Application Report
SPRA779 - November 2002

MP3/AAC Player Implementation in RF3

Tsutomu Furuse — *DCES Imaging Software Development*

ABSTRACT

This application report introduces and describes an MP3† /AAC‡ audio player for use with the TMS320C54x™ digital signal processor (DSP) devices. This audio player is based on Reference Framework Level 3 (RF3). Reference Framework for eXpressDSP™ Software is a startware for developing applications that use DSP/BIOS™ and the TMS320™ DSP Algorithm Standard.

The main challenge of this application is to use LIO-based drivers for bitstream input and pulse code modulation (PCM) output. There is a large variety of input/output (I/O) devices. This approach provides great code reusability. As a result, developers can adapt this application for their own systems with minimum efforts.

This document also describes a number of DSP/BIOS techniques for decompression applications.

† MP3: ISO-MPEG Audio Layer-3 standard (ISO/IEC 11172-3, 13818-3)
‡ AAC: ISO-MPEG2 Advanced Audio Coding standard (ISO/IEC 13818-7)

Contents

List of Figures

List of Tables

1 Introduction

1.1 Application Overview

This application is originally designed for TI's C54x™-based TMS320DSC25, a low-power, programmable DSP imaging platforms. This processor has an ARM7TDMI core, imaging accelerator, and a wide variety of peripherals in addition to the TMS320C5409 DSP core. However, this application does not depend on those special modules. Therefore, it can be ported to other TI DSP easily.

This application was tested with TI TLV320AIC23 stereo audio codec device connected with the McBSP serial port interface of TMS320DSC25. The TLV320AIC23 includes high-performance analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) with highly integrated analog functionality for portable digital audio applications. Of course, flexibility of Reference Framework enables you to use other audio codec devices with minimum efforts. See Figure 1.

This application controls the following TI's XDAIS modules to play MP3/AAC bitstream:

- MP3 decoder (IDECODE)
- AAC decoder (IDECODE)
- Sample rate converter (ISRC)
- Volume controller (IVOL)
- (MP3) spectrum analyzer (ISPECAN) (optional)
- Equalizer (IEQ) (optional)

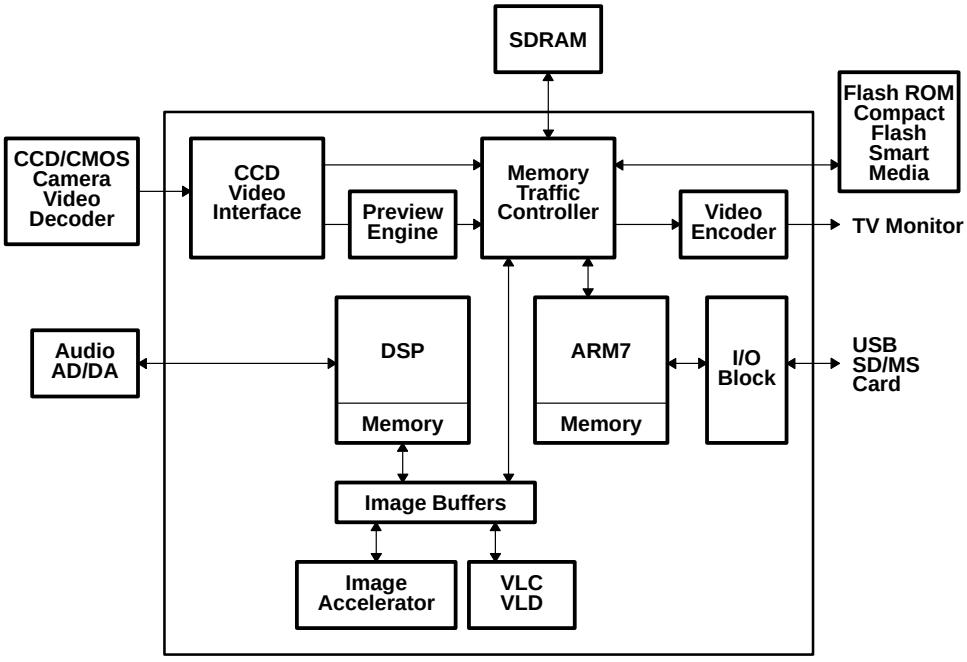


Figure 1. TMS320DSC25 Functional Block Diagram

C54x is a trademark of Texas Instruments Incorporated.

1.2 Application Features

This application provides the following commands:

- Start
- Stop
- Fast-Forward
- Fast-Rewind
- Pause
- Resume(from Pause)
- Resume(from FF/FR)
- Mute
- Volume

2 Data Path

Figure 2 describes the data path in this application. The input from the host processor and output to the multichannel buffered serial port (McBSP) should be abstracted because there is a large variety of I/O devices.

The decoder input buffer in the figure is a bitstream object, which is the standard input for the IDECODE interface implemented decoders.

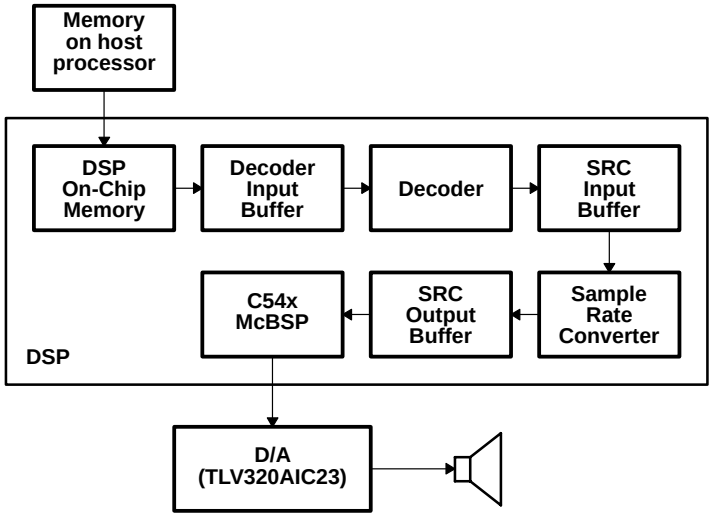


Figure 2. Data Path

3 Adapting RF3 to the MP3/AAC Player Application

3.1 Application Structure Overview

The Reference Framework Level 3 must be most appropriate level for this application. However, you need to change the RF3 code to adapt it to an MP3/AAC player. Figure 3 describes the modified application structure. As an RF3 mechanism, DSP/BIOS software interrupt (SWI) threads process bitstream, and the bitstream is driven by DSP/BIOS data pipes. To spread the central processing unit (CPU) load over processing time, you have two software interface (SWI) threads for the data processing.

The swiDecode thread decodes compressed bitstream in the pipRx pipe to PCM bitstream. Correctly, swiDecode extracts one MP3 frame (variable-length) from a pipe frame, then decodes to 1152 (Left:576 + Right:576) 16-bit PCM samples for MP3, For AAC, it decodes one AAC-frame (variable length) to 2048 (Left: 1024 + Right:1024) 16-bit PCM samples. While the pipe frame size of the pipDec is equal to the decoded size of a MP3/AAC-frame (1152 words and 2048 words, respectively), the pipe frame size of the pipRx pipe is larger than the size of MP3/AAC-frame. Therefore, the swiDecode thread may not finish up an input pipe frame for a run, but it finishes up a output pipe frame every run.

The swiPcmproc thread executes sample rate conversion, volume control, and filtering of the decoded 16-bit PCM data. The thread outputs 128 (Left: 64 + Right: 64) 16-bit PCM samples for a run. Like swiDecode, swiPcmproc may not finish up an input pipe frame for a run, but it finishes up a output pipe frame every run. Note that the sample rate conversion changes the number of samples. Naturally, the priority of swiPcomproc is higher than the one of swiDecode.

As mentioned above, the consumption rate is different from the production rate for the pipRx and pipDec pipe. See Figure 3.

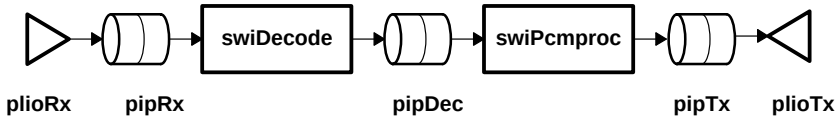


Figure 3. Basic Application Structure in RF

3.2 Thread Synchronization With Data Pipe

RF3 uses DSP/BIOS data pipe's notify functions to schedule threads. In RF3, notify functions do not post a software interrupt directly. Instead, they clear the specified bit from the SWI's mailbox, and the SWI is called by the DSP/BIOS kernel when all bits are cleared. See Figure 4.

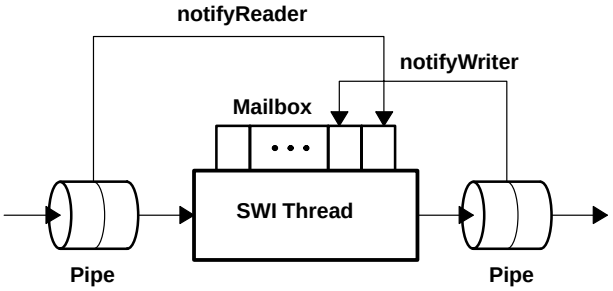
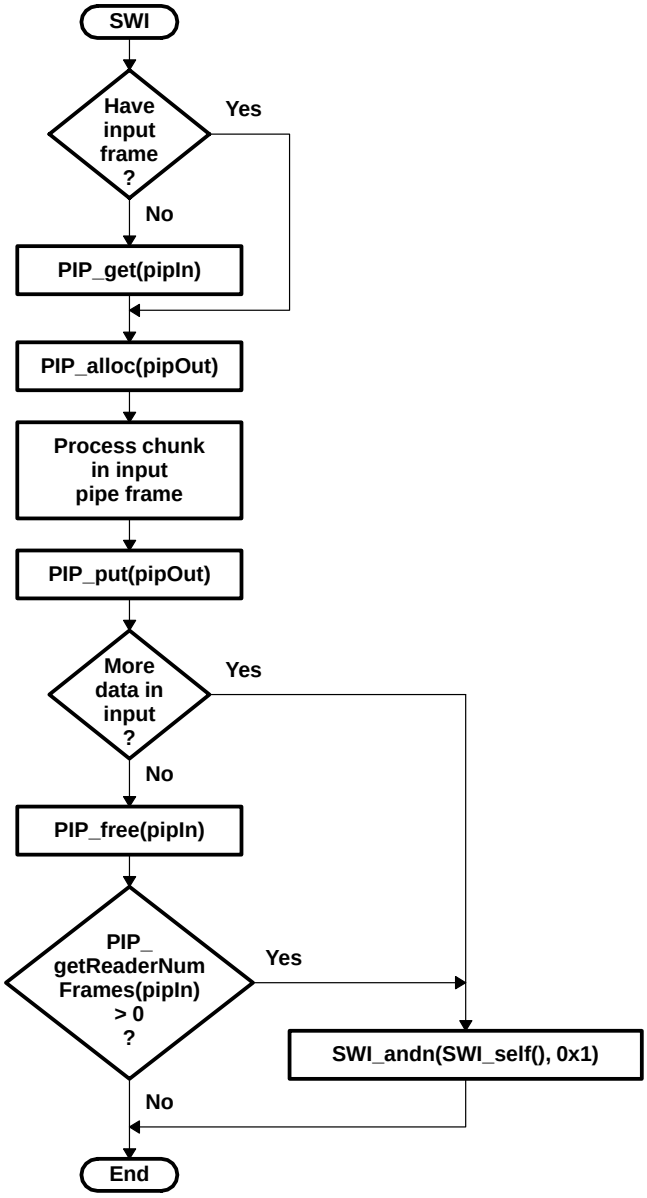


Figure 4. Pipe's Notify Function

For applications which do not change data size between the input and output or the change is fixed size, the above mechanism works fine. However, decoding compressed bitstream and sample rate conversion change data size between the input and output, and the size difference varies for each process.

For instance, the MP3 decoder captures a MP3 frame (variable-length) and decodes to a 1152-word PCM data. The decoder thread swiDecode locks an input frame until the decoder cannot find an entire MP3 frame. Only the decoder knows about MP3 frame, therefore you have a big enough sized input pipe for the decoder. The decoder generates multiple output pipe frames for an input pipe frame.

To support this situation, the decoder thread will lock an input pipe frame until all data in the pipe frame is processed. While a pipe frame is locked, the decoder thread needs to clear the notifyReader bit in the mailbox by itself. Figure 5 describes the process flow of the thread. The swiPcmproc thread also is designed with this style.



NOTE: Supposed that the notifyReader bit and notifyWriter bit are the bit 0 (LSB) and bit 1 respectively.

Figure 5. Data Process Flow

3.3 Disabling Pipe Notification

To support the stop and pause functions, you need to have a gate for notifyReader as illustrated in Figure 6. You can stop the bitstream processing by disabling the notification. This gate is implemented as an adapter function of SWI_andn. Note that this gate does not suppress the bitstream, but the notification. You can get a new pipe frame with PIP_get if an available pipe frame is in the pipe.

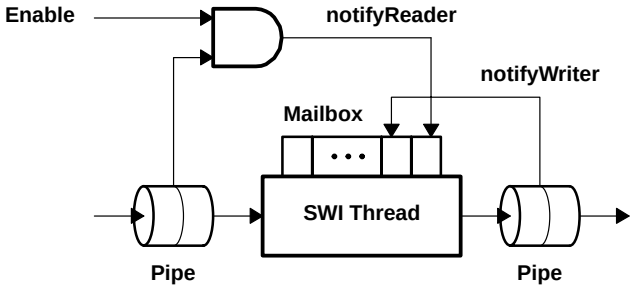


Figure 6. Notification Enable Flag

3.4 Notification Latency

Prior to start or resume play, you need to clear garbage data in the input data pipe. Use PIP_get function to clear the garbage. However, PIP_get function causes a spurious notifyReader call in this clear process because, if full frames are available after PIP_get gets a frame, PIP_get runs the function specified by the notifyReader property of the PIP object. For this reason, you need a mechanism for the notification prime latency. See Figure 7.

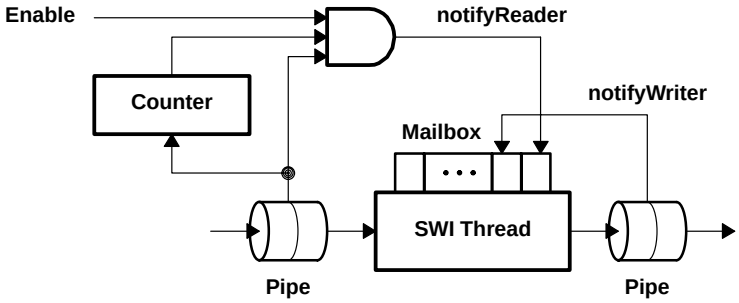


Figure 7. Notification Latency

The following code clears garbage in a pipe.

```
/**
 *  Clears a pipe.
 */
Int appPipeClear(PIP_Handle pip, PIPNTFY_Handle pipNotifier)
{
    Int readerFrames, i;

    readerFrames = PIP_getReaderNumFrames(pip);

    /* latency = readerFrames – 1 */
    PIPNTFY_setPrimeLatency(pipNotifier, readerFrames – 1);

    for(i = 0;i < readerFrames;i++) {
        PIP_get(pip);
        PIP_free(pip);
    }

    return (readerFrames);
}
```

The disabling pipe notification and notification latency are implemented with the following stub function. This stub function is configured as the notifyReader functions for the pipRx and pipDec pipes.

```
/**
 *  Primes a notification
 *  This is the notifier body.
 */
Void PIPNTFY_prime(PIPNTFY_Handle handle)
{
    if(handle–>isEnabled == TRUE) {
        if(handle–>primeLatency > 0) {
            handle–>primeLatency––;
        }
        else {
            SWI_andn(handle–>swi, handle–>mask);
        }
    }
}
```

3.5 Resetting SWI Mailbox

For the application's thread-scheduling mechanism, the notifyWriter bit of swiDecode's mailbox may remain a 1 when the play stops by the stop command. To ensure the mailbox will be reset for the next play, the swiDecode main function has an argument for the dummy call. The application will issue the dummy call at end of the stop process. The call does not process bitstream, but DSP/BIOS will reset the mailbox.

4 Application Structure

4.1 Software Modules

- Application threads
  - Decode thread

    The decode thread, swiDecode, decodes compressed bitstream to PCM data.
  - Pcmproc thread

    The PCM processing thread, swiPcmproc, executes sample-rate conversion, volume control, and filtering.
  - Control thread

    The control thread, swiControl, handles commands from the host processor and interprets them to the Decode and Pcmproc threads.
  - Evtmgr thread

    The event manager thread, swiEvtmgr, handles various events while MP3/AAC plays, monitors the system state, and replies the status to the host processor.
- XDAIS
  - MP3 decoder (IDECODE)
  - AC decoder (IDECODE)
  - Sample rate converter (ISRC)
  - Volume controller (IVOL)
  - (MP3) spectrum analyzer (ISPECAN) (optional)
  - Equalizer (IEQ) (optional)
- LIO drivers
  - Input driver
  - Output driver
- PLIO

  Pipe adapter for the LIO module. The upper layer of the device driver.
- LIO

  Low-level I/O device driver interface
- CSL

  Chip Support Library
- DSP BIOS

4.2 Application Execution Flow

Figure 8 and Figure 9 describes the execution sequences at the start command and pause/resume commands, respectively. These figures are based on the sequence diagram notation of OMG UML. The activation box (vertical bars on each process's axes) of SWI threads means the period of the thread running, and the activation box of LIO drivers means the period of the ISR running.

Suppose that the number of pipRx frames and pipDec frames are 1 and 2, respectively.

In the beginning of a play, the Control thread clears garbage data in pipes. In Figure 8, you can see clearing one frame of the input pipe and two frames of the decode pipe. The number of decode pipe clears varies from the previous state. You can see a reader notification is blocked for the notification latency function described in section 3.4.

The pause function is implemented with the stub function described in section 3.3. Disabling the reader notification blocks the Pcmproc thread priming. Then MP3/AAC play will be paused. To resume the play, the Control thread calls SWI_andn to clear the reader bit in the mailbox.

The number of swiPcmproc runs per swiDecode run varies from codecs (MP3 or AAC) or sample rates.

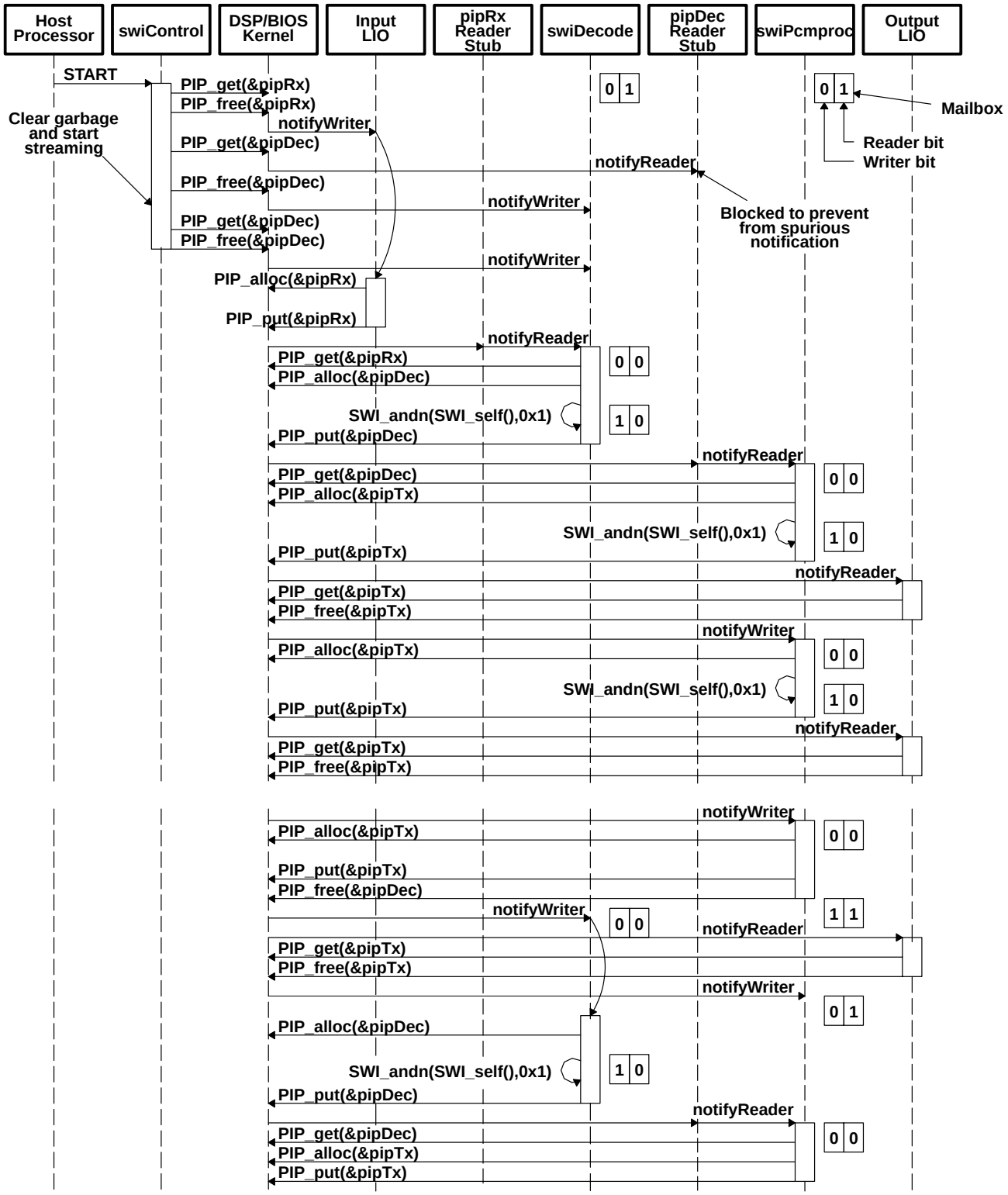


Figure 8. Start Sequence

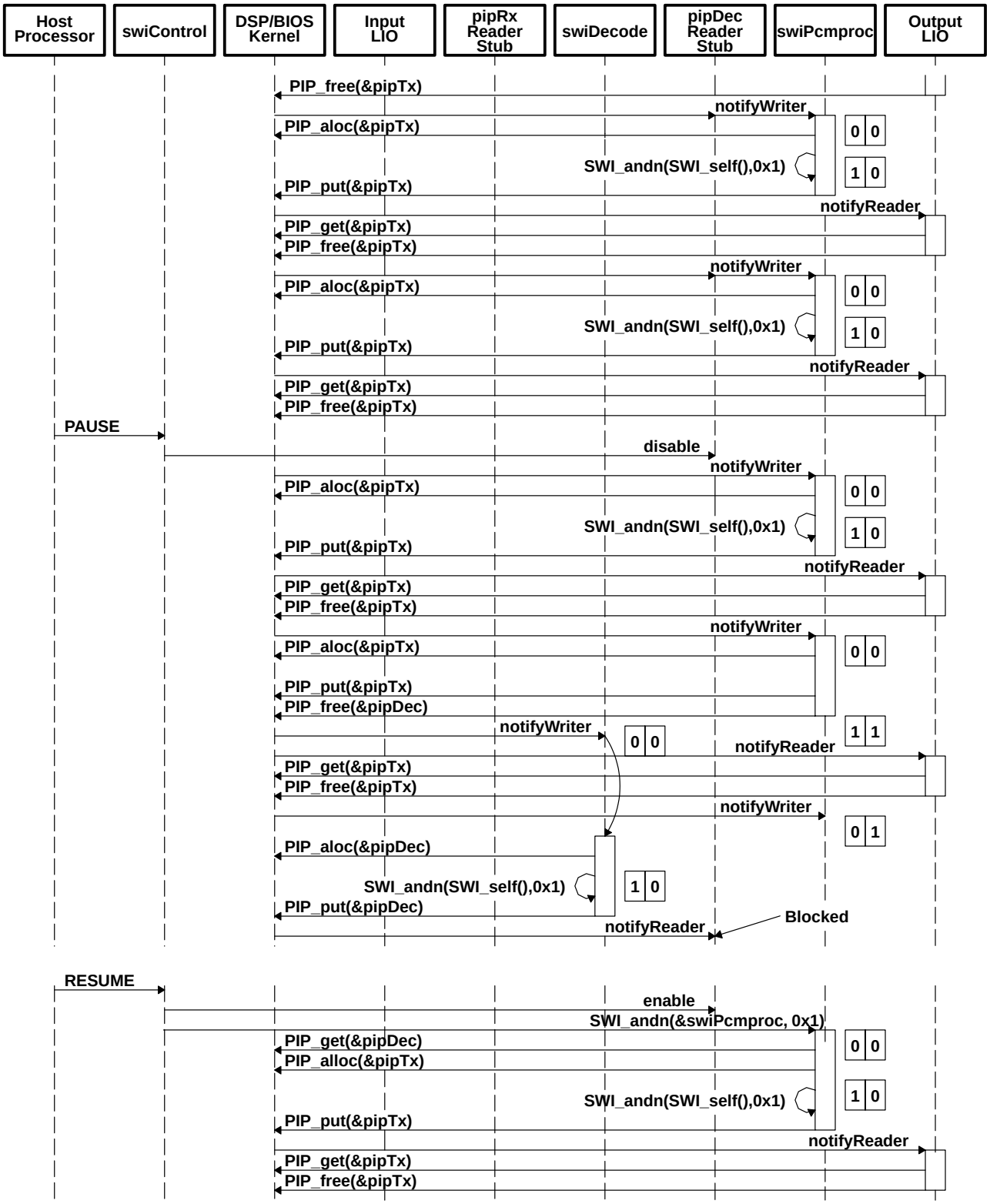


Figure 9. Pause/Resume Sequence

4.3 Application State Model

Figure 10 describes the state transitions when the application takes the commands. This diagram is compliant with the state diagram notation of OMG UML.

A small circle containing an ‘H’ is a history-state indicator. If a transition to the history indicator fires, it indicates that the object resumes the state it last had within the composite region.

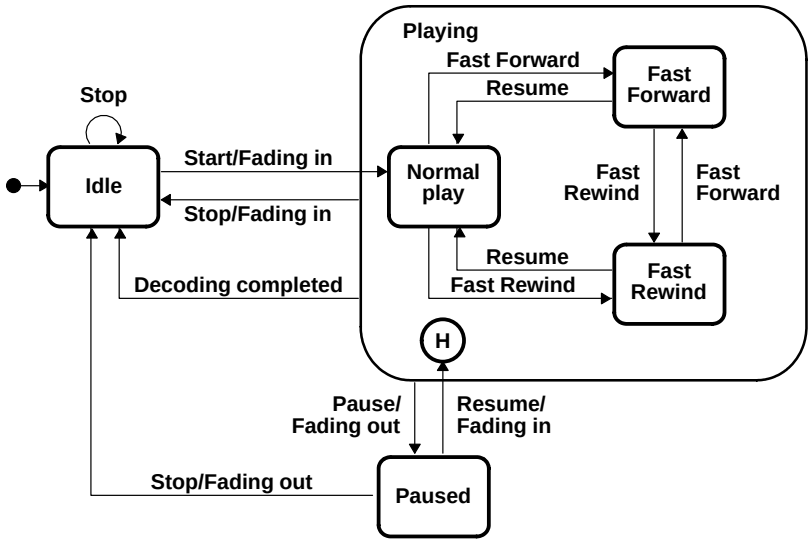


Figure 10. Player State Model

5 Performance and Footprint

5.1 Performance Characteristics

The MP3 performance characteristics of the application are shown in Table 1. The measurements are made under the following conditions:

- Platform: TMS320DSC25 (C5409 core) EVM running at 99 MHz
- Sample bitstream: Fraunhofer IIS MPEG Layer III Sample Bitstream

Table 1. CPU Usage Statistics

Title	Fs	Bitrate (total)	CPU Load (peak)
funky.mp3	44.1 kHz	96 kbit/s	46.20%
spot1.mp3	44.1 kHz	96 kbit/s	46.35%
spot2.mp3	44.1 kHz	96 kbit/s	46.20%
spot3.mp3	44.1 kHz	96 kbit/s	46.10%
classic1.mp3	22.05 kHz	56 kbit/s	39.88%
classic2.mp3	22.05 kHz	48 kbit/s	39.32%
Idle	–	–	2.07%

The CPU load percentages were measured with the CPU Load graph of Code Composer Studio™.

5.2 Memory Footprint

The memory footprints of the modified RF3 are shown in Table 2. XDAIS algorithms, algorithm-specific data buffers, LIO drivers, and DSP/BIOS are not included in these numbers.

Table 2. Memory Footprint

Modification	Footprint (Program)	Footprint (Data)
Modified RF3 for MP3	2278 words	286 words
Modified RF3 for AAC	2390 words	292 words

6 Conclusion

Reference Framework enables you to develop well-organized DSP applications rapidly. It is also true for compressed audio decoding applications. However, you need some DSP/BIOS techniques described in this application report for such applications. The MP3/AAC player application described in this application report shows that LIO-based drivers worked fine with the application.

7 References

1. *Reference Framework for eXpressDSP Software: RF3, A Flexible, Multi-Channel, Multi-Algorithm, Static System* (SPRA793).
2. *Reference Framework for eXpressDSP Software: API Reference* (SPRA147).
3. *Writing DSP/BIOS Device Drivers for Block I/O* (SPRA802).
4. *TMS320 DSP/BIOS User's Guide* (SPRU423).
5. *TMS320C5000 DSP/BIOS Application Programming Interface (API) Reference Guide* (SPRU404).
6. *TMS320C54x DSP CPU and Peripherals Reference Set Volume 1* (SPRU131).
7. *OMG Unified Modeling Language Specification Version 1.4.*

Appendix A Command Description

A.1 Host Commands

The host commands of this application are shown in Table A–1. A host processor issues these command to the DSP. The offset in the table is an offset address in the linker section, .cmdInBuf.

Table A–1. Host Commands

Offset	Start	Stop	Resume	Fast-Forward	Pause	Volume	Fast-Rewind	Mute
0x00	Command ID 0x0510 (MP3) / 0x0530 (AAC)							
0x01	Reserved							
0x02	0x0002	0x0003	0x0004	0x0005	0x0006	0x0007	0x0008	0x0009
0x03	Reserved	–	–	–	–	–	–	–
0x04	Data address (MSW)	–	–	–	–	Volume	–	–
0x05	Data address (LSW)	–	–	–	–	–	–	–
0x06	Buffer size (MSW)	–	–	–	–	–	–	–
0x07	Buffer size (LSW)	–	–	–	–	–	–	–
0x08	Fade-in cycles	Fade-out cycles	Fade-in cycles	–	Fade-out cycles	–	–	–
0x09	Volume	–	–	Volume	–	–	Volume	–
0x0A	Data length (MSW)	–	–	–	–	–	–	–
0x0B	Data length (LSW)	–	–	–	–	–	–	–
0x0C	Buffer offset (MSW)	–	–	–	–	–	–	–
0x0D	Buffer offset (LSW)	–	–	–	–	–	–	–
0x0E	Reserved	–	–	–	–	–	–	–
0x0F … 0x1F	Reserved							

A.2 DSP Reply

The DSP reply of this application is shown in Table A–2. The DSP issues this reply to a host processor. The offset in the table is an offset address in the linker section .cmdInBuf.

Table A–2. DSP Reply

Offset	Description
0x20	Reply code
0x21	Reserved
0x22	Stop status
0x23 … 0x3F	Reserved

A.3 DSP Command

The DSP command of this application is shown in Table A–3. The DSP issues this command to a host processor to inform the processing status. The offset in the table is an offset address in the linker section, .cmdOutBuf.

Table A–3. Host Command

Offset	Description
0x00	Command ID 0x200 (MP3) / 0x300 (AAC)
0x01	Reserved
0x02	Status ID
0x03	Left channel peak level in the frame
0x04	Right channel peak level in the frame
0x05	Elapsed time [x100ms]
0x06	Decoded data size (MSW)
0x07	Decoded data size (LSW)
0x08 … 0x09	Reserved

Appendix B Sample LIO Drivers

The specifications of the DSC25 image buffer DMA LIO driver and the DSC25 AIC23 LIO driver are shown in the following pages. The image buffer LIO driver and AIC23 LIO driver are used as the bitstream input driver and PCM output driver for the DSC25 platform, respectively.

The image buffer DMA LIO driver transfers data between the image buffer memory in the DSP data space and the host processor synchronous random-access memory (SDRAM) space by direct-memory access (DMA). This driver also has ring buffer management and address skip functions.

The AIC23 LIO driver outputs and inputs 16-bit stereo PCM data to/from an AIC23 codec device through C5409 DMA-McBSP.

B.1 DSC Image Buffer DMA Driver

Table B–1. DSC Image Buffer DMA Driver Overview

Driver Name	Vendor	Arch	Board	Version	Doc Date	Library Name
DSC2X_IBUF	Texas Instruments	DSC2x (5000)	DSC25 EVM DSC25 SDK	1.0	Aug 30 2002	dsc25_ibuf.l54

Table B–2. Hardware Interrupts Plugged

Hardware Source	Interrupt No.	ISR Name	Argument
Image buffer DMA controller	8 (External user interrupt #3)	DSC2X_IBUF_isr	None

Table B–3. Mandatory Configuration Parameters

Module	Parameter	Value	Description
HWI_SINT8	Source	Image buffer DMA	
HWI_SINT8	Function	DSC2X_IBUF_isr	Driver ISR
HWI_SINT8	Use dispatcher	True	Use HWI dispatcher

Table B–4. CTRL API Description

Description	Command Syntax	Argument
Gets the image buffer management parameters	Bool DSC2X_IBUF_ctrl(LIO_Mode mode, DSC2X_IBUF_CMD_GETSTATUS, Ptr args)	DSC2X_IBUF_MgmtParams *args
Sets the image buffer management parameters. Use this function to initialize or change the ring buffer position, size, and offset.	Bool DSC2X_IBUF_ctrl(LIO_Mode mode, DSC2X_IBUF_CMD_SETSTATUS, Ptr args)	DSC2X_IBUF_MgmtParams *args

Table B–5. Memory Overhead

Category	Sections	Size
CSL	.text	475
	.cinit	174
	.const	42
	.csldata	172
PLIO + driver	.text	1017
	.cinit	33
	.const	70
	.bss	28

NOTE: Reusable memory space: .text:init = 91

B.2 DSC25 AIC23 Driver

Table B–6. DSC25 AIC23 Driver Overview

Driver Name	Vendor	Arch	Board	Version	Doc Date	Library Name
DSC2X_DMA_MCBSP	Texas Instruments	DSC2x (5000)	DSC25 EVM DSC25 SDK	1.0	Aug 30 2002	dsc25_aic23.l54

Table B–7. Hardware Interrupts Plugged

Hardware Source	Interrupt No.	ISR Name	Argument
DMA 2	10 (DMA channel 2 interrupt)	C54XX_DMA_MCBSP_isr	0
DMA 3	11 (DMA channel 3 interrupt)	C54XX_DMA_MCBSP_isr	1

Table B–8. Mandatory Configuration Parameters

Module	Parameter	Value	Description
HWI_SINT10	Source	DMA channel 2 interrupt	Associate DMA 2 with McBSP 0 Rx
HWI_SINT10	Function	C54XX_DMA_MCBSP_isr	Driver ISR
HWI_SINT10	Use dispatcher	True	Use HWI dispatcher
HWI_SINT11	Source	DMA channel 3 interrupt	Associate DMA 3 with McBSP 0 Tx
HWI_SINT11	Function	C54XX_DMA_MCBSP_isr	Driver ISR
HWI_SINT11	Use dispatcher	True	Use HWI dispatcher

Table B–9. CTRL API Description

Description	Command Syntax	Argument
No CTRL commands implemented	N/A	N/A

Table B–10. Memory Overhead

Category	Sections	Size
CSL	.text	1193
	.cinit	174
	.const	42
	.csldata	172
PLIO + driver	.text	806
	.cinit	69
	.const	75
	.bss	74

NOTE: Reusable memory space: .text:init = 714

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 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.

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.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265