_DB7
NET lcd_ip_0_lcd_pin<4> LOC = G6 | IOSTANDARD = LVCMOS33  | DRIVE = 2 | SLEW = SLOW; # LCD_DB6
NET lcd_ip_0_lcd_pin<5> LOC = G7 | IOSTANDARD = LVCMOS33  | DRIVE = 2 | SLEW = SLOW; # LCD_DB5
NET lcd_ip_0_lcd_pin<6> LOC = T9 | IOSTANDARD = LVCMOS33 | DRIVE = 2 | SLEW = SLOW; # LCD_DB4
```

```
#### AC97 constraints
Net opb_ac97_controller_ref_0_SData_In_pin  LOC = AE18;
Net opb_ac97_controller_ref_0_SData_In_pin  IOSTANDARD = LVCMOS33;
Net opb_ac97_controller_ref_0_SData_Out_pin LOC = AG16;
Net opb_ac97_controller_ref_0_SData_Out_pin IOSTANDARD = LVCMOS33;
Net opb_ac97_controller_ref_0_Sync_pin      LOC = AF19;
Net opb_ac97_controller_ref_0_Sync_pin      IOSTANDARD = LVCMOS33;
Net opb_ac97_controller_ref_0_Bit_Clk_pin   LOC = AF18;
Net opb_ac97_controller_ref_0_Bit_Clk_pin   IOSTANDARD = LVCMOS33;
Net opb_ac97_controller_ref_0_Bit_Clk_pin   PERIOD = 80;
Net flash_audio_reset_n   LOC = AG17;
Net flash_audio_reset_n   IOSTANDARD = LVCMOS33;
```

```
#### Rotary Endcoder
# Rotary INC A
Net Volume_Dial_GPIO_IO<2> LOC = AH30;
Net Volume_Dial_GPIO_IO<2> IOSTANDARD=LVCMOS18;
Net Volume_Dial_GPIO_IO<2> PULLDOWN;
Net Volume_Dial_GPIO_IO<2> SLEW=SLOW;
Net Volume_Dial_GPIO_IO<2> DRIVE=2;
# Rotary INC B
Net Volume_Dial_GPIO_IO<1> LOC = AG30;
Net Volume_Dial_GPIO_IO<1> IOSTANDARD=LVCMOS18;
Net Volume_Dial_GPIO_IO<1> PULLDOWN;
Net Volume_Dial_GPIO_IO<1> SLEW=SLOW;
Net Volume_Dial_GPIO_IO<1> DRIVE=2;
# Rotary PUSH
Net Volume_Dial_GPIO_IO<0> LOC = AH29;
Net Volume_Dial_GPIO_IO<0> IOSTANDARD=LVCMOS18;
Net Volume_Dial_GPIO_IO<0> PULLDOWN;
Net Volume_Dial_GPIO_IO<0> SLEW=SLOW;
Net Volume_Dial_GPIO_IO<0> DRIVE=2;
```

END of UCF file

After configuring port connections, addresses locations for the peripherals must be set. Within the System Assembly View tab in EDK, choose the Addresses Tab and Generate Addresses. With the .mhs description used above, the addresses should look like that of Figure 33-1.



Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name
microblaze_0's Address Map						
dlmb_cntlr	C_BASEADDR	0x00000000	0x0000FFFF	64K	SLMB	dlmb
ilmb_cntlr	C_BASEADDR	0x00000000	0x0000FFFF	64K	SLMB	ilmb
xps_timer_0	C_BASEADDR	0x20000000	0x2000FFFF	64K	SPLB	mb_plb
SRAM	C_MEM0_BASEA...	0x20100000	0x201FFFFF	1M	SPLB	mb_plb
Volume_Dial	C_BASEADDR	0x40000000	0x4000FFFF	64K	SPLB	mb_plb
Push_Buttons_5Bit	C_BASEADDR	0x81400000	0x8140FFFF	64K	SPLB	mb_plb
LEDs_Positions	C_BASEADDR	0x81420000	0x8142FFFF	64K	SPLB	mb_plb
LEDs_8Bit	C_BASEADDR	0x81440000	0x8144FFFF	64K	SPLB	mb_plb
DIP_Switches_8Bit	C_BASEADDR	0x81460000	0x8146FFFF	64K	SPLB	mb_plb
xps_intc_0	C_BASEADDR	0x81800000	0x8180FFFF	64K	SPLB	mb_plb
SysACE_CompactFlash	C_BASEADDR	0x83600000	0x8360FFFF	64K	SPLB	mb_plb
debug_module	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb
DDR2_SDRAM	C_MPMC_BASEA...	0x80000000	0xBFFFFFFF	256M	XCL0:XCL1	
lcd_ip_0	C_BASEADDR	0xCF400000	0xCF40FFFF	64K	SPLB	mb_plb
opb_ac97_controller_ref_0	C_BASEADDR	0xFFFF8000	0xFFFF80FF	256	SOPB	opb_v20_0
Unmapped Addresses						
plbv46_opb_bridge_0	C_RNG0_BASEA...	0xFFFF8000	0xFFFF80FF	256	SPLB	mb_plb

Figure 33-1. Address configurations for FPGA MP3 player peripherals

Finally, generate the hardware configuration file, known as a .bit file.

Hardware=> Generate Bitstream
This process should take about 30-40 minutes. Any minor change in the mhs or ucf files will require a re-generation of this bitstream.

Building the software platform

Software => Add Software Application Project

Add all required .c and .h files as summarized by Table 34-1.
Declare memory allocations for program: Software => Generate linker script
Place all vector sections in the 256 Mb DDRAM. The stack and heap should also be placed into DDRAM. For more details, please refer to the Program size (memory usage) section of this report.

Software => Build All User Application

Table 34-1

C files	Lines of Code		**Header files**	Lines of code
timer.c	501		stereo_coefficients.h	68
synth.c	1414		windowing_coefficients.h	58
stream.c	154		imdct_coefficients.h	61
madlid_withoutOS.c	830		scalefactors.h	251
layer12.c	637		requantization.h	8637
layer3.c	2109		sysace_stdio.h	49
huffman.c	3092		timer.h	80
frame.c	511		synth.h	50
fixed.c	64		stream.h	88
decoder.c	577		layer12.h	53
bstdfile.c	208		layer3.h	40
bit.c	130		huffman.h	99
xac97_l.c	280		global.h	508
sleep.c	31		frame.h	70
lcd.c	204		fixed.h	
gpio.c	47		decoder.h	
			bstdfile.h	
Total lines (c files)	10789		bit.h	
			xcache_l.h	
			sleep.h	
			lcd.h	
			Total lines (h files)	10112
			Total lines of code	20901

Although a program with roughly 21,000 lines of code (including comments) may sound initially complex and overwhelming, at least 50% of the code involves large look up tables for coefficients and constants that are stated in the ISO standard document [5] and which enable the MAD decoder to be a universal and backwards compatible decoder. For more information

on this concept, please refer to the Universal and Backwards Compatibility sections in this report. A short description of each file, as well as the functions used, are found at the top of each .c file, where these files are placed the C and H files folder on the provided data disc.

Results

Specifications

The first few weeks have been dedicated to becoming familiar with the Embedded Development Kit Platform Studio software. Xilinx tutorials and board demonstration projects have been studied. In addition, peripheral setup, compilation, and debugging procedures have been practiced extensively.

C programs have been written to read from a compact flash memory (CFM). First, the CFM is preloaded with .txt or .dat files. Then, the C program running on the FPGA fetches the file (.txt or .dat) and outputs the contents on a hyperterminal. It proves that CFM is accessible using C programming. In the actual implementation of the project, MP3 files are preloaded instead.

The majority of specifications and goals were met. MP3 files can be accessed and read from the compact flash memory. The list of songs on the compact flash can be scanned and selected. The title can also be displayed on the LCD, although this does not require the decoding the ID3 tag. The MP3 bit stream can be decoded in real time and heard on an external speaker with adequate quality. The "STOP," "PAUSE," and "Volume control" functions was completed in the last week. Rewind and forward modes were never experimented due to time constraints.

Observations and Debugging Procedures

Testing and debugging on the Virtex2 board proved inconclusive. An attempt to run the MAD decoder on the Virtex2 was to determine whether the processor speed affected the decoding playback speed, since the Virtex 2 uses a hardware driven PowerPC processor at 300 MHz, about 3 times faster than the Virtex 5 software driven Microblaze processor at 100 MHz. No output audio resulting from the MAD decoder program was heard with a speaker, although the hardware-software interface for the opb_ac97 peripheral was verified to be correct. After obtaining a .bit and .elf from a base builder reference design project for the Virtex 2 Pro board, hardware connections (from the .mhs file) and software function (from the c code) was used to play a game sound representing by integer values (digital loopback).

Failure to get results with the project schedule time winding down, a switch to use to the Virtex 5 development board that would allow the use of a full user-interface subsystem including an LCD and rotary encoder (for volume control) was made.

The use of instruction and data cache significantly improved the speed of the MP3 decoder program. It was observed that enabling the cache before setting up the stereo ac97 codec and sending PCM samples to the ac97 resulted in no audio output. The reason for this conflict has not been determined or fully investigated.

It was also observed that no interrupts were necessary for the MP3 decoder system to operate. PCM samples could be sent to the ac97 and heard from an external speaker regardless of whether the ac97 playback and record interrupts were connected. In addition, interrupts were not needed for user control inputs while the decoding process was running. Instead, a quick check for any user input is performed at the beginning of the decoding while loop before decoding a frame and synthesizing PCM samples.

By configuring the Microblaze process to enable a barrel shifter, real-time playback was achieved with accurate (although not perfect) sound quality. It was also observed that enabling a hardware divider had no effect on sound quality or output speed. This was simply a trial-and-error procedure since enabling the cache to format the MAD decoder's fixed PCM output to a 16 bit sample results in no audio. (See MadFixedToSshort function in madlld_withoutOS.c file)

Profiling

A thorough assessment and evaluation of a system involves both a qualitative and quantitative analysis of several parameters, including execution time (speed or performance), memory usage, and power consumption. For this project, the dissipation of power does not need to be optimized and thus will not be investigated.

Using the profiling tool available in the Embedded Development Kit software platform environment, a breakdown of function call execution times, as well the percentage of time consumed by a particular function can be obtained. The results were generated in a text file in tabular form and inspected thoroughly before deducing information. The data was later placed into Microsoft Excel to approximate the percentage of time used for each procedure in the MP3 decoding process, as well as the user interface operations.

Profiling Procedures:

The MHS file (hardware settings) was modified to include a timer and interrupt controller, with default settings for the timer and making sure that the Microblaze interrupt was connected to the interrupt controller. As a result, a new bitstream had to be generated.

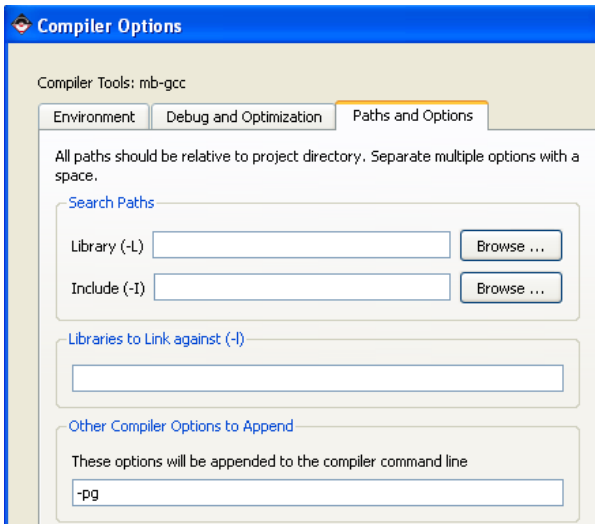
The MSS file had to be modified as well. However, it is simpler using the EDK interface.
Software => Software platform settings => OS and Lib configuration

Stand_alone

Enable_sw_intrusive_profiling => true

Profile_timer => xps_timer_0

Right-click on project => Set compiler options => Path and Options tab => type ***–pg*** for Other compiler options to append (library needed for profiling)



Re-compile (Software => Build all user applications)

Launch XMD
Connect mb mdm
Cd c:/mp3/mp3
Profile –config sampling_freq_hz 10000 binsize 4 profile_mem 0xBE000000
Dow executable.elf
Bps exit
Con
After program is finished type, ***Profile***
(gmon.out file should be generated)

Now open Xilinx Bash Shell
Project=>Launch Xilinx Bash Shell

xmd
cd mp3
mb-gprof executable.elf gmon.out > mp3_profile.txt

Open text file for tabulated results.

Profiling Results:

Table 38-1

Each sample counts as 0.0001 seconds.

% time	cumulative seconds	self seconds	self calls	self s/call	total s/call	name
41.93	5.52	5.52	1	5.52	8.03	play_song
30.75	9.57	4.05				XUartLite_SendByte
9.28	10.79	1.22	134784	0.00	0.00	XAC97_WriteFifo
3.44	11.25	0.45				L_start
3.27	11.68	0.43	117	0.00	0.01	synth_full
1.35	11.86	0.18	117	0.00	0.01	mad_layer_III
1.28	12.03	0.17	8424	0.00	0.00	dct32
1.27	12.19	0.17	10528	0.00	0.00	III_imdct_l
1.17	12.35	0.15	415	0.00	0.00	usleep
1.06	12.49	0.14				microblaze_invalidate_icache
0.67	12.57	0.09	11573	0.00	0.00	III_overlap
0.63	12.66	0.08	67889	0.00	0.00	mad_bit_read
0.51	12.72	0.07				XIo_EndianSwap16
0.48	12.79	0.06				XSysAce_RegRead32
0.43	12.84	0.06	21056	0.00	0.00	fastsdct
0.33	12.89	0.04	426	0.00	0.00	III_aliasreduce
0.28	12.92	0.04				L_done
0.18	12.95	0.02				L_start
0.15	12.97	0.02	16219	0.00	0.00	III_requantize
0.15	12.99	0.02				microblaze_enable_icache
0.14	13.01	0.02				memcpy
0.14	13.02	0.02				microblaze_disable_icache
0.13	13.04	0.02				XIntc_SetIntrSvcOption
0.11	13.06	0.02	1045	0.00	0.00	III_imdct_s
0.10	13.07	0.01				XSysAce_ReadDataBuffer
0.09	13.08	0.01				microblaze_enable_dcache
0.08	13.09	0.01	7488	0.00	0.00	III_freqinver
0.08	13.10	0.01				L_start
0.07	13.11	0.01				microblaze_flush_dcache
0.05	13.12	0.01				XSysAce_RegRead16
0.04	13.12	0.01				microblaze_invalidate_dcache
0.04	13.13	0.00				L_done
0.03	13.13	0.00	121	0.00	0.00	mad_header_decode
0.03	13.14	0.00				__udivsi3
0.03	13.14	0.00	121	0.00	0.01	mad_frame_decode
0.02	13.14	0.00	240	0.00	0.00	enable_cache
0.02	13.15	0.00	586	0.00	0.00	mad_bit_skip
0.02	13.15	0.00				microblaze_disable_dcache
0.01	13.15	0.00	367	0.00	0.00	GpioInputExample
0.01	13.15	0.00				__umodsi3
0.01	13.15	0.00				mad_bit_crc
0.01	13.16	0.00	1	0.00	0.00	XAC97_ClearFifos
0.01	13.16	0.00				XIo_EndianSwap32
0.01	13.16	0.00				XSysAce_RegWrite32
0.01	13.16	0.00	117	0.00	0.00	mad_timer_add
0.01	13.16	0.00				_program_init
0.01	13.16	0.00	117	0.00	0.01	mad_synth_frame
0.01	13.16	0.00	468	0.00	0.00	mad_bit_length
0.00	13.16	0.00				outbyte
0.00	13.16	0.00				is_buffered
0.00	13.16	0.00	234	0.00	0.00	disable_cache
0.00	13.16	0.00				xil_printf

0.00	13.16	0.00				_hw_exception_handler
0.00	13.16	0.00				microblaze_init_icache_range
0.00	13.16	0.00	118	0.00	0.00	decode_header
0.00	13.17	0.00	1	0.00	0.00	mad_frame_finish
0.00	13.17	0.00				L_done
0.00	13.17	0.00				outnum
0.00	13.17	0.00	357	0.00	0.00	mad_bit_init
0.00	13.17	0.00				_malloc_r
0.00	13.17	0.00				microblaze_init_dcache_range
0.00	13.17	0.00	15	0.00	0.00	WriteAC97Reg
0.00	13.17	0.00	1	0.00	0.00	mad_synth_init
0.00	13.17	0.00				XSysAce_ReadSector
0.00	13.17	0.00				L_done
0.00	13.17	0.00				bc_get_free_entry
0.00	13.17	0.00	122	0.00	0.00	volume_control
0.00	13.17	0.00	50	0.00	0.00	WriteInst
0.00	13.17	0.00	27	0.00	0.00	WriteData
0.00	13.17	0.00	4	0.00	0.00	InitInst
0.00	13.17	0.00	3	0.00	0.00	BstdRead
0.00	13.17	0.00	2	0.00	0.00	mad_stream_sync
0.00	13.17	0.00	1	0.00	0.00	mad_frame_mute
0.00	13.17	0.00				XGpio_DiscreteRead
0.00	13.17	0.00				XGpio_SetDataDirection
0.00	13.17	0.00				_vfprintf_r
0.00	13.17	0.00				end_len
0.00	13.17	0.00				get_partition_info
0.00	13.17	0.00				make_child_directory
0.00	13.17	0.00				malloc_wd
0.00	13.17	0.00				padding
0.00	13.17	0.00				read_from_file
0.00	13.17	0.00				update_bufcache
0.00	13.17	0.00	1	0.00	0.00	select_song
0.00	13.17	0.00				__divsi3
0.00	13.17	0.00				_crtinit
0.00	13.17	0.00	1	0.00	0.01	SetupAC97
0.00	13.17	0.00				XAC97_Delay
0.00	13.17	0.00	352	0.00	0.00	mad_bit_nextbyte
0.00	13.17	0.00	118	0.00	0.00	mad_timer_set
0.00	13.17	0.00	40	0.00	0.00	MoveCursorRight
0.00	13.17	0.00	27	0.00	0.00	LCDPrintChar
0.00	13.17	0.00	3	0.00	0.00	reduce_timer
0.00	13.17	0.00	2	0.00	0.00	BstdFileEofP
0.00	13.17	0.00	2	0.00	0.04	LCDSetLine
0.00	13.17	0.00	2	0.00	0.00	MoveCursorHome
0.00	13.17	0.00	2	0.00	0.00	mad_stream_buffer
0.00	13.17	0.00	2	0.00	0.00	reduce_rational
0.00	13.17	0.00	1	0.00	0.00	BstdFileDestroy
0.00	13.17	0.00	1	0.00	0.00	LCDClear
0.00	13.17	0.00	1	0.00	0.01	LCDInit
0.00	13.17	0.00	1	0.00	0.00	LCDOn
0.00	13.17	0.00	1	0.00	0.13	LCDPrintString
0.00	13.17	0.00	1	0.00	0.00	NewBstdFile
0.00	13.17	0.00	1	0.00	0.15	lcd_initialize
0.00	13.17	0.00	1	0.00	0.00	mad_frame_init
0.00	13.17	0.00	1	0.00	0.00	mad_header_init
0.00	13.17	0.00	1	0.00	0.00	mad_stream_finish
0.00	13.17	0.00	1	0.00	0.00	mad_stream_init
0.00	13.17	0.00	1	0.00	0.00	mad_synth_mute
0.00	13.17	0.00	1	0.00	0.00	mad_timer_abs
0.00	13.17	0.00	1	0.00	0.00	mad_timer_string
0.00	13.17	0.00	1	0.00	8.18	main
0.00	13.17	0.00	1	0.00	0.00	scale_rational

% time	the percentage of the total running time of the program used by this function.
cumulative seconds	a running sum of the number of seconds accounted for by this function and those listed above it.
Self seconds	the number of seconds accounted for by this function alone. This is the major sort for this listing. calls the number of times this function was invoked, if this function is profiled, else blank.
self ms/call	the average number of milliseconds spent in this function per call, if this function is profiled, else blank.
total ms/call	the average number of milliseconds spent in this function and its descendents per call, if this function is profiled, else blank.
name	the name of the function. This is the minor sort for this listing. The index shows the location of the function in the gprof listing. If the index is in parenthesis it shows where it would appear in the gprof listing if it were to be printed.

To make more sense of the profiling results, the data was placed into Microsoft Excel where the %time numbers were adjusted after removing the XUartLite_SendByte function execution percentage. This function is supposedly the second most time consuming operation, next to the play_song function, which involves the entire decoding process, including sending PCM samples to the stereo ac97 codec FIFO register. Unfortunately, after searching the entire project (.c and .header files, as well as in Xilinx-related folder), the XUartLite_SendByte function was no where to used or even initialized. For these reasons, I have made the assumption that this function has been called as a result of the profiling analysis, where the program has to be sampled using an interrupt timer. Afterwards, the %time numbers were added for related operations. For example, III_imdct_l, III_overlap, III_freq_inver, and III_imdct_s are all functions used for the IMDCT block of the MP3 decoding process. In the same manner, execution time percentages were calculated for AC97 codec operations, Subband synthesis filter bank, IMDCT, initial decoding stages, and peripheral (compact flash memory and LCD) operations. The results are illustrated in the Figure 40-1. For more details on these calculations, please refer to MP3_PROFILING_TABLE_CHART.xls on the provided data disc.

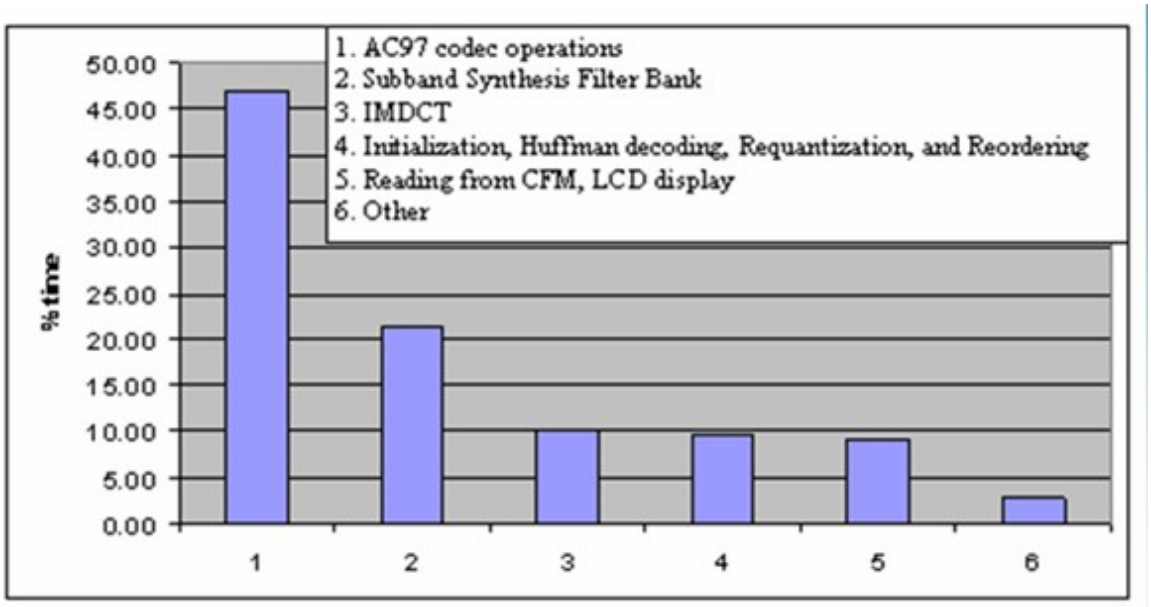


Figure 40-1. Profiling results of the FPGA based MP3 decoder system

Program Size (Memory Usage)

All sections of the code are placed in the 256 Mb external DDRAM memory
Vector sections of the .elf file (for software) are placed in 64 K BRAM. The definitions of these sections were obtained from the "Embedded System Tools Reference Guide," found in the EDK_Manuals folder of the provided data disc and tabulated in Table 41-1.

Table 41-1

Section	Description
.vectors.reset	Reset vector code.
.vectors.sw_exception	Software exception vector code.
.vectors.interrupt	Hardware Interrupt vector code.
.vectors.hw_exception	Hardware exception vector code.
.text	Program instructions from code in functions and global assembly statements.
.rodata	Read-only variables.
.sdata2	Small read-only static and global variables with initial values.
.data	Static and global variables with initial values. Initialized to zero by the boot code.
.sdata	Small static and global variables with initial values.
.sbss2	Small read-only static and global variables without initial values. Initialized to zero by boot code.
.sbss	Small static and global variable without initial values. Initialized to zero by the boot code.
.bss	Static and global variables without initial values. Initialized to zero by the boot code.
.heap	Section of memory defined for the heap.
.stack	Section of memory defined for the stack.

Section	Size (bytes)	Memory
.text	0x0001A7E4	DDR2_SDRAM_C_MPMC_B...
.rodata	0x0000BE3C	DDR2_SDRAM_C_MPMC_B...
.sdata2	0x00000000	DDR2_SDRAM_C_MPMC_B...
.sbss2	0x00000000	DDR2_SDRAM_C_MPMC_B...
.data	0x000006A4	DDR2_SDRAM_C_MPMC_B...
.sdata	0x00000004	DDR2_SDRAM_C_MPMC_B...
.sbss	0x00000000	DDR2_SDRAM_C_MPMC_B...
.bss	0x000B1534	DDR2_SDRAM_C_MPMC_B...

Boot and Vector Sections:

Section	Address	Memory
.vectors.reset	0x00000000	ilmb_cntlr_dlmb_cntlr
.vectors.sw_exc...	0x00000008	ilmb_cntlr_dlmb_cntlr
.vectors.interrupt	0x00000010	ilmb_cntlr_dlmb_cntlr
.vectors.hw_exc...	0x00000020	ilmb_cntlr_dlmb_cntlr

Section	Size (bytes)	Memory
Heap	0x50000	DDR2_SDRAM_C_MPMC_BASEADDR
Stack	0x50000	DDR2_SDRAM_C_MPMC_BASEADDR

Memory	Start Address	Length
ilmb_cntlr_dlmb_cntlr	0x00000000	64K
SRAM_C_MEM0_BASEADDR	0x20100000	1024K
DDR2_SDRAM_C_MPMC_BASEADDR	0xB0000000	262144K

The program size was calculated at several critical points in the project.
(Software => Get program size).

FPGA MP3 decoder system with cache hardware configuration

text	data	bss	dec	hex
185988	1620	8209596	8397204	802194

Dec=text+data+bss = number of bytes used for executable.elf file
Hex=dec # in hex

FPGA MP3 decoder system with cache and barrel shifter enabled n Microblaze

text	data	bss	dec	hex
157972	1580	8209596	8369148	7fb3fc

FPGA MP3 decoder system with added user-defined functions (FINALIZED DESIGN)

text	data	bss	dec	hex
157320	1724	1381688	1540732	17827c

Universal Compatibility Test:

The MAD Decoder can successfully decode MPEG Layer III files of various bit rates and sampling frequencies. The most common bit rates for MP3 encoding are 128 kbps, 256 kbps, and 320 kbps. The three allowed sampling frequencies are 32 kHz, 44.1 kHz (CD quality), and 48 kHz (DVD quality).

Backwards Compatibility Test:

The MAD Decoder can successfully decode MPEG Layer II files, as it is required for MP3 (MPEG Layer III) decoder should be able to do so. A MP2 file was placed onto the compact flash memory for test the compliance. In a similar manner, a .WAV file was tested and as expected, the MAD decoder could not recognize the format and decoding did not continue.

Patents:

Reference codes for existing MP3 decoders were obtained. The MP3 decoder reference design, known as MAD, requires no license and users are free to implement or modify them (see [8]).

Future Work:

Due to time and board constraints, several specifications have not been met. Firstly, although the system is able to decode MP3 files in real-time, a perfect reconstruction, that is, a very accurate sound quality has not been achieved. The background beats and instrumentation are high quality as well as the correct frequency. However, regardless of the true artist's voice, the voice seems to be at a slightly higher frequency, resembling a lower pitch than a fast forward (chipmunk sounding) voice. The exact reason for this problem has not been determined. However, a few possibilities have come to mind. First off, the Virtex 5 board runs with the Microblaze processor at roughly 100 MHz, about 3 times slower than the Virtex 2 board, with a hardware driven 300 MHz Power-PC processor. The processor used also

determines the method of the MAD decoder 32 bit fixed point calculation implementation, as defined in fixed.c of the FPGA MP3 player project. The programmers have allowed several methods, depending on a supported processor, including the ARM (the most common processor in the MP3 player industry), Intel, MIPS, PowerPC, and others. Secondly, the reason for no audio output with the cache being enabled before sending PCM samples to the ac97 codec FIFO has also not been determined. This may also be the reason for inaccurate processing.

In addition to perfecting the sound quality of the MP3 output, forward and rewind modes could also be designed. Due to time constraints, these operations have not even been fully investigated. A simple increment of the *FrameCount* variable failed to fulfill the forward function.

Finally, it would have been beneficial to load a project .ACE file, that combines the project hardware (.bit) and software (.elf) files onto the compact flash memory card, so that the MP3 decoder program can be loaded after booting the Xilinx Virtex 5 development board. This would enable the user to use the system without manually downloading the .bit and .elf files using the EDK software. Doing so would make a computer unnecessary to the design, and thus the system would become portable, provided that a power outlet is nearby.

An attempt to generate an .ACE file for the Xilinx Virtex 5 XUPV5 board proved that the ace generation procedure is not supported for this particular board. On the other hand, an ACE file was successfully generated for a Xilinx Virtex 5 ML505 board. The procedure and results for generating an ACE file for this project can be found in the MISC folder on the provided data disc (ace_file_generation_procedure_results.txt).

Conclusion:

In this project, a FPGA-based MP3 decoder has been implemented on the Xilinx Virtex 5 development board. It can read MP3 files from a compact flash memory device, then decode and play it through the AC97 codec. Different controls such as song selection, pause and stop modes are included.

Contents of data disc

Code

1. C source files
2. Header files

Datasheets.

- U1 Virtex®-5 Family Overview
- U1 Virtex-5 Data Sheet
- U2 System ACE™ CF Controller
- In-Box Tianma LCD Display Module
- U22 Analog Devices AC '97 Codec
- SW2 Edge Drive Jog Encoder with Switch (Volume controller)
- SW3, SW6, SW8 8 Position DIP Switch for GPIO, SMT
- P1 CompactFlash Card Header

EDK manuals

1. Embedded system tools reference guide.pdf
2. Adding IP cores to EDK project.pdf
3. OS and libraries document collection.pdf
4. Profiling procedures in EDK and SDK.pdf

Hardware files

1. Mhs, mss, and ucf files
2. Peripheral drivers (.vhdl files)
    a. lcd_ip_v1_00_a (LCD)
    b. opb_ac97_controller_ref_v1_00_a (Stereo ac97 codec)

Miscellaneous

1. MP3_decoder_lines_of_code.xls
2. MP3_numbers.xls
3. MAD_decoder_function_calls.rtf
4. ace_file_generation_procedure_results.rtf

MP3 literature

1. ISO_IEC_11172-3.pdf (ISO standard document for MP3 encoding and decoding)
2. A hardware accelerated MP3 decoder.pdf
3. Design and implementation of MP3.pdf
4. Design of the audio coding standards for MPEG and AC-3.pdf
5. Psychoacoustic models and non-linear human hearing.pdf
6. The theory behind MP3.pdf

Profiling

1. MP3_profile_cache_OFF_barrel_shifter.txt
2. MP3_profile_cache_ON_barrel_shifter.txt
3. MP3_PROFILING_TABLE_CHART.xls
4. profile_procedures.rtf

Project deliverables

1. Project report
2. Presentation powerpoint
3. Electronic demonstration poster
4. Project proposal (First semester)

Virtex 2 designs

1. BIST_REV_HW_SW_GAME_AUDIO_WORKS_digital_loopback.zip
2. MP3_virtex2_MAD_decoder_program.zip

Virtex 5 designs

1. ac97_test_recording_loopback.zip
2. FPGAMP3PLAYER.zip (finalized MP3 decoder system)
3. xupv5_bsb_design_LCD.zip

Other MP3 decoder reference designs

1. __mp32pcm_16bit_source

References

[1] A. Abdel-Gawad, "A *full hardware implementation for an MP3 decoder chip using VHDL*", Project report, University of California at Santa Barbara, December 2008.

[2] M. Botteck, H. Blume, J. von Livonius, M. Neuenhahn, and T. Noll, "*Programmable architectures for real-time music decompression*", Proceedings of parallel computing: architectures, algorithms, and applications conference, vol. 38, pp. 777-784, September 2007.

[3] K. Brandenburg, and H. Popp, "*An introduction to MPEG layer-3*", European Broadcasting Union Technical Review, pp. 1-15, June 2000.

[4] P. Chandraiah, and R. D¨omer , "*Specification and design of a MP3 audio decoder*", Technical Report CECS-05-04 , University of California at Irvine, pp. 1-83, May 2005.

[5] International Standard ISO/IEC 11172-3. "*Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s – Part3: Audio,"* January 1999.

[6] W. Jiang, S. Polisetty, and X. Li, "Huffman decoder", Project report, University of Tennessee at Knoxville, May 2003.

[7] G. Liaskos, "MP3 file structure", Internet resource: http://www.multiweb.cz/twoinches/MP3inside.htm#MP3FileStructure

[8] R. Leslie, "MAD: MPEG audio decoder" Internet resource: http://www.mars.org/home/rob/proj/mpeg/

[9] M. Hipp and O. Fromme. "MPG123" Internet resource: http://www-ti.informatik.uni-tuebingen.de/~hippm/mpg123.html

[10] R. Raissi. "The theory behind MP3." December 2002.

[11] K. Salomonsen, "Design and implementation of an MPEG/Audio layer III bitstream processor", Master's thesis, Aalborg University, Denmark, 1997