SmartFusion cSoC: Implementation of FLAC Player Using Hardware and Software Partitioning

Table of Contents

Introduction

The SmartFusion® customizable system-on-chip (cSoC) integrates FPGA technology with a hardened ARM® Cortex™-M3 based microcontroller subsystem (MSS) and programmable high-performance analog blocks built on a low power flash semiconductor process. The MSS consists of hardened blocks, such as a 100 MHz ARM Cortex-M3 processor, peripheral DMA (PDMA), embedded nonvolatile memory (eNVM), embedded SRAM (eSRAM), embedded FlashROM (eFROM), external memory controller (EMC), watchdog timer, the Philips Inter-Integrated Circuit (I^2C), serial peripheral interface (SPI), 10/100 Ethernet controller, real-time counter (RTC), GPIO block, fabric interface controller (FIC), in-application programming (IAP), and system registers. The programmable analog block contains the analog compute engine (ACE) and analog front-end (AFE), consisting of ADCs, DACs, active bipolar prescalers (ABPS), comparators, current monitors, and temperature monitors.

Audio compression and decompression is commonly known as audio codec. It is used in almost all multimedia applications. It contains computational intensive algorithms to compress and decompress the audio signal. SmartFusion cSoCs provide the flexibility to offload a portion of digital audio processing to the FPGA fabric to boost overall performance. The parallel processing of digital audio signal in the FPGA fabric helps to free up the ARM Cortex-M3 processor.

The intent of this application note is to demonstrate the capability of SmartFusion cSoCs for multimedia audio applications. This application note explains:

1. Porting of a free lossless audio codec (FLAC) audio decoder on a SmartFusion cSoC; decoding and playing encoded *.flac file in real time using in-built DAC.
2. The usage of FPGA fabric to implement portion of FLAC DSP algorithms to improve the decoding performance.

The design examples attached with this application note can be used as a reference when you need to implement a FLAC decoder on SmartFusion cSoC device to decode and play the encoded audio file.

A basic understanding of SmartFusion cSoC design flow is assumed.

Refer to the *SmartFusion Microcontroller Subsystem User's Guide, Using UART with a SmartFusion cSoC - Libero SoC and SoftConsole Flow Tutorial*, and *Libero SoC v10.0 User's Guide* to understand the SmartFusion design flow.

SmartFusion cSoC for Audio Applications

SmartFusion cSoCs are the only devices that integrate an FPGA fabric, 32-bit ARM Cortex-M3 processor based MSS and programmable analog. These unique features make SmartFusion cSoCs an ideal choice for low cost, low power, and high performance audio applications.

Figure 1 illustrates high-level audio codec application processing using an ARM Cortex-M3 processor and FPGA fabric on a SmartFusion cSoC device.

A codec application running on a Cortex-M3 processor compresses (encodes) and decompresses (decodes) the incoming samples. The decoded samples are passed to the DAC to reconstruct the audio signal. The output of the DAC is fed to a speaker system to play back the audio signal.

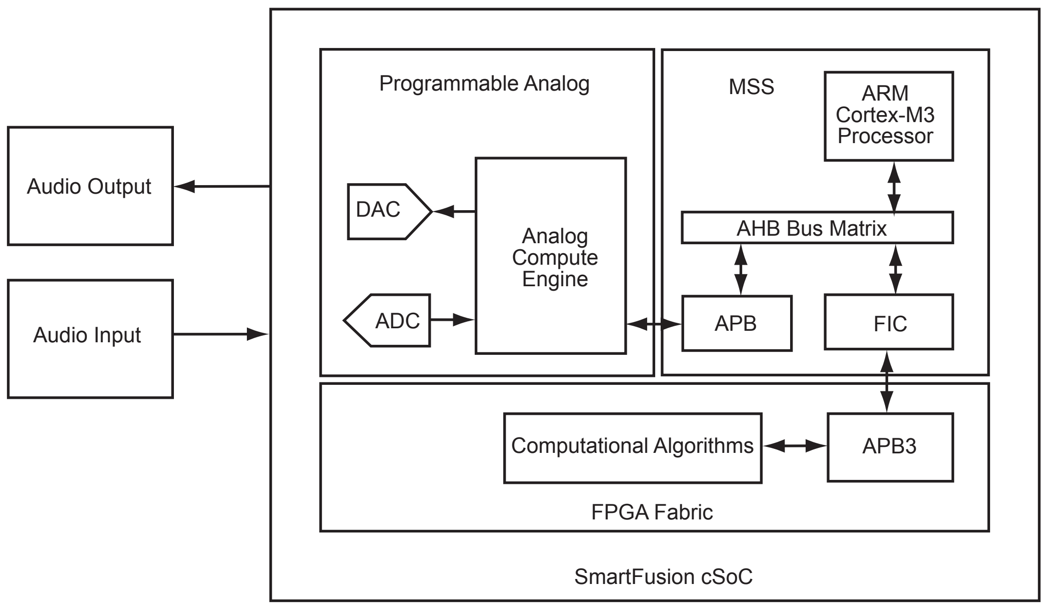

***Figure 1* • Audio Codec Application Processing on SmartFusion cSoC Device**

Design Example Overview

This design example explains the FLAC audio decoder design implementation on the SmartFusion Development Kit Board with the following objectives:

- Decoding an encoded *.flac file and playing the decoded samples in real time using the built-in DAC.
- Implementing a portion of FLAC decoder algorithms in the FPGA fabric to improve the speed of the FLAC decoder.

This FLAC decoder design example supports sample rates from 8 kHz to 44.1 KHz and a word length of 16 bits.

Figure 2 shows the high-level decoder design implementation on the SmartFusion Development Kit Board.

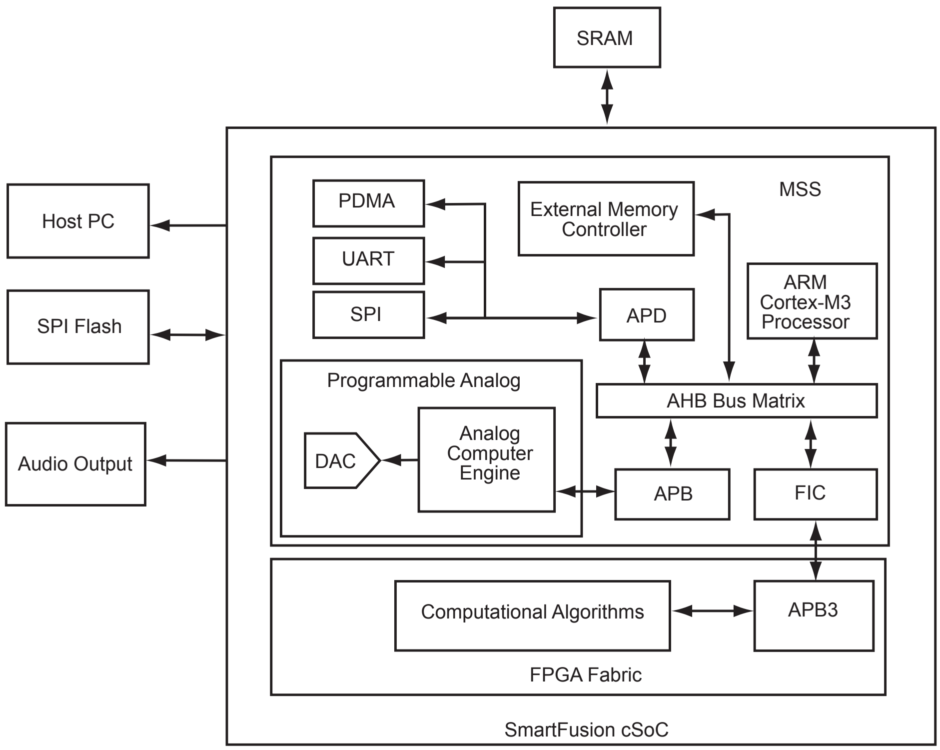

***Figure 2* • Decoder Design Implementation on SmartFusion cSoC Device**

The FLAC decoder reads the stored encoded *.flac file from SPI flash and starts decoding the samples. The encoded *.flac file is preloaded from the host PC to SPI flash using the UART interface of the SmartFusion cSoC. The control switches between the Cortex-M3 processor and FPGA fabric as a portion of the decoder algorithms are moved to FPGA fabric to improve the performance of the decoder. The decoded samples are passed to the built-in DAC to reconstruct the audio signal. A speaker system is connected to the DAC output pins on the SmartFusion Development Kit Board to play the audio signal.

Design Example Internal Details

Introduction to FLAC

FLAC is a lossless audio compression codec primarily authored by Josh Coalson. A PCM sample audio data file compressed by FLAC can be decompressed into an identical copy of the original audio data file. Audio sources encoded by FLAC are typically reduced to 50–60% of their original size.

The audio input to the FLAC encoder is passed block by block to its prediction stage where the encoder tries to find a mathematical description (usually an approximate one) of the signal. This description is typically much smaller than the raw signal itself. Since the methods of prediction are known to both the encoder and decoder, only the parameters of the predictor need be included in the compressed stream. FLAC currently uses four different classes of predictors to approximate the audio signal based on its content. If the predictor does not describe the signal exactly, the difference between the original signal and the predicted signal (called the error or residual signal) must be coded losslessly. If the predictor is effective, the residual signal will require fewer bits per sample than the original signal. The FLAC encoder currently uses Rice coding to code the error signal (residual signal). Each compressed block becomes a frame in the compressed format.

An encoded frame is written on the FLAC output file for each block of the input file. The frame starts with a header and is followed by a number of sub-frames. Each sub-frame starts with its own header, followed by Rice codes for encoded audio samples from the same channel. A frame consists of one sub-frame for each audio channel, and each sub-frame consists of the same number of Rice encoded audio samples.

The FLAC codec is an open and royalty-free format with a free software implementation made available. For more information, refer to http://flac.sourceforge.net/.

FLAC Decoder

The FLAC decoder decompresses the compressed FLAC source file into original audio source without any loss in quality.

An encoded FLAC output stream starts with the four-byte identifying string fLaC. This is followed by the STREAMINFO metadata block, including all the needed side information for the decoder and for the end-user. The remainder of the stream consists of encoded audio frames.

The FLAC decoder consists primarily of two functional blocks responsible for performing tasks on different types of information within the FLAC stream received from the metadata decoder. These blocks are the stream decoder, and the frame decoder. Figure 3 illustrates the complete FLAC audio player architecture.

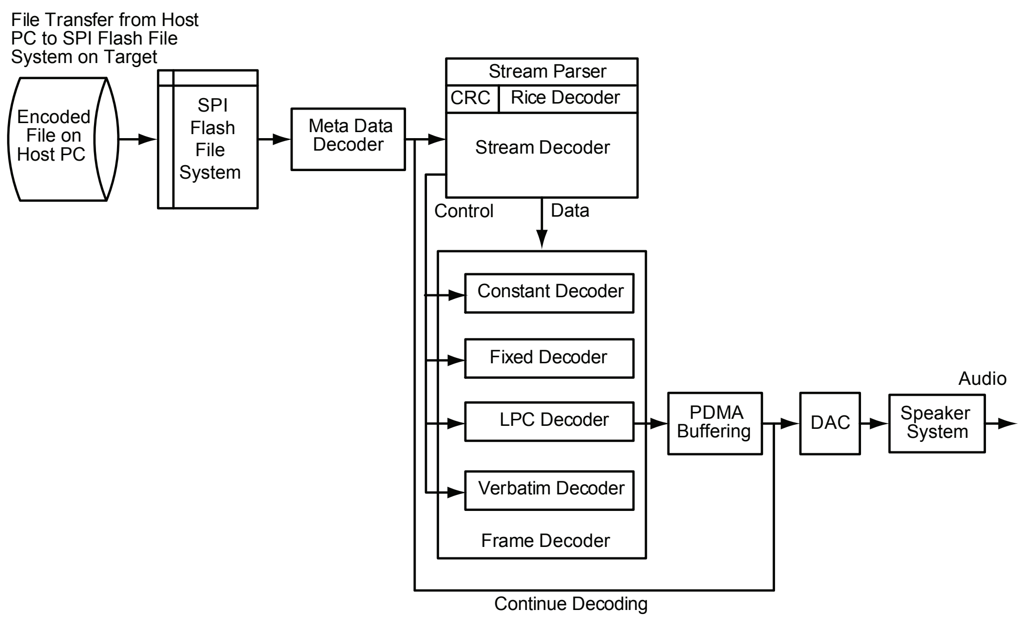

***Figure 3* • FLAC Audio Player Architecture**

The FLAC decoder system software, first reads the encoded FLAC file (*.flac) from the host PC and stores in SPI flash on target board. Then it decodes the *.flac file and plays back the audio. The following functional blocks are involved in decoding and playing the audio.

- Metadata decoder
- Stream decoder
- Frame decoder
- PDMA buffering
- DAC

Metadata Decoder

The metadata decoder is a software function responsible for decoding FLAC metadata frames. These are the frames that are not essential for audio playback as they do not contain any encoded audio sub-frames.

The first mandatory metadata block is a STREAMINFO block that contains information about the entire output stream. This block is followed by the 32-bit FLAC stream marker at the start of each new stream.

The metadata blocks are followed by the frames where each frame starts with a header. Thus, this block is responsible for parsing the entire stream information which helps the decoder to manage its buffers.

Stream Decoder

The stream decoder is responsible for reading the frame header and footer information and directing the audio frame to the correct frame decoder unit.

The frame header contains information such as the frame size, sampling rate, and an 8-bit CRC.

Encoded audio data follows the frame header, succeeded by a frame footer, which contains a 16-bit CRC. The 8-bit CRC is used to determine header validity, whereas the 16-bit CRC is used to the check the whole frame’s validity.

A valid frame header begins with a 14-bit sync code, 11111111111110, as an indication of the start of a frame. Unfortunately, the Rice codes inside a frame feature an arbitrary bit pattern that may look like a sync code. Thus, when the decoder finds the bit pattern of a sync, it has to read the rest of the header and verify its CRC to make sure that this is a frame header.

Stream Parsing

Following the detection of a sync code, the stream decoder extracts the important information for the frame from the frame header for future use to play back when the frame is ready. This frame information will be buffered for playback in the same way that the frame will be buffered for output.

After the frame header, a sub-frame type code follows that is part of the sub-frame header information in the stream. It causes the stream decoder to forward the sub-frame’s contents to the appropriate frame decoder.

Cyclic Redundancy Check (CRC)

The error detection within the FLAC audio stream is handled on a frame-by-frame basis through the use of cyclic redundancy check (CRC) checksums. The header of each audio frame is protected by a CRC-8 checksum, while the frame itself is protected using a CRC-16 checksum. If an error is detected in CRC calculations, decoding of the frame is abandoned, and the decoder attempts to synchronize on a new frame sync code.

Rice Decoder

FLAC uses the Rice coding scheme to encode the residual (error) signal to achieve high compression ratios. The residual signal is the difference between the actual input and the model of the signal generated by the predictor. The Rice coding scheme currently implemented is a form of run-length encoding known as partitioned Rice encoding.

Frame Decoder

The Frame decoder functional block consists of four sub-blocks that implement the four types of decoding in the FLAC subset specification. These decoding methodologies are as follows:

- LPC decoding
- Fixed decoding
- Constant decoding
- Verbatim decoding

LPC Decoder

Restoration of a signal that has been encoded using an n^{th} order linear predictor involves the implementation of an n^{th} order finite impulse response (FIR) digital filter. Such a filter may be implemented in hardware using a series of multipliers and accumulators.

Fixed Decoder

The fixed decoder restores a signal using a fixed FIR digital filter, a predictor of order between 1 and 4. This filter can be implemented in hardware using a series of shift registers, adders, and subtracters to compute the appropriate mathematical operation to decode the data frames.

Constant Decoder

Constant encoding is used for encoding silence in audio tracks wherein all the samples in an entire block size consists of the same value. As a result, the encoded sub-frame contains only one sample that will be output n times, where n corresponds to the frame's block size.

Verbatim Decoder

Verbatim signals have zero compression, and therefore a verbatim signal is the same as the raw signal. The verbatim decoder needs only to output each of the signal's samples.

For more information on FLAC decoder blocks, refer to http://flac.sourceforge.net/ and https://ece.uwaterloo.ca/~cmestewa/Publications/dd.pdf

PDMA Buffering

A PDMA controller is used to buffer the data from a decoded output buffer to another play buffer. This allows the decoder to continue decoding the encoded samples and the player block in hardware takes care of passing the samples to the DAC from this play buffer. A synchronization mechanism is used in allowing the decoder to decode the samples without overwriting any of the earlier decoded samples and to pass the samples from the other buffer (play buffer) to the DAC, based on the sampling rate.

DAC

The ACE DAC is used to reconstruct the audio signal from the decoded frame to play the audio on speaker system.

Refer to the *SmartFusion Programmable Analog User's Guide* for more information on the ACE DAC.

Figure 4 shows the complete flow of the FLAC decoder in decoding and playing the encoded *.flac file.

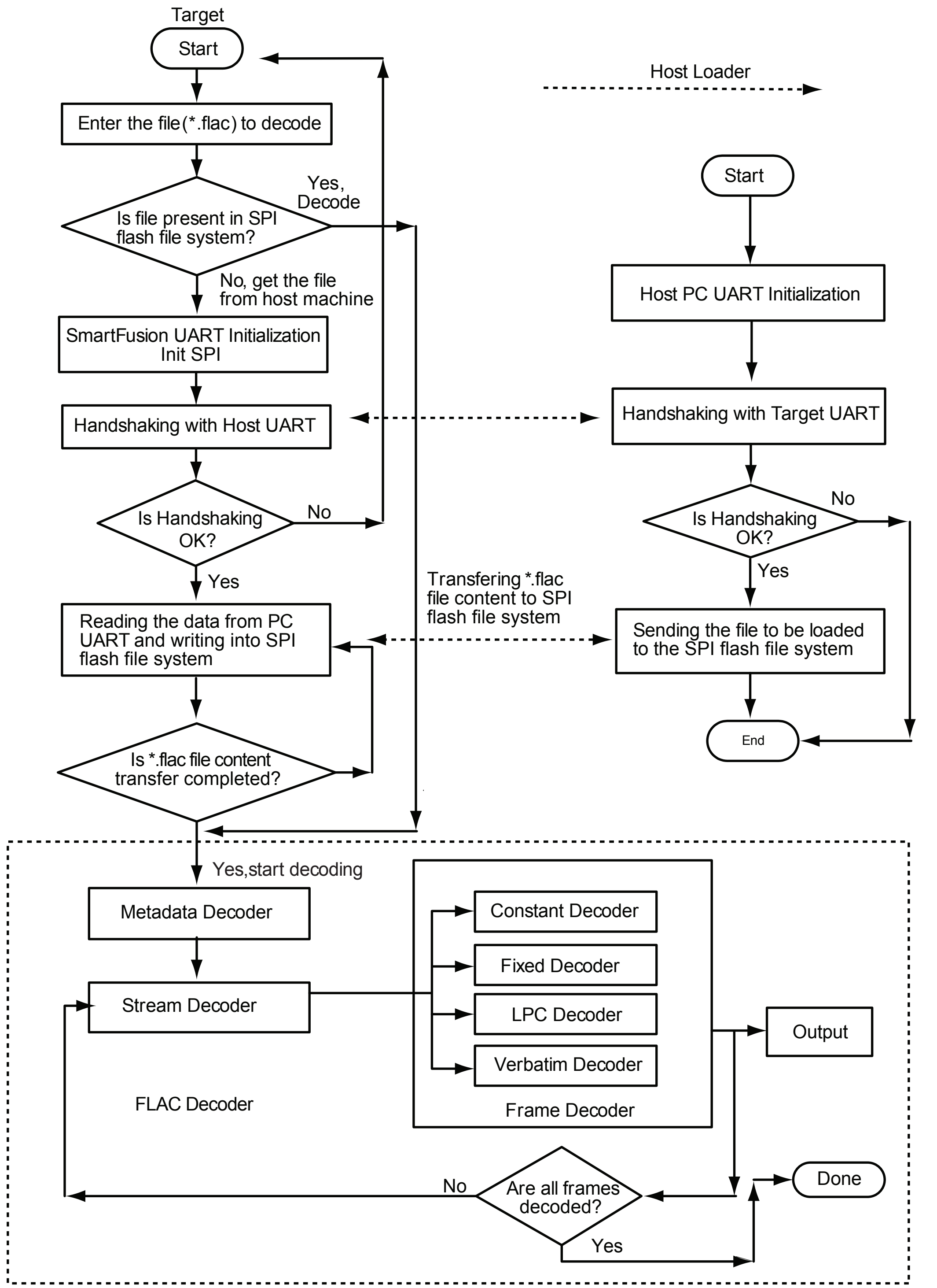

***Figure 4* • Complete FLAC Decoder Flow**

Porting and Optimizing the FLAC Decoder on a SmartFusion cSoC

Porting the FLAC decoder on a SmartFusion cSoC requires mounting a file system on SPI flash to store the encoded *.flac files, which helps the decoder operating on files. In this design example, the FatFs file system is ported on SPI flash to store the encoded files from the host PC onto SPI flash to operate the decoder on file operations. Refer to the *Implementation of FatFs on Serial Flash* application note on how to port the file system onto the SmartFusion cSoC device.

The FLAC decoder can decode and play 11 KHz *.flac files without performing any optimizations. To play higher sampling rate audio files, the performance of the decoder needs to improve by intelligently partitioning the code between the Cortex-M3 processor and FPGA fabric on the SmartFusion cSoC.

Software and Hardware Partitioning

The pure software implementation of the decoder is unable to decode the higher sampling rate audio samples in real time. The decoder performance can be improved by hardware and software partitioning of computational and noncomputational modules.

Out of the four sub-frame decoding methods, LPC and fixed decoding blocks are the most computationally intensive tasks. The Rice decoder is also a heavily called function that involves a great deal of unary logic.

The Cortex-M3 processor on a SmartFusion cSoC is used for noncomputational intensive modules such as frame header parsing, CRC computation, and buffer management. The FPGA fabric on the SmartFusion cSoC device is used for computational intensive modules such as LPC and Rice decoding. This improves the decoder performance and enables it to play higher sampling rate audio samples.

The designer has to do profiling of various functional blocks of the decoder to identify the highest processing time taken by critical functional blocks. The profiling summary is shown in Table 1 for decoding 44.1 KHz sample rate audio files. The profiling measurements—time taken for a particular functional block—are measured using Timer2, which is a part of the SmartFusion MSS.

***Table 1* • Profiling Summary**

Function Name	Time (seconds)
Metadata decoding	0.001
Stream decoder	0.00004
Rice decoding	0.085
LPC decoding	0.014
Constant decoding	0.000
Verbatim decoding	0.000
Fixed decoding	0.005
CRC -16	0.000004
Endian conversion	0.000023

Based on the profiling summary, the functional blocks that take most of the processing time are Rice decoding and LPC decoding. The provided design files implement VHDL code for these two functional blocks in FPGA fabric.

Table 2 shows profiling with computational blocks in the FPGA fabric. This shows reduction of processing time compared to the Table 1 profiling summary. In this way, offloading computational blocks improves the overall performance of the decoder to play 44.1 KHz samples in real time. Table 2 provides the profiling summary after implementing decoders in the FPGA fabric.

***Table 2* • Profiling Summary After Implementing Decoders in FPGA Fabric**

Function Name	Time (Seconds)
Rice decoding	0.065
LPC decoding	0.0049

Figure 5 shows the high-level FLAC decoder architecture with the partitioning of computational modules and non-computational modules into software and hardware. The colored modules are implemented in the FPGA fabric and the rest of the modules are in software that is running on Cortex-M3 processor.

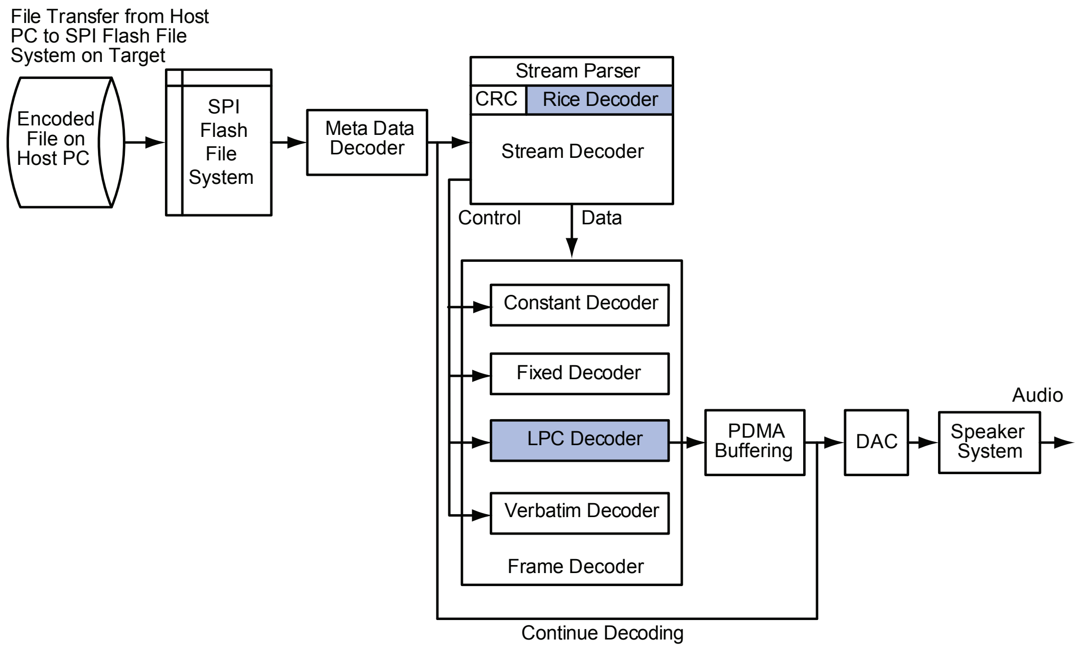

***Figure 5* • Software and Hardware Partitioning of FLAC Decoder**

Hardware Implementation

The hardware implementation involves configuring the MSS and designing the computational blocks and player block that go into the FPGA fabric.

The design example consists of the MSS, decoder block, and the player block. The decoder block includes the Rice decoder and eighth order LPC decoder with (see Figure 11 on page 15) with custom APB3 interface.

The MSS is configured with FIC, clock conditioning circuit (CCC), GPIOs, ACE, EMC, SPI, and UART. The CCC generates an 80 MHz clock, which acts as the clock source. The FAB_CLK is selected as a 40 MHz clock. The FIC is configured to use the master and slave interface with an AMBA APB3 interface. Six GPIOs in the MSS are used in this design. Three GPIOs are configured as outputs and routed through the FPGA fabric. They are used to handle the decoder scheme and player to run on hardware. The remaining three GPIOs are configured as inputs and routed through the FPGA fabric to interrupt the processor when the corresponding operation is completed. The ACE is used and the sigma-delta DAC is configured to produce an analog signal that can be fed to speakers. The EMC is configured for Region 0 as Asynchronous RAM and port size as halfword. Region 1 is configured as NOR flash and the port size as halfword. SPI_1 is configured for writing an audio file into flash memory.

The UART_0 is configured for interfacing with the user through HyperTerminal. The PDMA is used for buffer management between the decoder buffer and play buffer.

Figure 6 illustrates the FLAC player hardware design in SmartDesign. The FLAC_HW block consists of the MSS_FLAC_PLAYER block and FLAC_DECODER block, which has a Rice decoder sub-block, LPC decoder sub-block, and player sub-block. The signals start1 and start2 are driven by the Cortex-M3 processor to enable the decoder logics, Rice and LPC. The signal start3 is also driven by the Cortex-M3 processor to enable the player logic. The stop1, stop2, and stop3 signals are monitored by the Cortex-M3 processor to track the decoder and player status in the fabric.

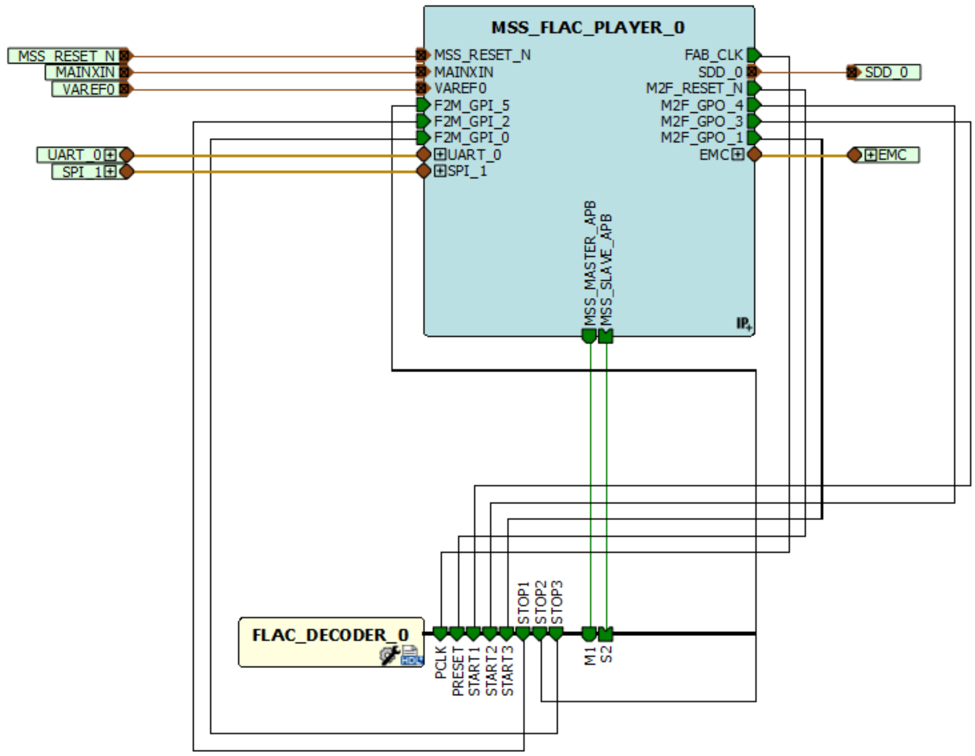

***Figure 6* • FLAC Player Smart Design**

Figure 7 shows the graphical view of the FLAC_HW block in Synplify HDL-Analyst RTL-Viewer. It shows how the Rice decoder, LPC decoder, and player are combined. The start signals are used to select the decoder block and connect the custom APB master of the selected decoder block with the FIC slave.

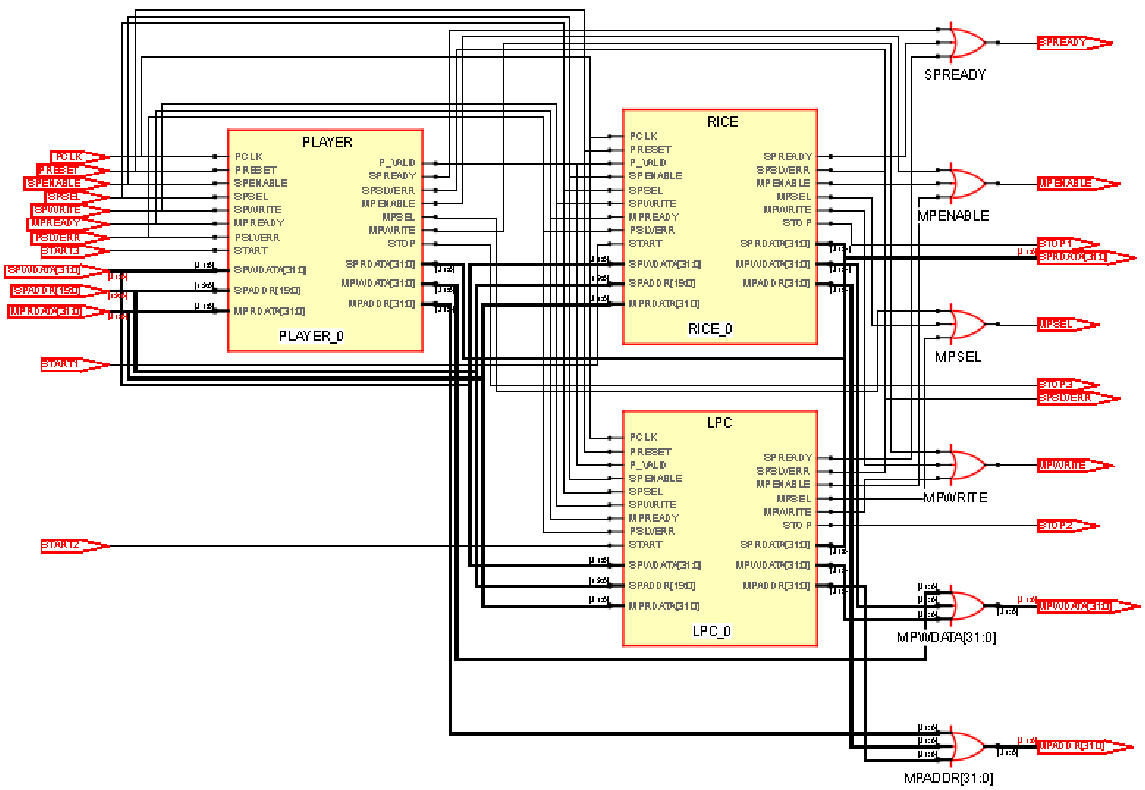

***Figure 7* • RTL View of FLAC_HW**

RICE Decoder

The Rice decoder in the FPGA fabric is called by the software that is running on the Cortex-M3 processor. To make a call to the Rice decoder, the Cortex-M3 processor sets the GPIO corresponding to the Rice decoder block and then using the APB master of the FIC, the parameters that are required to decode the frame are passed to the decoder block. The logic in the FPGA fabric reads the parameters using the custom APB slave.

The custom APB master reads the encoded samples from the external SRAM and also writes the decoded samples into external SRAM. Also, the APB master returns the decoding status to the software that is running on the Cortex-M3 processor.

Figure 8 shows the Rice decoder state machine along with the APB master and slave.

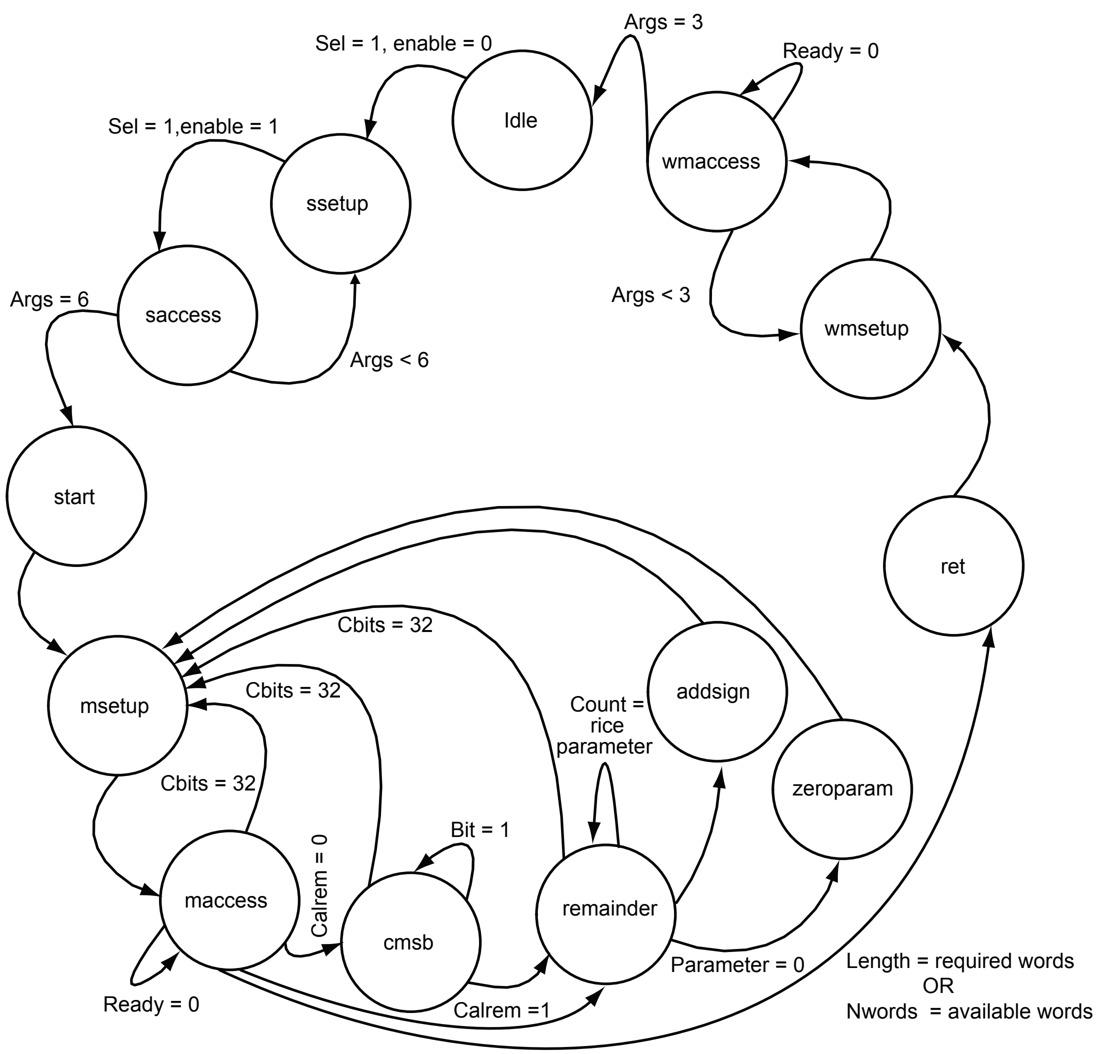

***Figure 8* • Rice Decoder State Machine**

Steps to Implement Rice Decoder in FPGA Fabric

A state machine is designed that comprises of a master and slave for receiving parameters, reading, and decoding. The following steps are performed in the state machine:

Step 1

Develop a slave interface logic to the Cortex-M3 processor for receiving parameters such as address of the residual, address of a buffer for writing return values, Rice parameter, number of words available, number of samples required to decode, consumed bits and address of status.

This slave interface logic is implemented in the FPGA fabric with different states such as Idle, ssetup, and saccess in reading the parameters from the Cortex-M3 processor through the FIC. Refer to *Connecting User Logic to the SmartFusion Microcontroller Subsystem* application note for more details on how to connect the user logic in the FPGA fabric to the microcontroller subsystem.

Step 2

The master logic APB is developed in the FPGA fabric to perform the following tasks using the states msetup and maccess.

1. Read the bitstream from the residual address.
2. Write the decoded values into return buffer.

Step 3

The states cmsb (count MSB bits), remainder, addsign, and zeroparam are used to decode the residual bitstream. The decoded sample is written into return buffer using master logic APB, as mentioned in Step 2.

Step 4

The states ret, wmsetup, and wmaccess are used to indicate the decoding status, such as how many words are decoded, consumed words, and bits used for decoding. A stop2 is driven High to indicate the completion of the decoding.

Figure 9 shows how a hardware functional block is called by the software that is running on the Cortex-M3 processor.

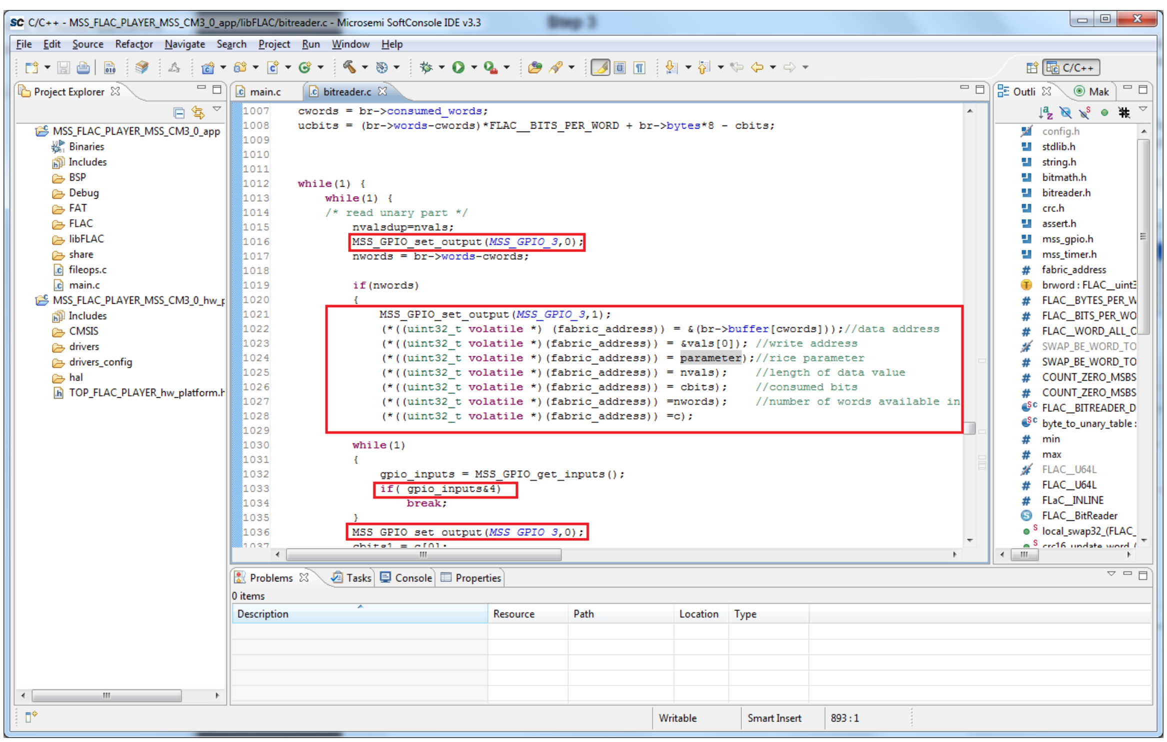

***Figure 9* • Rice Decoder Hardware Function Call Between Cortex-M3 Processor and FPGA Fabric**

LPC Decoder

The linear prediction coding (LPC) decoder contains most of the computational tasks such as multiply and accumulate. Most of the FLAC encoders use eighth order LPC, which requires eight 16-bit × 14-bit multipliers and seven 32-bit adders. Running them on the Cortex-M3 processor consumes much processing time to perform many arithmetic computations. To accelerate the decoding process, the LPC decoder is implemented in FPGA fabric. To make a call to the LPC decoder, the Cortex-M3 processor sets the GPIO corresponding to the LPC decoder block and then using the APB master of the FIC, the parameters that are required to decode the frame are passed to the decoder block. The logic in the FPGA fabric reads the parameters using a custom APB slave.

After reading the parameters, the decoding process starts by reading warm-up samples and coefficients of the corresponding frame from external SRAM with the help of the APB master in the fabric, which is connected to the FIC slave. After decoding each sample, the master writes this sample into external SRAM and reads the next residue for the next sample calculation.

Figure 10 shows the LPC decoder state machine along with the APB master and slave.

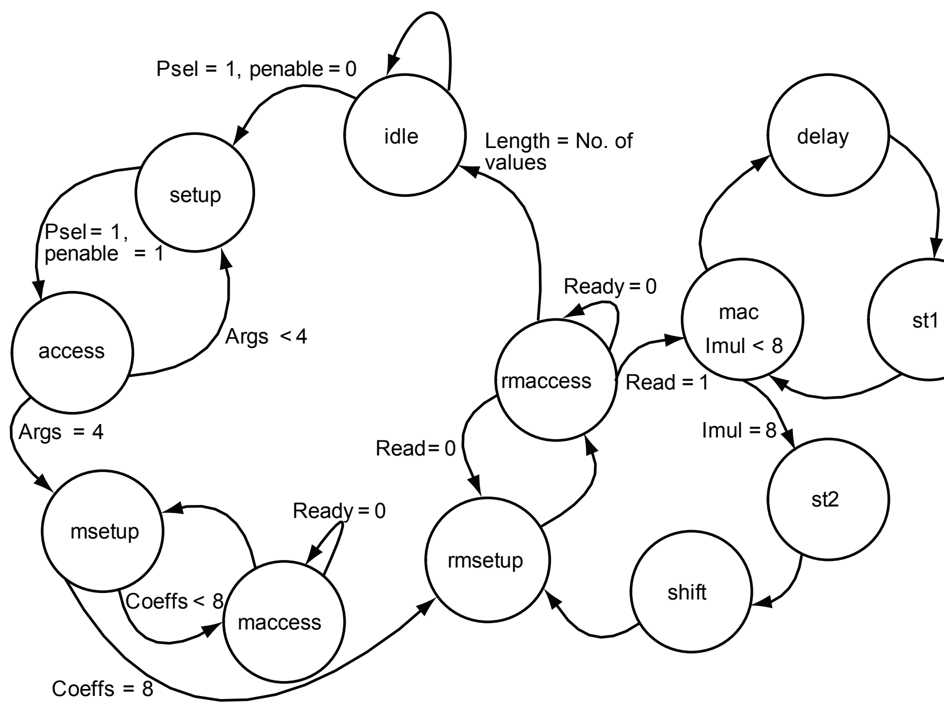

***Figure 10* • LPC Decoder State Machine**

EQ 1 is used to calculate the samples.

$$Sample_i = \frac{\sum_{j=0}^{order-1} \text{QLP Coefficent}_j \times Sample_{i-j-1}}{2^{\text{QLP shift needed}}} + Residual_i$$

EQ 1

The eighth order LPC architecture is shown in Figure 11.

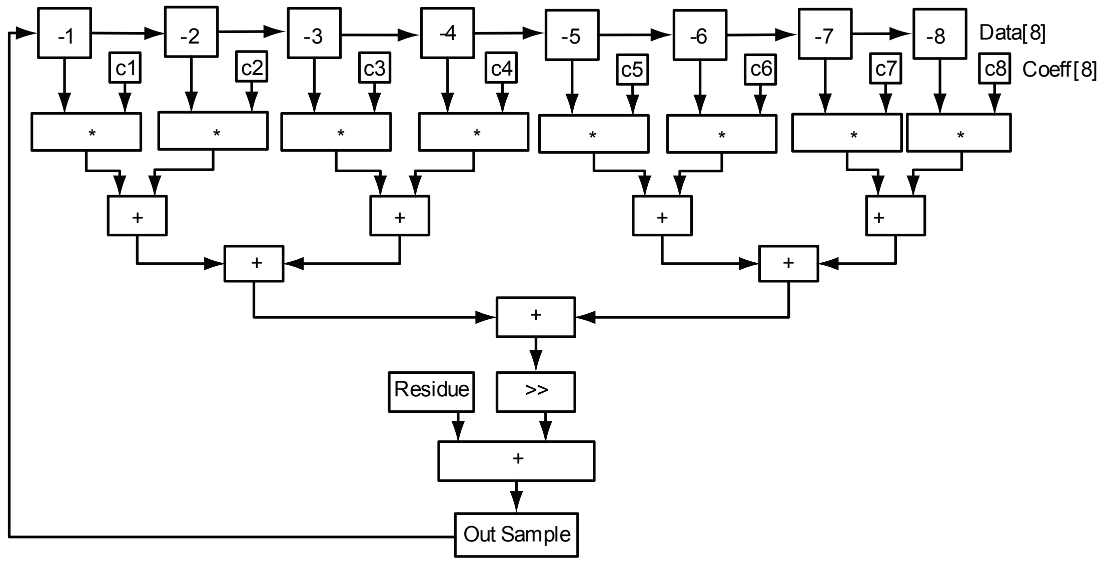

Figure 11 • **LPC (Eighth Order) Decoder Architecture**

Steps to Implement LPC in the FPGA Fabric

A state machine is designed which comprises a master and slave for receiving parameters, reading, and decoding.

The state machines are explained in a sequence as shown below:

Step 1

Develop slave interface logic to the Cortex-M3 processor for receiving parameters such as address of warm-up samples, address of coefficients, address of the residual, LPC shift, and length of the frame.

This slave interface logic is implemented in the FPGA fabric with different states such as Idle, ssetup, and saccess in reading the parameters from the Cortex-M3 processor through the FIC.

Step 2

The provided design files implemented the LPC decoding with eighth order, which actually requires eight coefficients and eight warm-up samples. The parameters received in Step 1 contain the addresses for these coefficients and samples. The master logic APB is developed in the FPGA fabric using states msetup and maccess to read the coefficients and warm-up samples from the external SRAM memory.

Step 3

The LPC filter contains multiplication and accumulation. The multiplication is implemented by using the advanced booth's multiplier macro. The multiplier operation is done sequentially for satisfying minimum area and timing. The multiply and accumulation operation is performed using the states mac (multiply and accumulation), delay, and st1.

Step 4

Apart from multiplication and accumulation, the LPC filter also contains shifting and residue addition operation. To perform this, the following things are implemented in the design:

1. The LPC filter contains shifting operations; barrel shifter is designed to perform the shifting operation. The state st2 is used to perform this shifting operation.
2. The residue value reads from the memory to add this to the shifted value. This read operation is performed using the states rmsetup and rmacess.
3. Finally the shifted value is added to the residue value using state st2.

Step 5

The decoded samples are written to the return buffer—the memory location of the residual. The states rmsetup and rmacess are used to perform the writing operation.

Step 6

Steps 3, 4, and 5 are repeated until all the samples of the frame are decoded. The Stop1 signal is driven High when decoding is completed.

Player Block

The player block takes the decoded audio sample from playbuffer and gives it to the DAC at the sampling rate. The Cortex-M3 processor sets the corresponding GPIO to enable the player in the fabric and then passes the play buffer address and sampling rate to the player block.

Figure 12 shows the player state machine along with the APB master and slave.

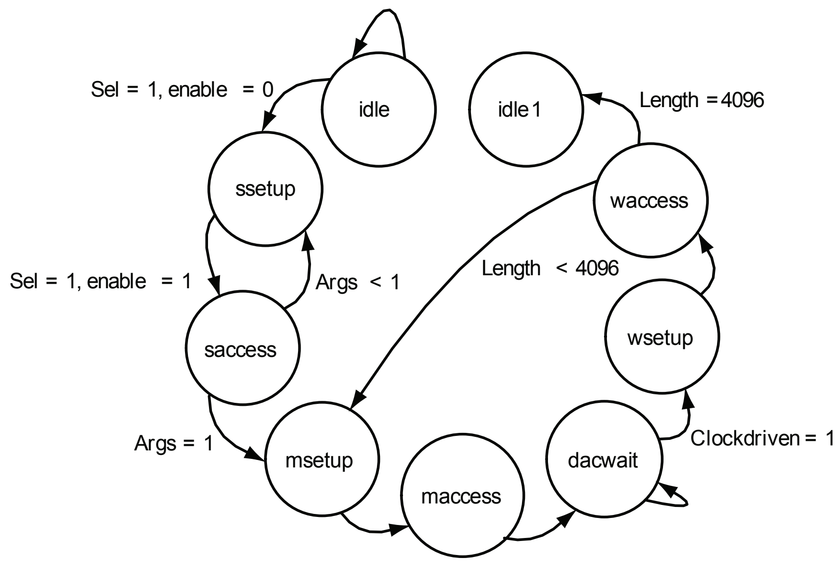

Figure 12 • **Player State Machine**

Steps to Implement the Player Block in the FPGA Fabric

The state machines are explained in the following sequence:

Step 1

Develop the slave interface logic to the Cortex-M3 processor for receiving parameters such as the address of the play buffer and sample rate.

This slave interface logic is implemented in the FPGA fabric with different states such as Idle, ssetup, and saccess in reading the parameters from the Cortex-M3 processor through the FIC.

Step 2

The master logic APB is developed in the FPGA fabric using states msetup and maccess to read the audio sample from the play buffer in external SRAM. The signal clockdriven is derived from the fabclk based on the sample rate passed by the Cortex-M3 processor.

Step 3

The state dacwait is used to wait until the positive edge of the clockdriven signal. The states wsetup and waccess are developed for passing the audio sample to DAC.

Step 4

Step 2 and Step 3 are repeated until the all samples in the play buffer are played. After playing, the complete buffer stop signal is driven to High.

Simulation

Figure 13 shows the simulation of the Rice decoder by enabling the Rice decoder using the start1 signal. The rice parameter is taken as four and the required words as six to show the simulation result clearly.

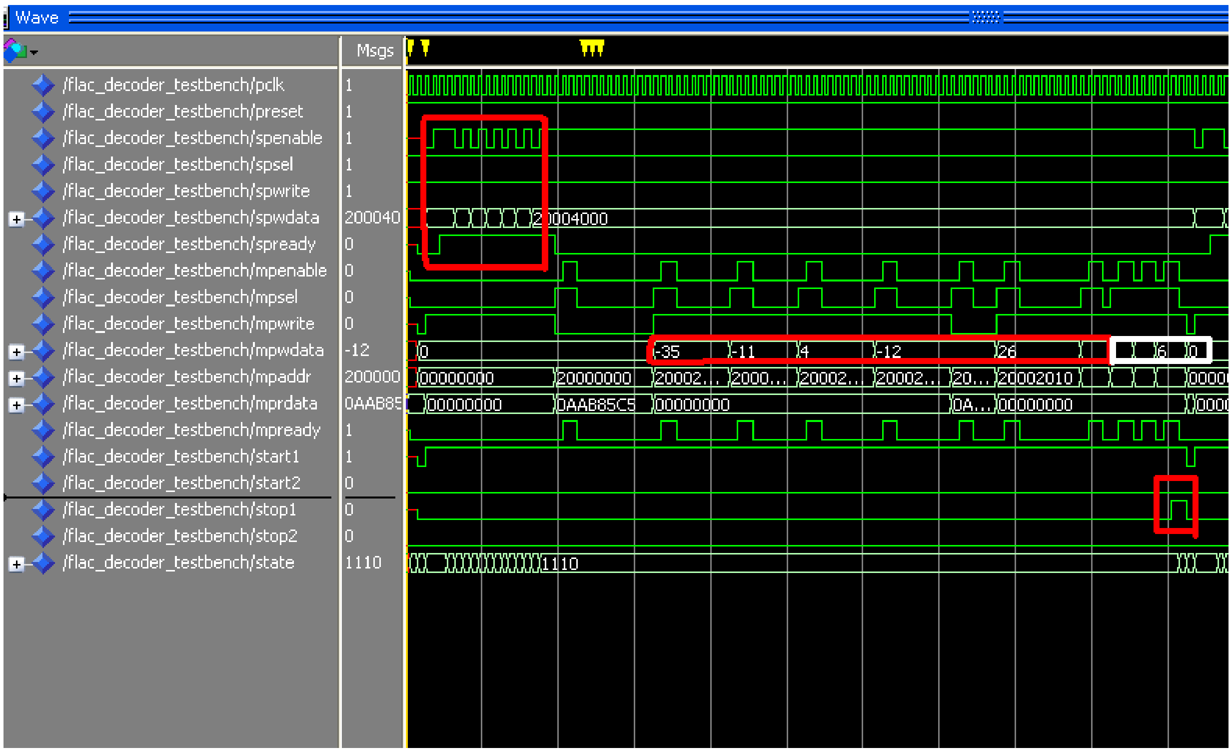

Figure 13 • Simulation of Rice Decoder (Only Six Words)

For 4,096-word decoding, Figure 14 shows the simulation result for an entire frame decoding. It takes 0.002284663858 seconds for complete decoding.

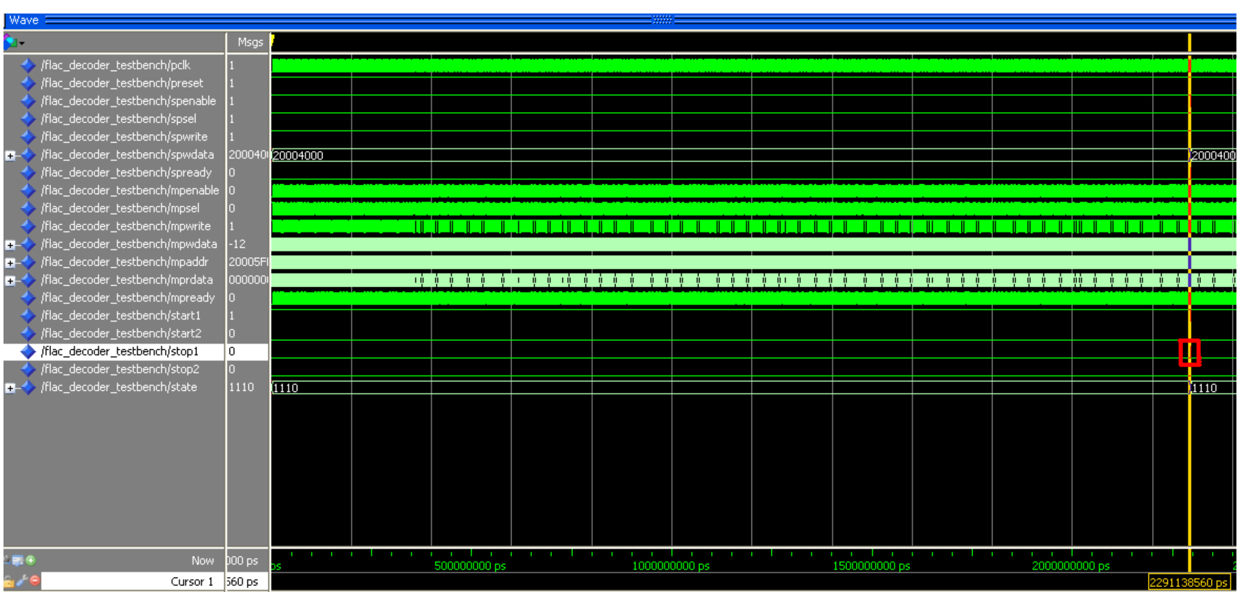

Figure 14 • Simulation Result of Rice Decoder For Complete Frame

Figure 15 shows the simulation of an LPC decoder by enabling the Rice decoder, using the start1 signal. The coefficients, warm-up samples, and residue values are taken by creating a memory in the testbench. For illustration purposes, only ten values are given as input for the decoder.

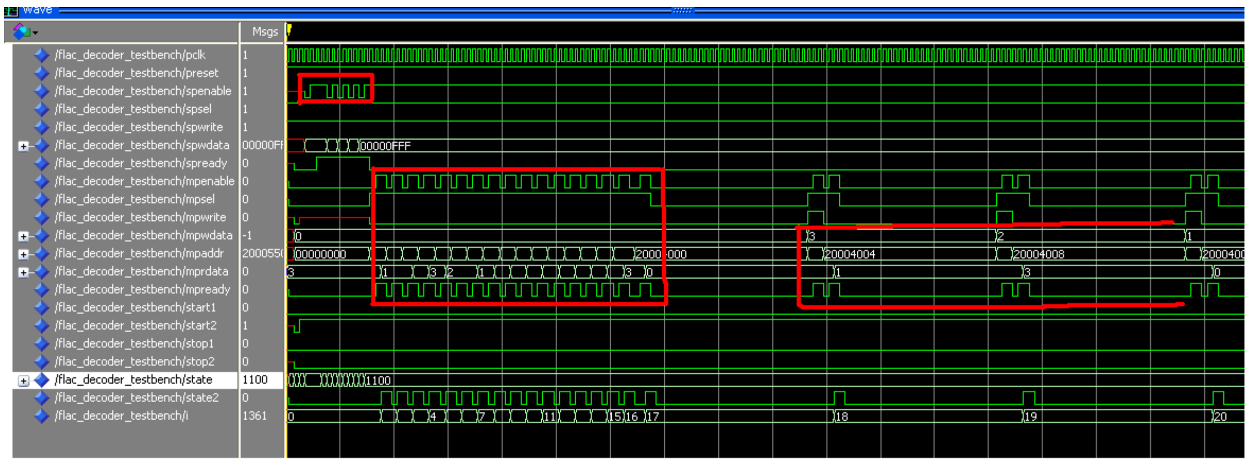

Figure 15 • Simulation of LPC Decoder Showing Few Values For Understanding

Figure 16 shows interrupt signal generation after completion of decoding. Here the total frame is decoded.

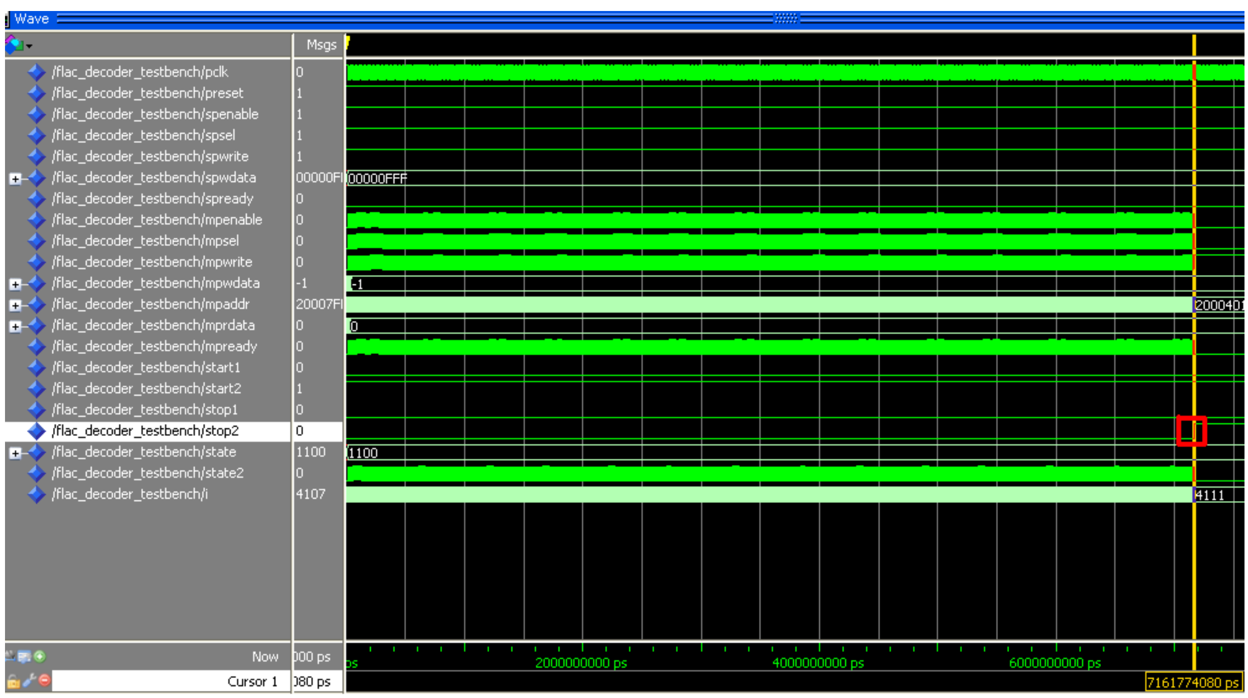

Figure 16 • Simulation of LPC Decoder For Frame Length

Running the Design

Board Settings

The design example works on the SmartFusion Development Kit Board with default board settings. Refer to the following user's guide for default board settings:

SmartFusion Development Kit User's Guide.

The FLAC decoder footprint is slightly larger, so an external memory that is available on the SmartFusion Development Kit Board is used to port the decoder on the SmartFusion Development Kit Board.

This solution requires connecting an amplified speaker system to the built-in DAC pins on the SmartFusion Development Kit Board (A2F500-DEV-KIT).

Figure 17 and Figure 18 on page 20 show the complete audio system setup with an amplified speaker system connection to the DAC (DACOUT0, pin3) and AGND (AGND, pin2) pins on the SmartFusion Development Kit Board.

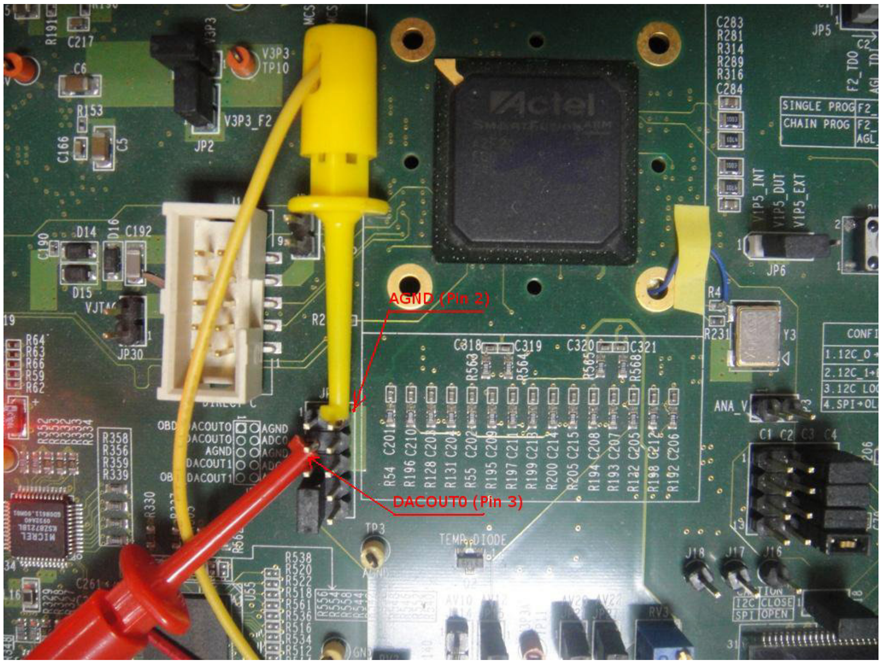

***Figure 17* • Amplified Speaker System Connection to the DAC Pins on A2F500 SmartFusion Development Kit Board**

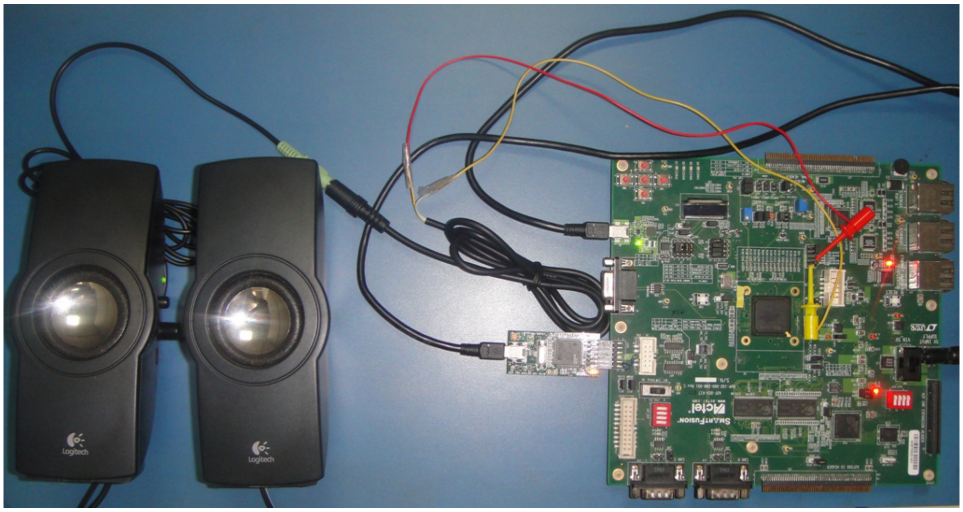

***Figure 18* • Complete Audio System Setup on A2F500 SmartFusion Development Kit Board**

Program the Design and Running the Application

To run the FLAC decoder, an encoded input file (*.flac) must be provided to the FLAC decoder. This encoded file (*.flac) can be generated using the encoder tool, flac.exe, for a given pulse code modulated audio data file that is in *.wav format. A sample *.wav file is available in the design files (*A2F_AC376_DF\Tools\flac-1.2.1-win*).

A Win32 binary tool can be downloaded from the FLAC source website. Open a command prompt window in the host PC and move to the directory where the encoder tool is located (refer to "Appendix A – Design Files" section on page 25). Type **flac.exe** and help on how to encode a *.wav file using the encoder tool is printed.

Figure 18 shows the encoder tool (flac.exe) help for encoding a *.wav file to a *.flac file. A sample encoded *.flac file is available in the design files (*A2F_AC376_DF\Tools\flac-1.2.1-win*).

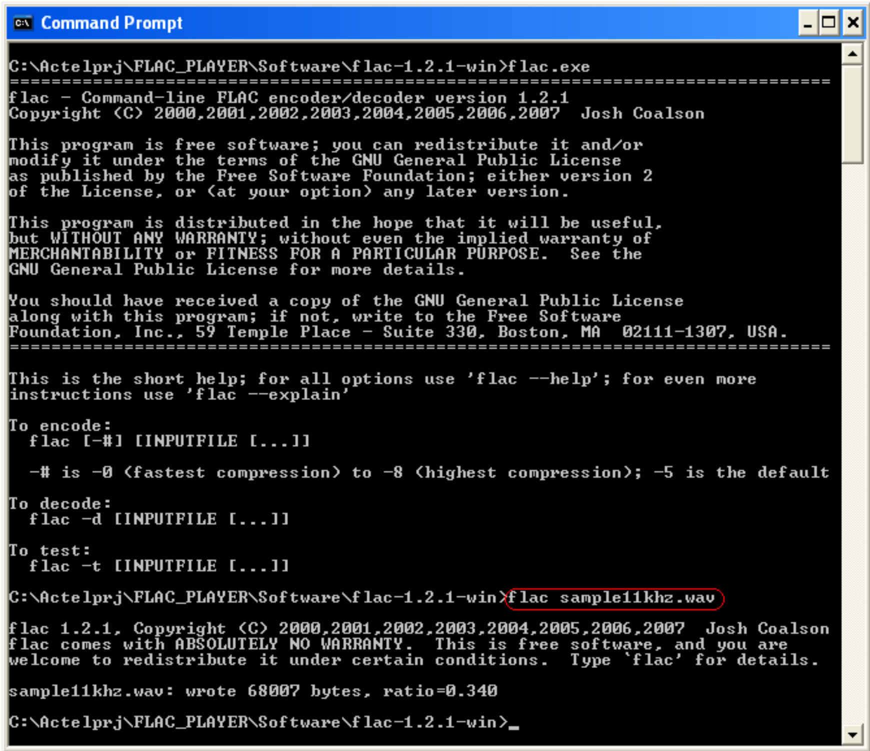

***Figure 19* • Encoding *.wav File to *.flac File**

Program the SmartFusion Development Kit Board with the downloaded STAPL files and then power cycle the board. Refer to "Appendix A – Design Files" section on page 25 for information on using FlashPro.

To open the SoftConsole project, double-click **Write Application Code** under **Develop Firmware** in the Libero SoC **Design Flow** window. Build the project using the build options provided in the project.

Figure 20 shows the SoftConsole Explorer view for the FLAC_Decoder_SC_prj project.

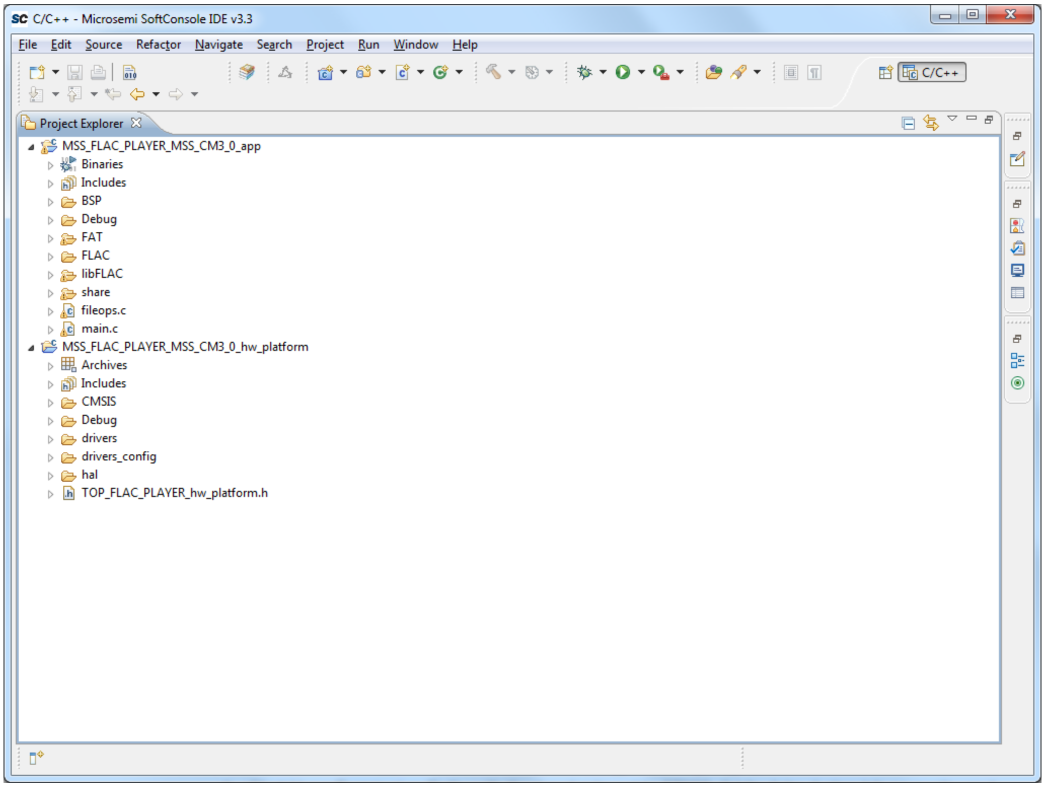

Figure 20 **• Project Explorer View**

1. Launch the debugger and run the project **FLAC_Decoder_SC_prj**.
2. Start HyperTerminal or Putty with the baud rate set to 57600, 8 bits data, 1 stop bit, no parity, and no flow control. The application starts printing the file system commands on HyperTerminal.
3. To play a *.flac file, type the command **fl** in HyperTerminal to select one of the available files from the list of the files that are stored in SPI flash file system on the target board. If there are no stored files in SPI flash, create a file to transfer the *.flac file from the host PC to SPI flash file system.

Figure 21 shows HyperTerminal in creating a *.flac file on SPI flash.

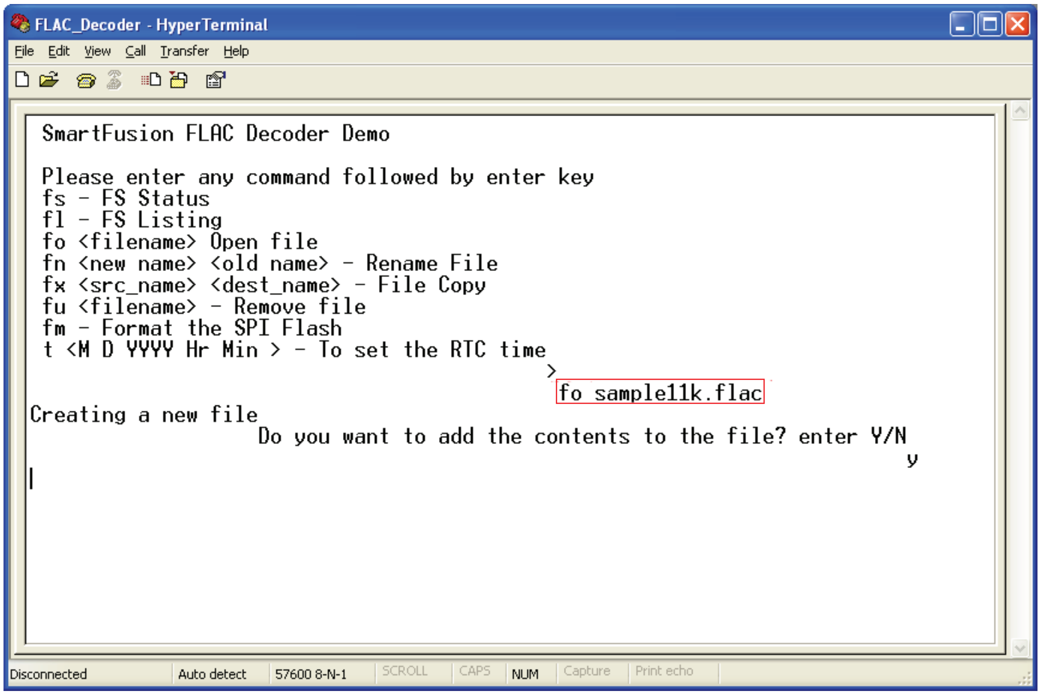

***Figure 21* • HyperTerminal in Creating a FLAC File on SPI Flash**

4. After creating a file on SPI flash, disconnect from HyperTerminal and launch the memory loader at the command prompt. Memory loader is an executable program (*.exe) provided in the host tool folder. The memory loader program runs on the host PC and is used to transfer the encoded file (*.flac) from the host PC to the board.

   Open a command prompt window in the host PC and move to the directory where the host tool is located (refer to "Appendix A – Design Files" section on page 25). Type **ASF2memoryloader.exe**. Help on how to run the host loader is printed.

   Type the following command to load the *.flac file into SPI flash file system:

   ```
   ASF2memoryloader.exe spi 0x0 sample11k.flac 3
   ```

Figure 22 shows the memory loader (ASF2memoryloader) programming to transfer the *.flac file from host PC to the target.

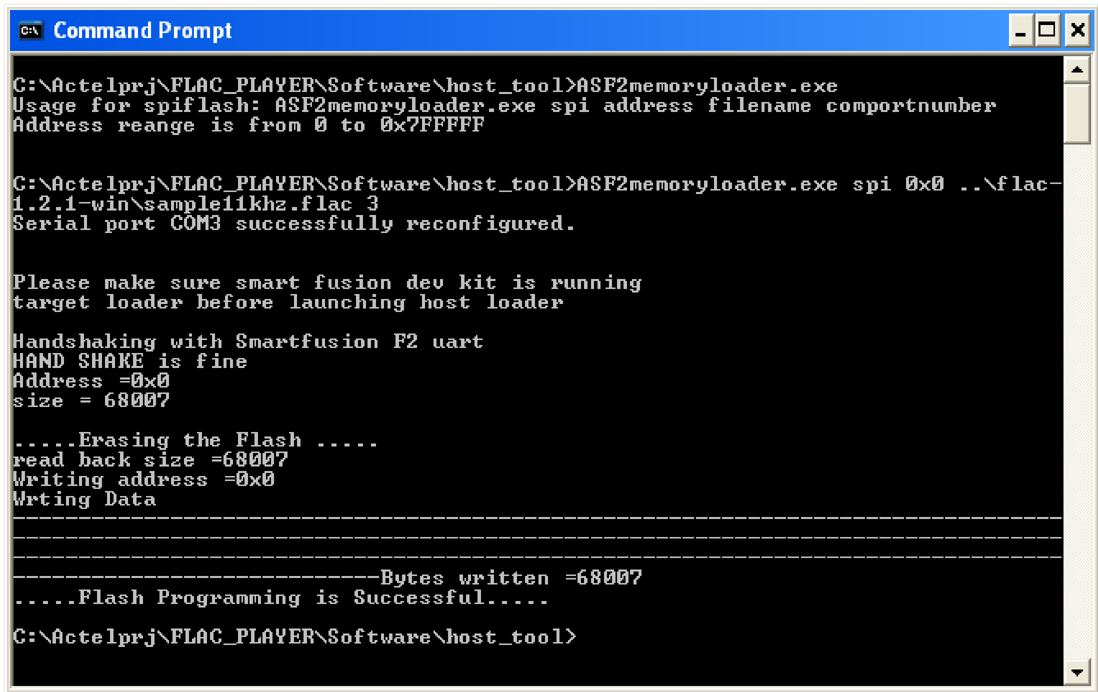

Figure 22 • **Debug Messages After Running the Memory Loader (ASF2memoryloader) on Host PC**

5. After completion of the file transfer, connect with HyperTerminal to see the decoder status on HyperTerminal.

Meanwhile, the decoder starts decoding the samples and sends the decoded samples to the DAC to construct an audio signal that is played on the speaker system. You can hear audio playing from the speaker.

Figure 23 shows the debug messages while FLAC decoder decodes the encoded *.flac file.

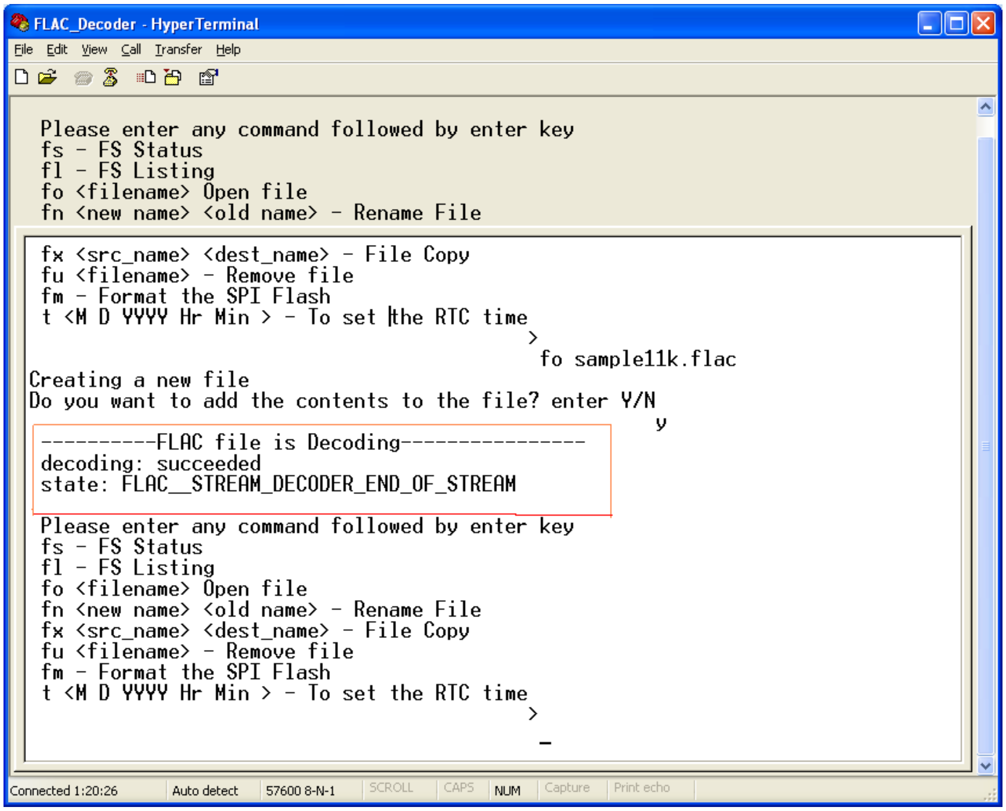

Figure 23 **• Debug Messages While Decoding the *.flac File**

Conclusion

This application note describes the capability of the SmartFusion cSoC device for multimedia audio applications. This application note also explains the following:

1. Porting of the FLAC audio decoder on SmartFusion cSoC, decoding, and playing the encoded *.flac file in real time using the built-in DAC.
2. Using the FPGA fabric to implement a portion of the FLAC DSP algorithms to improve the decoding performance.

Appendix A – Design Files

You can download the design files from the Microsemi SoC Products Group website:

www.microsemi.com/soc/download/rsc/?f=A2F_AC376_DF

The design zip file consists of Libero SoC project and programming file (*.stp) for A2F500 Development Kit Board. Refer to the ReadMe.docx file for directory structure and description.

List of Changes

The following table lists critical changes that were made in each revision of the document.

Revision*	Changes	Page
Revision 2 (January 2013)	Added "Board Settings" section and "Program the Design and Running the Application" section (SAR 43469).	19
Revision 1 (February 2012)	Figure 6, Figure 9, and Figure 20 were updated (SAR 35836).	10, 13, 22
	The "Running the Design" section was updated (SAR 35836).	19
	The "Appendix A – Design Files" section was updated (SAR 35836).	25

Note: *The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

51900244-2/01.13