White Paper
SPRAAW3–August 2008

Audio Bitstream Coprocessor Architecture Illustration Using MP3 Decoder

Fitzgerald J. Archibald, Ramesh Naidu G., Stephen H. Li, Michael O. Polley

In this paper, an audio coprocessor architecture is discussed for low power applications like portable audio players and music playback features in cellular phones. The architectural details are split into hardware and software domains. The audio coprocessor hardware consists of a bitstream processor to handle parsing, buffering and control, and an arithmetic unit to handle math intensive operations. Additionally, the hardware contains peripherals, memory modules and interfaces like ports, DMA, RAM/ROM and a Inter-IC sound interface. The use of hardware features and capabilities is illustrated with an MPEG1 Layer 3 decoder as an example.

Index Terms: audio coprocessor, audio decoder processor, audio encoder processor, MP3 decoder

Contents

List of Figures

1 Introduction

This paper explains the decoding of an MPEG 1 bitstream with a dual core architecture model. The bit processing unit (BPU) performs bitstream parsing and Huffman decoding, and acts as master for controlling the arithmetic unit (AU). The AU performs math intensive operations like dequantization, inverse transform and synthesis filter.

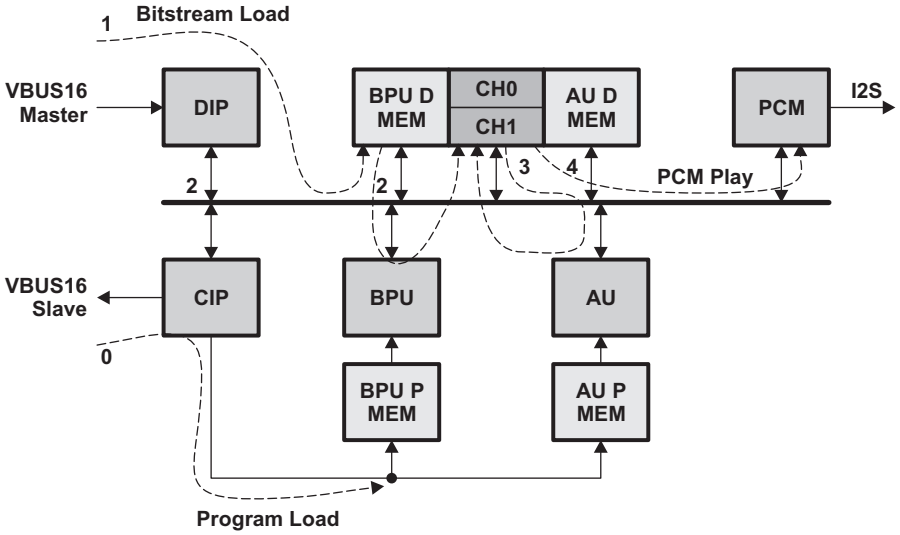
The paper is organized into three major sections namely hardware description, BPU software and BPU-AU interface. The hardware description focuses mainly on BPU hardware architecture. The BPU software section provided insight into BPU software architecture and top-level software design for MP3 decoder.

The BPU-AU interface provided insight into control and data flow across various modules in this dual core audio coprocessor architecture by taking an MP3 decoder as an example. The AU architecture is not described in detail as this is similar to other digital signal processor (DSP) architectures with capabilities limited to audio processing functionality and precision requirements.

2 Hardware Description

The functional block diagram of the audio core is shown in Figure 1. The audio core is composed of two autonomous processing units: bit processing unit (BPU) and arithmetic unit (AU), and I/O peripherals. The BPU and AU interface through shared memory. The BPU is essentially a control processor with special instructions and hardware for bitstream parsing. The AU is a programmable fixed-point computation engine for performing DSP operations like discrete cosine transform (DCT) and filters [5]. Core and I/O peripheral synchronization is carried out by the BPU (master). The audio core can be used as a coprocessor for audio codecs and effects.

Figure 1. JAZZ Audio Core Block Diagram



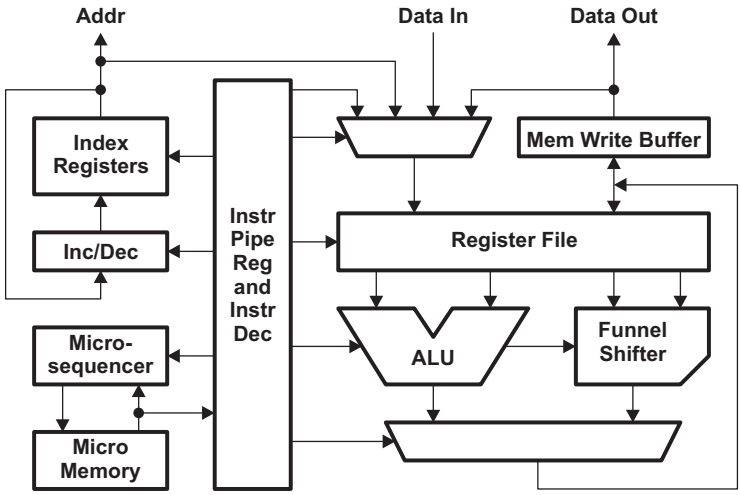
The audio core is a slave requiring a master (HOST) for initialization. The master could be a general-purpose processor like ARM [7]. While holding the audio core in reset (halting processor execution), the HOST loads programs of BPU and/or AU (flow 0). The BPU can load programs onto the AU. For reducing program memory size, memory overlay can be used. That is, only the program that is needed is loaded onto the audio core memory. The program/data memory is loaded dynamically based on need. Though dynamically loaded, the dynamic loading is per stream decode/encode. That is, program/data is not dynamically loaded for each frame [4].

Consider Figure 1 with respect to an MP3 decode. The HOST fetches the streams for decode onto the external memory (SDRAM) from the hard-disk or flash memory. The HOST would inform the data start and size parameters to the audio core via the control input port (CIP). The BPU can program the data input port (DIP) DMA to fetch the streams onto the internal memory of the audio core (flow 1). The BPU unpacks the bitstream and decodes variable length (Huffman decode) encoded information. The BPU shares the control information extracted from the bitstream and the Huffman decoded frequency samples into shared memory (flow 2). The BPU gives control to the AU for the generation of PCM samples from frequency samples. The AU performs de-quantization, joint-stereo, anti-alias butterfly, inverse modified DCT (IMDCT), overlap-add, frequency inversion and synthesis. The synthesized PCM samples are written into shared memory (flow 3). Once the PCM samples are available in shared memory, the AU gives control to the BPU for streaming the samples to the DAC via the PCM port (flow 4). An inter-IC sound (I2S) interface is supported for connecting the external DAC to the PCM port [10].

The BPU is a programmable processor with hardware acceleration and instructions customized for audio decoding. It is a 16-bit RISC processor with register-to-register operations and an address generation unit operating in parallel. This unit is capable of performing an ALU operation, a memory I/O, and a memory address update operation in one system clock cycle.

The unit has two pipeline stages: Instruction Fetch/Pre-decode, and Decode/Execution. The decoding is split and merged with the Instruction Fetch and Execution respectively. This arrangement reduces one pipeline stage and thus branching overhead. Also the shallow pipe in Figure 2 operation enables the processor to have a very small register file (three general purpose registers, a dedicated bitstream address pointer, and a control/status register), since memory can be accessed with only a single cycle delay [2].

Figure 2. BPU Block Diagram



The architecture model includes memory (RAM and ROM), a 5 by 16 bit ALU register file (r0, r1, r2, r3, r6 (bit)), 6 by 13 bit Index Register file (ir0, ir1, ir2, ir3, ir4, ir5) status register (sr), I/O enable register (en/r4), program counter (pc) and a special register named r5 with all 1's named as "-1" or r5. The bit register is used for holding bitstream position in terms of bits as required by bitstream parsers.

Each instruction word contains a memory control part and an ALU control part. In each cycle, the microcode engine can perform one ALU operation, and one memory read or write operation.

The destination of a memory read is always one of two ALU registers (r0/r1), or the PC for branch instructions. The source of a memory write is always the ALU result. Inputs to ALU operations always come from the ALU register file, and the destination is one of the source registers, memory, or an index register. This architecture effectively pipelines the operand reads for the ALU operation, but makes the pipeline registers part of the programming model. This gives more flexibility and decreases control logic for the pipeline.

The ALU register R5 (alternate name "-1") contains constant bit pattern of all ones. Thus when used as an ALU source it is not necessary to load the actual value -1 into a register. This is used to implement increment, decrement, logical complement, and logical and arithmetic shift right by one. R5 cannot be used as an ALU destination.

There is no stack in the machine. Interrupts are handled by a one-level memory-mapped interrupt return address register. Interrupt nesting is handled by copying the return address to a private memory location. Subroutines are handled by explicitly passing the return address in the register file. These methods are straightforward when the interrupt handler and/or subroutine are not re-entrant [1].

2.1 Addressing Modes

There are four possible modes:

- **Immediate**: Load a signed 13-bit value from the instruction word.
- **Direct**: Load a memory location specified by a 13-bit field in the instruction word.
- **Register**: Load a value from index register IR0-5, or ALU register R0 or R6.
- **Indirect**: Load a value from memory, addressed via index register IR0-5, or ALU register R0 or R6.

2.2 Interrupts

There are six sources of interrupts:

- Data input port (DIP)
- PCM output buffer empty
- Control input port (CIP)
- Input buffer breakpoint
- Arithmetic unit operation complete
- Real-time failure

The first three interrupts (DIP, PCM, and CIP) are by convention only. These three interrupt sources are external to the BPU/AU core. The last three interrupts are internal to the BPU/AU core and are hardwired.

Data Input Port (DIP) Interrupt

The DIP interrupt is generated when the DIP finishes the DMA copy of new data into the memory. This interrupt is used for DMA copying of the new data each time, as the input bitstream contains large amount of data.

PCM Buffer I/O Breakpoint Interrupt

This interrupt is similar to the input buffer I/O breakpoint interrupt and is used by the PCM buffer output unit to signal the output buffer empty condition. The response should be to refill the PCM output buffer. Placing this under microcode control allows more flexible management of the PCM output buffer, leading to reduced buffer size.

Control Input Port (CIP) Interrupt

This interrupt is generated by the CIP port. This interrupt service routine (ISR) can be used for receiving commands issued by control processor (ARM).

Input Buffer Breakpoint Interrupt

This interrupt is generated when the bit/r6 register touches the word boundary of the programmed memory word. This interrupt can be used for managing the circular input buffer. Using the breakpoint interrupt handler to wrap the read pointer allows the size of the buffer to be optimized for the determined worst case buffer conditions.

Arithmetic Unit Operation Complete Interrupt

This interrupt is generated after the AU executes the NAP instruction signaling the end of the execution sequence started by the auop () instruction.

Real-Time Failure Interrupt

There are situations that occur during the correct operation of the decoder which force real-time deadlines to be missed. These can be caused by pathological input bitstreams which underflow the PCM output buffer due to limited input buffer capacity. These events must be detected to take corrective action.

2.3 Memory

The microcode engine uses a 13-bit address for each of the instruction and data spaces. The data space contains both RAM and table lookup ROM. The memory unit used here is word (16 bits).

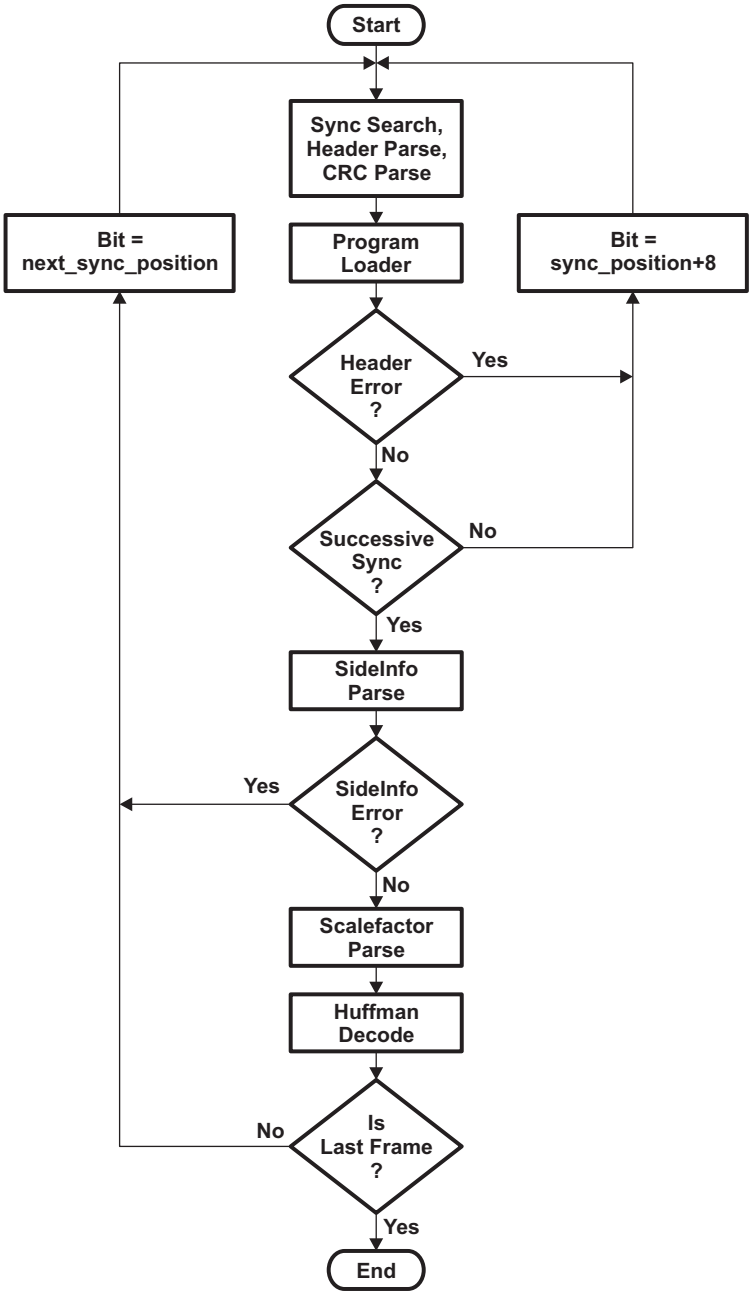
Address Space	
0x0000-0x0FFF	BPU RAM (4K)
0x1000-0x1BBF	BPU ROM
0x1BC0-0x1BFF	Memory Mapped Registers
0x1C00-0x1FFF	AU RAM Window (1K)

3 BPU Firmware Case Study

This section uses an MPEG 1 Layer 3 decoder for illustrating the program flow and buffering capabilities of the BPU.

Figure 3 provides a top-level program flow in the BPU for an MPEG1 Layer 3 decoder. At the start of decoder operation, the bit pointer (bit) is initialized to the buffer start position of the input buffer. Sync word is searched on byte by byte basis. On finding the sync word, the header and CRC information are parsed.

Figure 3. MP3 Decoder Flowchart on BPU



Upon finding the layer of the stream, respective layer code is loaded into program memory. The parsed header information is validated with the header information of the 11172-3 standard. If there are any header errors, the bit is moved back to the found sync position + 8 bits. This is done to minimize the possibilities of pseudo sync word detection.

Upon finding the valid header info, the successive sync position is parsed with the frame size. If the successive sync position is not present, the bit is moved back to the found sync position + 8 bits. This step is used for validating the sync word.

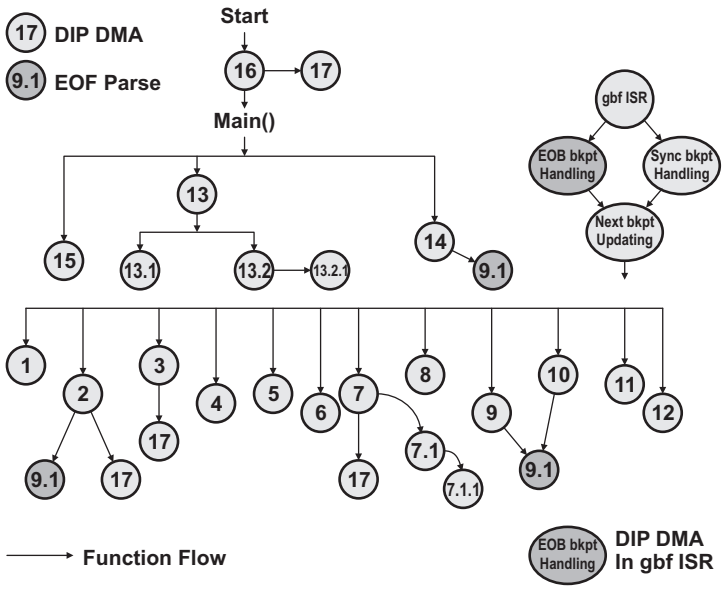
After finding the successive sync, the Side Information is parsed. The parsed side info parameters like main_data_begin, block_type, scfsi, etc. are tested for validity [3]. If there are any errors, the bit is moved to the next sync position. This allows for graceful handling of corrupted frames.

The main data, which includes Scale Factors and Huffman data, is parsed. After the completion of the decoding of main data, if the decoded frame is not the last frame, the bit is moved to the next sync position for further decoding. If the decoded frame is the last frame, the decoding operation is closed.

3.1 BPU Firmware Modules

The function of the BPU is to generate the frequency samples and manage all the hardware resources as it acts as master in the programming environment. The BPU program is divided into small modules as shown in Figure 4. This section explains briefly about each module. The numbers in parentheses indicate the module number. The flow proceeds from START and goes through various modules and ends up in module (9.1), and EOF parse is called. The reader is assumed to have familiarity with the MPEG1 audio standard [3]. Input buffer of size 499 words (480 + 19(SI Maximum)) is used for decoding purpose.

Figure 4. BPU Modules Tree Diagram



(1) **Sync Search**

Search for sync word (0xfff) in the bitstream on each byte boundary. If the bit pointer exceeds the end of file (EOF) marker, exit sync search stating EOF reached.

(2) **DIP DMA gbf ISR**

This function triggers DMA to refill if the sync word is not found up to the EOB. Refilling is needed to bring new data in the input buffer starting from the previous refill point to EOB. On completion of refill, the routine branches to the INTERRUPT RETURN address of sync search function. It calls module 9.1 (EOF parsing) if sync search exceeded the EOF mark during sync search. (Note: gbf is the instruction for extracting bits from the bitstream buffer).

(3) **Sync DMA**

If the expected sync position is not equal to the actual frame start position (having junk between frames), a refill is needed from the previous refill point to the present frame start.

(4) **Header + Cyclic Redundancy Code (CRC) Parse**

This module parses the Header and CRC info which follow the sync word.

(5) **Header Error**

This routine validates the header bits for each MP3 frame. If any header error is found except emphasis (not used for any computation of decoding process), it returns back with error code and the bit position is wrapped to the present frame start + 8 bits for further sync search.

(6) **Main_data_begin_extraction**

This function extracts the main data begin location (9 bits) from the bitstream, which is used for buffering.

(7) **MD Begin Look Back DMA**

This function finds the main data begin location in the bitstream buffer using the non-zero main data begin value extracted from the bitstream, with the help of the MD Bit Look Back routine. If the present main data begin location falls in the previously parsed main data or future data (due to circular buffering), returns with error code.

After finding the main data begin buffer pointer, the number of words from the previous buffer refill point to the present main data begin (for non-zero main data begin) or current frame start (for zero main data begin) are filled with bitstream for further decoding by use of DMA.

(7.1) *MD Bit Look Back*

This subroutine helps to find the main data begin buffer pointer for the non-zero main data begin by skipping the Header and Side Info bits. This routine uses the H_SI_skip routine, if needed.

(7.1.1) *H_SI skip*

Skip the Header and Side Info bits during the MD Bit Look Back function.

(8) **SI Parse**

Parse the Side Info from the main data begin extraction point.

(9) **Look Ahead Sync**

This routine searches for a successive sync in the bitstream for decoding the current frame. If sync is found, the function returns to caller successfully. If sync is missing but not the last frame, the function returns back with error code. If the last frame decode with the frame fully available in the buffer, the function returns with the last frame decoded samples. If the current frame is the last frame, this routine invokes Song Completion.

(9.1) *Song Completion (EOF parsing)*

This function is called by Look Ahead Sync when the last frame is not fully in the buffer or by the DIP DMA gbf ISR when the bit pointer exceeds the EOF mark during sync search or on completion of the last frame decoding by BPU.

(10) **Buffer Error handling**

This is used to handle errors by muting the PCM buffer when look ahead sync is not found (bit moved to frame start + 8 bits) or the wrong main data begin, like the first few frames pointing to future data or the present main data begin falling in previously used main data (bit moved to next Sync position). If the error is in the last frame decode returns to EOF parsing.

(11) **Buffer Management**

This module sets up the breakpoints for the main data parsing. The sync position that follows the present main data begin is found for non-zero main data begin case and the break point is set at (sync word-1) word if the main data begin position is less than the sync position else on EOB.

(12) **CRC and scfsi check**

This is used to validate the CRC and scfsi of the MP3 audio frame. If the parsed CRC and computed CRC do not match, error code is returned. As CRC is optional in the stream, CRC validation cannot be used for error detection entirely. The scfsi is forced to zero for a frame if any of the granules have non zero scfsi for block_type=2.

(13) **Main data parsing**

This function parses the main data of two granules, and one or two channel(s) of a frame.

(13.1) *Scale factor parsing (Part2 length)*

The scale factors (long/short/mixed) are parsed.

(13.2) *Huffman Parsing (part3 length)*

If part2 length > part2_3 length for grX/chY, the sample memory is cleared to zero without any further decode of the frame. If no errors are detected, Huffman parsing is done.

(13.2.1) *Remove Stuffing Bits*

If stuffing bits are present for grX/ChY, these are removed using this module with 1 bit advancement each time.

(14) **MD Completion**

After a frame is decoded, the bit pointer is advanced for the next frame decode and returns back to the main () position. If the last frame decoding is done, returns to EOF parsing.

(15) **Header Side Info**

This is the module for decoding the frame data up to the main data begin of a frame. If any header/buffer errors are encountered, branches to the sync search module.

(16) **Master Reset Handler**

This module is used to initialize the flags and variables that are used for decoding process. The first 499 words in the bitstream are fetched into the buffer using the DMA for the start of decoding operation.

(17) **DIP DMA**

The BPU is forced to sleep during DMA transfer for saving power (when no data is available for processing). It also gives the EOF marker position if the file size words are completely transferred by DMA.

gbf ISR

This ISR is triggered by any of the modules, namely 1(sync search), 4(Header and CRC), 8(SI parse), and 13 (Main data parse). The breakpoint can be from EOB for all the above modules and SYNC breakpoint for module 13 only. This ISR maps to the input buffer breakpoint interrupt. Depending on the breakpoint that caused the interrupt, further processing is carried out.

EOB Bkpt Handling

The last two words of the buffer are stitched to the top (first two locations of the input buffer) and the bit is wrapped to the top [9].

If sync search module (1) caused EOB breakpoint handling, the processing differs from the usual return to the INT RET address. This is due to failure of sync detection up to the EOB. In such a case, the processing comes out of gbf ISR saving the SYNC search module context to temporary memory and branch to the point colored with BLUE color stating that DIP DMA is triggered in gbf ISR. This is done to refill (from previous refill point to EOB) while keeping the ISR execution time short. Thus, in such case the return from ISR does not follow the usual rule.

SYNC Bkpt handling

The 2 or 2 words (based on the position of sync) are stitched into to the H+SI info of the breakpoint hit frame [9].

Next Bkpt Updating

The next breakpoint update is done only for main data parsing module (13) irrespective of the breakpoint hit position (that is, from EOB or SYNC). The next breakpoint is kept at the next (SYNC-1)th word if the main data of the present frame is in past frames or EOB if the main data is in the present frame only.

3.2 Program Flow

The BPU parses the header, CRC and main data begin of frame after finding the sync word of a frame in the buffer. The BPU initiates the DMA for refill from the previous refill point up to the main data begin of the current frame. The main data begin position gives how many number of words can be refilled into the buffer from the previous refill point. Once the new data is copied into the buffer using DMA, BPU looks for next frame sync word in the buffer to ensure that the detected sync word is not a false sync. After completing successive sync search successfully, BPU parses the remaining SI. The main data parsing which includes Scale Factors and Huffman samples is handled through the hardware input buffer breakpoint by a mechanism called *buffering* without any additional main data buffer. For buffering, the maximum frame span is calculated with this formula:

Maximum Frame Span=Maximum input Buffer Size/ Minimum Frame size (480/48 -> 10).

A Maximum frame span of 10 indicates that the present frame can have its main data in the previous nine frames and also in its frame.

The parsed scale factors and Huffman data, along with some parameters of SI that are together called as Control data, are copied into the shared memory between BPU-AU for granule0. After this copy the BPU gives control to the AU. The AU, which is designed for math operations, takes the frequency samples and generates the 16-bit PCM samples into the shared memory and signals the BPU. The BPU copies these data into the PCM buffer for audio playback by a D/A converter (DAC). The BPU again generates control data for granule 1 and the above procedure continues for granule 1. A similar flow follows for both channels if there is stereo audio.

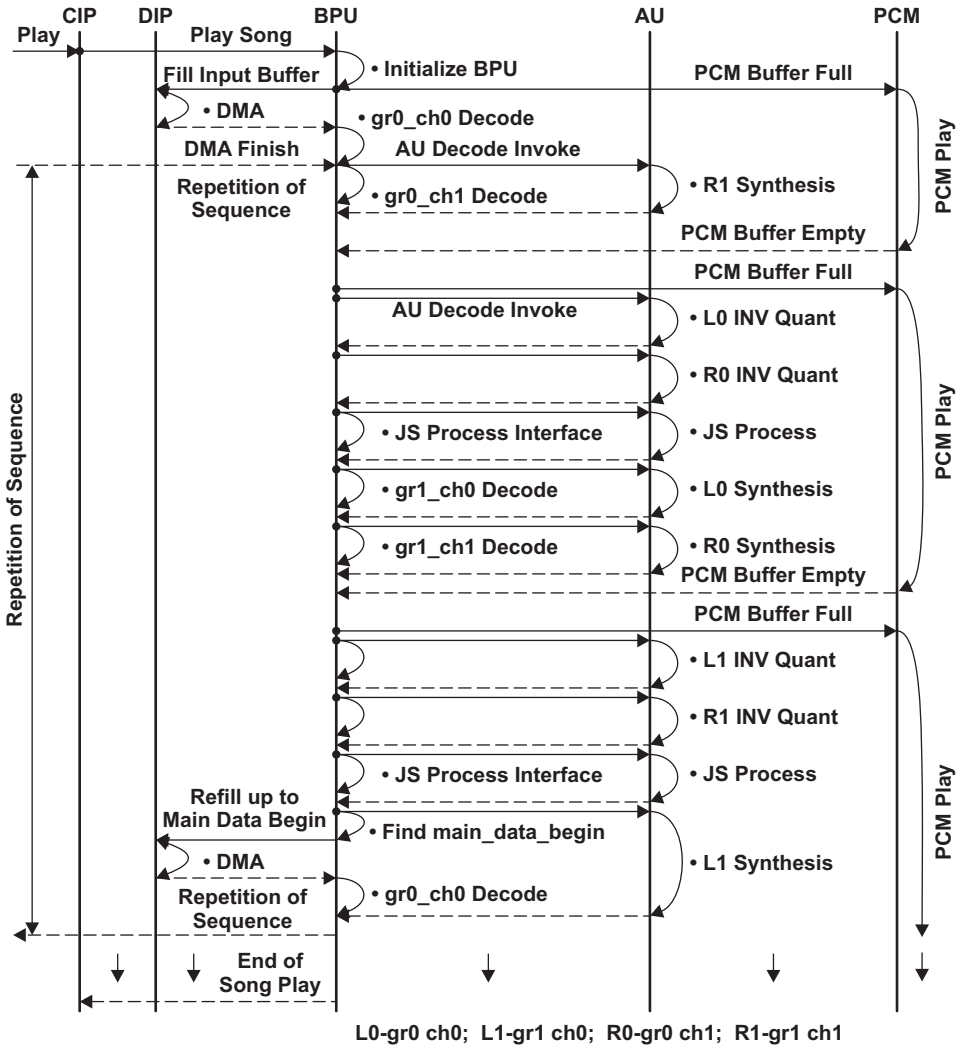
After a frame is fully decoded the bit pointer is moved to the next sync position, obtained from the Look Ahead Sync module. The decoding procedure continues in similar fashion as explained above for each frame.

The input buffer breakpoint interrupt is initiated when the bit pointer reaches the EOB. In the gbf ISR the bit pointer is wrapped to the top of the input buffer. Thus, this interrupt serves for circular buffering [9].

4 CIP-DIP-BPU-AU-PCM INTERFACE

The following steps explain the decoder and audio playback operations using various hardware blocks in the coprocessors. Figure 5 gives the timing and sequencing of various operations with respect to MP3 decoder.

Figure 5. Sequence Diagram of CIP, DIP, BPU, AU, PCM



1. The CIP initiates the BPU for the start of song playback.
2. The BPU initializes the data RAM locations like flags, variables, shared memory between BPU and AU (Left and Right Banks), PCM buffer, etc. that are needed for the decoder operation to start.
3. The BPU initiates the PCM for PCM data playback. The BPU initiates the DIP DMA to fill the input buffer of 499 words (Maximum frame size (480) + Maximum SideInfo (19)).
4. The BPU parses the gr0_ch0, which includes Header, SideInfo, scale factors and Huffman samples (frequency samples).
5. The BPU copies the frequency samples of gr0_ch0 along with some parameters of SideInfo, called control data into the shared memory of left channel memory. The BPU initiates the AU for generation of PCM samples for gr1_ch1 (Initially, the right bank is cleared by BPU). The AU accesses the shared right channel memory and runs the synthesis operation for generating the PCM samples. The BPU runs in parallel with the AU and decodes the gr0_ch1 frequency samples.
6. The BPU waits until the previously initiated operations of the AU and PCM are complete.
7. After the AU (synthesis) and PCM operations are completed. The BPU refills the PCM buffer with new samples and initiates the PCM for playback. The BPU copies the gr0_ch1 control data into the shared memory of right bank. The BPU invokes Inverse Quantization operation of gr0_ch0 of AU. The BPU waits until the AU operation is completed. After the AU completes, the BPU again initiates the AU for inverse quantization of gr0_ch1 and waits for AU completion.
8. The BPU and AU perform Joint Stereo (JS) processing for gr0. Here the BPU performs all data movement operations associated with JS, while the AU performs ordering and JS processing.
9. The BPU initiates the AU for generation of PCM samples for gr0_ch0. The BPU runs parallel with the AU and generates the frequency samples for gr1_ch0. After the AU completes operations, the BPU initiates the AU for gr0_ch1 PCM samples. The BPU runs in parallel with the AU and generates the frequency samples for gr1_ch1.
10. The BPU waits until the previous PCM buffer is played out. After the PCM buffer is played out, the BPU refills the PCM buffer with the AU generated PCM samples.
11. The BPU copies the gr1_ch0 and gr1_ch1 control data into shared memory of left and right channel memory, respectively. The BPU invokes Inverse Quantization operation of gr1_ch0 of the AU and waits for AU completion. After the AU completes, the BPU again initiates the AU for inverse quantization of gr1_ch1 and waits for AU completion.
12. The BPU and AU perform Joint Stereo (JS) processing for gr1. Here the BPU performs all data movement operations associated with JS, while the AU performs ordering and JS processing.
13. Now a full frame is decoded by BPU. The BPU initiates the AU for generation of gr1_ch0 PCM samples. The BPU runs the gr0_ch0 decode of the next frame concurrently with the AU processing. The BPU buffer, after finding the main data begin of the present frame decode in gr0_ch0 decode, needs a refill for the further decode of the frame. So the DIP DMA is initiated for refill up to the main data begin. The main data begin pointer gives a good indication of how much refill can be done. The BPU waits for further decode until the DIP completes the DMA. The DMA and AU run parallel.
14. After the DMA finishes data transfer, the BPU completes the remaining decode of gr0_ch0 and generates frequency samples for gr0_ch0.
15. The BPU waits until the AU completes the generation of PCM samples for gr1_ch0.
16. Step 6 is repeated.

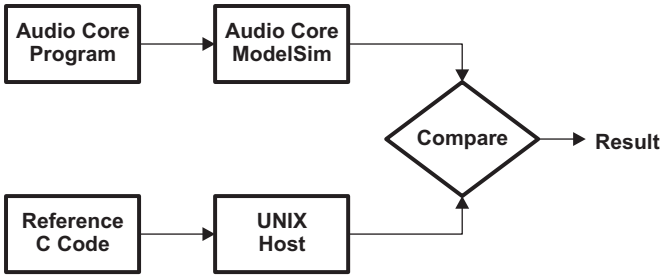
The processing, control and data flow continue in a similar fashion as explained in the above steps until the bitstream is entirely decoded (until the end of stream is reached).

5 Test Platform

The test platform is shown in Figure 6. The BPU and AU programs are tested on the JAZZ ModelSim (Register Transfer Level) RTL simulator. The breakpoints were inserted in the respective modules and the parsed data of the header, SI, Huffman and scale factors were dumped into files. A reference C code was run on the UNIX host and similar data was dumped into files. The files taken from both the JAZZ RTL simulator and the Reference C code were validated with file comparison software. If there is any mismatch of the files, the respective module assembly code was corrected and tested again. This process was carried on a frame to frame basis. The decoder was tested with many error test cases and PCM output was checked for mute in case of any bitstream errors.

The integration of BPU-AU was tested by generating PCM samples and analyzing the SNR and precision of output as specified by compliance test standard [6].

Figure 6. Test Platform Flowchart



6 Conclusion

The decoding of MPEG1 Layer 3 bitstream with a dual architecture model is realized on a ModelSim RTL simulator platform and validated for performance and compliance. The model is estimated to consume around 1mW power for MP3 audio playback (digital). The model is realizable using 20k Gates approximately with 40kB RAM or RAM/ROM combination. The firmware can be extended to perform AAC, AAC+, WMA9 and other format decoding. Additionally, audio effects algorithms also can be implemented to work on PCM or frequency samples.

7 Appendix

The memory usage and the BPU cycles are shown in Table 1. The cycles and memory comparison can be better understood from the rows Total/Max for coprocessor and c55x [8].

Table 1. Profile for MP3 Decoder

	Memory (Words)			MPS[1]
	RAM	ROM	CODE	
BPU	3189	2902	2816	14.4
AU	3584	847	1032	21.1
Total/Max	6773	3749	3848	24.9
C55x[2]	14000	7300	8500	21.5

(1) 48 KHz Stereo
(2) General purpose DSP

8 Acknowledgment

Authors thank Mohamed Mansour, R&D, Texas Instruments Inc. for his contributions in the development of MP3 decoder firmware on the audio coprocessor.

9 References

1. *Programming of Audio Coprocessor for MP3 Decoder* by R. G. Naidu., F. J. Archibald, S. H. Li; submitted for publication.
2. *An AC-3/MPEG Multi-standard Audio Decoder IC* by S. H. Li, J. Rowlands, P. Ng, M. Gill, D.S. Youm, D. Kam, S.W. Song, P. Look; CICC 1997, Proceedings of the IEEE 1997 Volume , 5-8 May 1997, pp. 245 - 248.
3. *Coding of moving pictures and associated audio and digital storage media at up to about 1.5Mbit/s, Part3: Audio*, ISO/IEC 11172-3:1993
4. *Audio system for portable market* by F. J. Archibald, AES Preprint 6906, Convention 121, Oct 2006
5. *Fixed-point arithmetic: An introduction* by R. Yates, Refer: http://www.digitalsignallabs.com/fp.pdf
6. *Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s, Part 4: Compliance testing*. ISO/IEC 11172-4:1995
7. *ARM926EJ-S Technical Reference Manual*. Refer: http://www.arm.com/pdfs/DDI0198D_926_TRM.pdf
8. *TMS320VC5509A Fixed-Point Digital Signal Processor* (SPRS205)
9. *Bitstream Buffer in Audio Coprocessor* (SPRAAJ6 by F. J. Archibald, S. H. Li, M.O. Polley, R. G. Naidu.
10. *I2S Bus Specification*, Refer: http://www.nxp.com/acrobat_download/various/I2SBUS.pdf

Fitzgerald Archibald earned B.E (Electronics and Communication Engineering) from PSG College of Technology, Coimbatore, India in 1996. He worked on onboard control systems software development for geo-synchronous satellites from 1996 to 1999 in ISRO Satellite Centre, Bangalore, India. In 2001-2002, he worked on speech decoder, real-time kernel, and audio algorithms for DVD audio team in Sony Electronics, San Jose, USA. While in Philips Semiconductors (Bangalore, India, and Sunnyvale, USA) in 1999-2001 and 2002-2004, he worked on audio algorithms and systems for STB, DTV, and internet audio. He is part of the DSPS group in Texas Instruments Inc, Bangalore, India from 2004-till date working on audio, video, and imaging systems and algorithm development.

Ramesh Naidu was born in Anantapur and received BTech (Electronics and Communications), from Intell Engineering College, Anantapur, India in 2004. This author is pursuing MTech (Telematics and Signal Processing) in NIT Rourkela, India from July 2005-June 2007.

He is doing MTech project work in Texas Instruments, Bangalore, India in the role of project trainee.

Stephen Li earned his BS in Electrical Engineering at University of Texas at Arlington, MS in Biomedical Engineering at UT Arlington/UT Southwestern Medical Center, and MS in Applied Mathematics at Southern Methodist University.

He joined Texas Instruments in 1984. He has worked, in different capacities, on the design, modeling, processing, application, and architecture definition of VLSI IC. He is the author of several technical papers and journal articles covering VLSI architecture and design. He owns over 25 U.S. patents in the same area, as well as A/V algorithm development.

Mike Polley received his B.S., M.S., and Ph.D. degrees in electrical engineering from the Massachusetts Institute of Technology. In 1996 Mike joined Texas Instruments to help TI influence the DSL standards bodies and enter the ADSL market. Mike led the ADSL research group developing DSP based solutions for evolving DSL standards. Mike then came unwired - he helped TI establish a fixed-wireless access chipset business and then led the way developing multiple-antenna wireless LAN technology and implementations in advance of the IEEE 802.11n standard. Mike currently manages the architecture team developing camera and video chips for cell phones and he is a Distinguished Member of Technical Staff.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Clocks and Timers	www.ti.com/clocks	Digital Control	www.ti.com/digitalcontrol
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Telephony	www.ti.com/telephony
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265