USER GUIDE

Real Audio Decoder Algorithm User Guide OMAP1710 (C55x)

Document Revision: 1.0

Issue Date: 01 March 2004

Table of Contents

List of Figures

LIST OF TABLES

Revision History

REV	DATE	AUTHOR	NOTES
1.0	01 Mar 2004	Srinivasa M.K	Initial Release

1. Introduction

This document describes the implementation of the RealNetworks Real Audio decoder for the OMAP1710 platform. It is intended to be used as a user's guide for building applications based on the TI implementation of the Real Audio 8 decoder. It references other artifacts including, but not limited to source code, object files, VOBs, XDAIS documents, spread-sheets and design documents.

1.1. Audience Description

The intended audience of this document includes developers, system integrators and test engineers.

1.2. Applicability Statement

This document describes the features, test procedure and usage of the RealNetworks Real Audio 8 audio decoder for the OMAP1710 platform.

1.3. Abbreviations, Acronyms and Definitions

- **RA:** Real Audio
- **TI**: Texas Instruments Incorporated
- **RN**: RealNetworks
- **DSP:** Digital Signal Processor or Digital Signal Processing, which ever is applicable
- **XDAIS:** e**X**press**D**SP™ **A**lgorithm **I**nteroperability **S**tandard (TMS320 Algorithm Interface Standard)
- **Kbps:** Kilo *bits* per second
- **KHz:** Kilo Hertz
- **MLT:** Modulated Lapped Transform
- **asm:** TMS320C55x Assembly language
- **C55x:** TMS320C55x DSP platform or device, which ever is applicable
- **codec**: Coder-Decoder algorithm, implementation or device
- **MAU: M**emory **A**lignment **U**nits (16-bit words for the TMS320C55x platform)

1.4. Notational Conventions

- Throughout the document the following conventions have been used:
- Program listings, program examples, and references to code appear in a `special typeface`.
- Unless otherwise specified, a **byte** refers to an 8-bit data type, while a **word** refers to a 16-bit data type.
- The term "algorithm" refers both to the procedure and its implementation depending upon the context used.

2. Program Description

2.1. Main Features

- xDAIS-2 compliant implementation of the RA8 decoder on OMAP1710 platform
- Support for following bit-rates (in kbps) : 8.0, 8.2, 9.9, 11.0, 12.3, 16.1, 6.5, 20.6, 22.0, 32.0, 44.0, 47.8, 64.0, and 96.3
- Support for following sampling-rates (in kHz) : 8, 11, 22, and 44.1
- Support for following stream types : mono, 2-channel mono (dual-mono), 2-channel joint stereo, 2-channel surround stereo
- Support for loss concealment using one-sided interpolation technique
- Available in three configurations:
  - low-mips version
  - constant-table DMA version (CONST_TABLE_DMA compiler flag)
  - low-memory version (WORKING_BUF_DMA flag)
- Integrated with DSP/BIOS™ Bridge DMA Manager for IDMA2/ACPY2 XDAIS specification.
- Independent of memory and peripheral configuration of device
- Data memory can be allocated at run-time by the framework.
- The data can use the whole addressing range (23-bit pointers).
- Fully validated on OMAP 1510 ES 2.6 silicon, Innovator™.
- Validated using multiple stack modes of the TMS320C55x DSP.
- Designed to work in a multi-threaded environment with multiple instances.
- Performance at 24mHz for 44.1kHz/96kbps stereo stream
- Compliant with the RealNetworks RA/RV Technology Compatibility Kit test cases (February 2003 version).
- Tested with code in external memory, and TMS32C55x I-Cache enabled
- Reduced internal memory requirements, achieved by allowing placement of constant tables in external memory. New linker sections have been defined for this purpose
- Requirement of only two physical DMA channels, mapped to two DMA-manager Queue IDs

2.2. Program structure

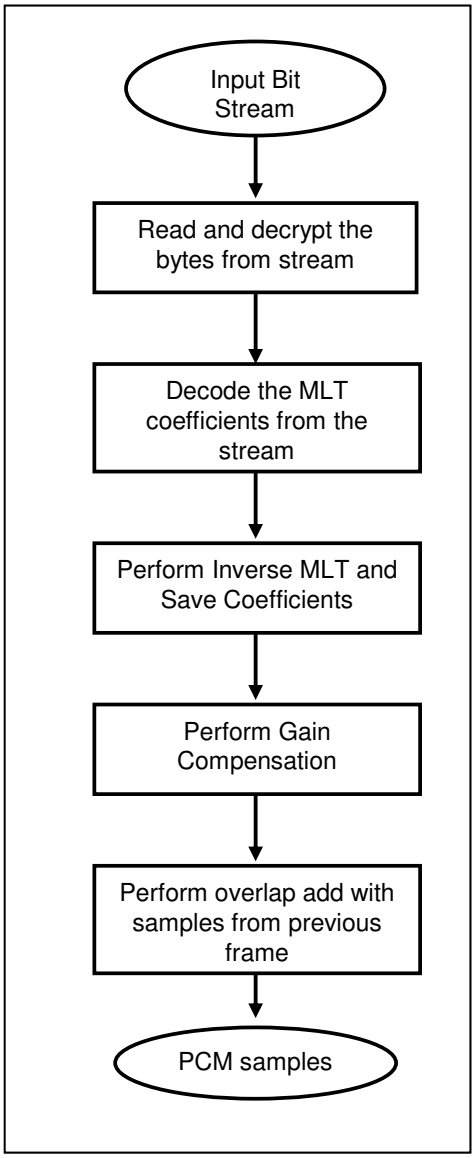

Figure 1 Flowchart of RA8 Decoder

2.3. Memory Optimization Scheme

In the TI implementation of the decoder, optimization of on-chip memory has been achieved primarily in 2 phases:

- Placement of constant tables in external memory
- Elimination of some working buffers, and placement of the rest in external memory.

These 2 different phases of memory optimization are conditioned by compiler flags, which lead to 3 possible configurations of the decoder:

- Best Performance / Maximum Memory Version (No flag)
- Medium Performance / Constant table optimization (`CONST_TABLE_DMA` flag defined)
- Low Memory Version (`WORKING_TABLE_DMA` and `CONST_TABLE_DMA` flags defined)

An MS-Excel spreadsheet gives the details about the Low Memory Version. Please see the section 13.4 for the details of this spreadsheet.

Certain constant tables have been now placed in the external memory of the DSP, thereby further reducing the internal memory requirements. To control the placement of these tables, new linker command sections have been defined in the COFF library of the RA8 decoder algorithm. Refer to the last sheet of the spreadsheet <OMAPSW_RA_Dec_Alg_SectionMoveDataSheet.xls> for more information about the memory placement of the tables and their impact on MIPS.

3. Library Characterization

The Real Audio decoder for the OMAP1510 Platform, is named **RA_TNI** following the XDAIS naming conventions. The library module is named `ra_tni.l55l`. This section gives the memory and performance characterization of the decoder module.

All figures quoted in this section were measured using a 96 Kbps/ 44.1 KHz stereo stream abba_25.enc (`abba.pcm` encoded using flavor 25), for the **low-memory** (WORKING_TABLE_DMA) version of the algorithm library. The algorithm was tested, validated and profiled on OMAP1510 ES2.6 silicon, on Innovator™ executing at 120MHz. Also, a summary of the characterization information for different audio streams is presented.

3.1. Code Memory Characterization

The code is assembled into the `.text` section of the `COFF` file. To facilitate the customization of the code, and reduce code size in case of static algorithm instance object creation, the different routines of the `IALG` interface are placed in different sections. The algorithm library code is placed entirely in the external memory space of the OMAP1510 DSP subsystem, with the Instruction cache enabled.

The following table gives the code section memory requirements:

Table 1 Code Memory Characterization

Code Sections	Characterization	
	Size (bytes)	Align (MAUs)
.text	6965	4
.text:algAlloc	180	4
.text:algInit	713	4
.text:algFree	198	4
.text:algNumAlloc	4	4
.text:algMoved	395	4
.text:dmaChangeChannels	38	4
.text:dmaGetChannelCnt	4	4
.text:dmaGetChannels	56	4
.text:dmaInit	110	4
.text:init	2	4
.text:exit	2	4

3.2. Data Memory Characterization

The Real Audio decoder library does not use any dynamic memory allocation, in compliance with the TMS320™ DSP Algorithm Standard. For an in-depth breakdown of memory, refer to the spreadsheet OMAPSW_RA_Dec_Alg_MemoryDataSheet.xls.

3.2.1. Heap Memory Characterization

The Real Audio decoder module requires that the following buffers be allocated by the framework:

```
struct RA_TNI_Obj;  /* algorithm instance object */
long decpcm[nSamples*2];      /* pcm samples, with nSamples*2  word alignment*/
long overlap[0][nSamples];    /* pcm samples of previous frame   */
long overlap[1][nSamples];    /* pcm samples of previous frame */
long workingBuf[4096];     /*working buffer in DARAM*/
long mltbuf[2][1024];      /*mlt samples of previous frame*/
short  DARAMBuffer[4098];     /* buffer for constant tables in DARAM*/
Thus the total heap memory requirement is ~20.6Kwords, of which ~12.5KW is
persistent memory.  nSamples is set to 1024.
```

NOTE:

The output buffer, which is not accounted for in this section, is used as a temporary buffer. For this purpose its size must be at least 2048 Words, irrespective of the number of samples/frame for the stream.

The detailed breakup of the heap memory is given in the following table:

Table 2 Heap Memory

Buffer	Attribute	Size (bytes)	Align (MAUs)	Space
RA_TNI_Obj	Persist	376	2	DARAM0
decpcm	Scratch	8192	2048	DARAM1
overlap[0]	Persist	4096	2	EXTERNAL
overlap[1]	Persist	4096	2	EXTERNAL
workingBuf	Persist	8192	2	DARAM0
decmlt	Persist	8192	2	EXTERNAL
DARAMBuffer	Scratch	8196	2	DARAM0
NOTE : These figures are for a 1024 sample / frame stream (44.1KHz sampling rate)				
NOTE: The unit for size is (16–bit) word and the unit for align is Minimum Addressable Unit (MAUs).				

3.2.2. Static Memory Characterization

The RA_TNI module library defines static tables that are used in the decoding process. The approximate size of this static data in the module is ~17KW. Of this, 14,678 Words can be placed in external memory, and are paged into the DARAMBuffer when needed, using DMA. The rest of the constants are divided between internal and external memory, and are accessed directly, without DMA.

Table 3 Static Memory

Section	Memory Type	Length in Words	Remarks
.const section	SARAM	1886	Internal memory constants
sqvh_tab_section	SARAM	21	RA_TNI_sqvh_tab, RA_TNI_sqvh_bitcount_tab
cpl_band_section	EXTMEM	51	cplband table
cpl_scale_section	EXTMEM	252	cplscale_section
Params_section	EXTMEM	56	IRA_PARAM, RA_TNI_IRA, RA_TNI_IDMA, RA_TNI_DMA_PARAMS
fqct_section	EXTMEM	28	Fqct tables
dec_vectors_section	EXTMEM	28	RA_TNI_kmax_iradix_tab, RA_TNI_vpr_vd_tab
fix_tables_section	EXTMEM	24	Fixed-pt table base-address tables
CONST_TABLE_DMA_SECTION	EXTMEM	14678	Constant tables
Total Const Data Size		**17024**	

3.3. Performance Characterization

The RA_TNI decoder has three configurations, which can be selected at compile time. These configurations differ in on-chip memory requirements and performance. This section gives the performance of the decoder in the least on-chip memory configuration. The performance numbers quoted are for the 96 Kbps/ 44.1KHz stereo stream (star25_4.enc).

Table 4 Performance Characterization

Function	Typical Call Frequency (microsec)	Worst-case Cycles/Period	Average Case Cycles/Period
IRA_TNI_algActivate	NA	NA	NA
IRA_TNI_algAlloc		400	400
IRA_TNI_algControl	NA	NA	NA
IRA_TNI_algDeactivate	NA	NA	NA
IRA_TNI_algFree		455	455
IRA_TNI_algInit		26949	26949
IRA_TNI_algMoved		360	360
IRA_TNI_algNumAlloc		2	2
IRA_TNI_getStatus	NA	NA	NA
IRA_TNI_setStatus	NA	NA	NA
IRA_TNI_reset	NA	NA	NA
IRA_TNI_findSync	NA	NA	NA
IRA_TNI_decode	23239	568196	550240
RA_TNI_init		9	
RA_TNI_exit		9	
dmaChangeChannels		60	60
dmaGetChannelCnt		2	2
dmaGetChannels		81	81
dmaInit		1319	1319

3.4. Characterization Summary

In this section a summary of the performance vs. memory consumption for the different sampling-rates supported by the decoder is presented. These numbers are quoted for the low memory version of the Real Audio decoder. This version is enabled by *enabling* both CONST_TABLE_DMA and WORKING_BUF_DMA compilation directives. Refer to the summary spreadsheet in the Excel workbook OMAPSW_RA_Dec_Alg_MipsDataSheet.xls, for complete profiling information.

The decoder was profiled with the following configuration:

- Algorithm library code in external memory (OMAP1710 SDRAM)
- Instruction cache enabled.
- DMA burst mode enabled.

Table 5 Summary for low memory version

Type	Sampling Rate (kHz)	Bit Rate (kbps)		Internal Memory (16-bit words)	Performance (mHz)
Mono	8.0	8.0		11,618	3.05
	8.0	8.2		11,618	2.89
	11.0	11.0		11,618	4.22
	22.0	16.1		12,642	6.71
	22.0	20.6		12,642	7.31
	22.0	32.0		12,642	7.47
	44.0	20.6		14,690	14.28
	44.0	32.0		14,690	12.58-13.67
	44.0	44.0		14,690	13.78
	44.0	64.0		14,690	14.08
Dual Mono	11.0	9.9		13,666	7.18
	22.0	16.1		14,690	12.09
	22.0	22.0		14,690	13.56
	44.0	32.0		16,738	23.13
	44.0	47.8		16,738	25.96
Stereo	22.0	16.5		14,690	10.89
	22.0	20.6		14,690	11.61-11.66
	22.0	32.0		14,690	12.47
	44.0	32.0		16,738	21.10
	44.0	44.0		16,738	21.91-22.00
	44.0	64.0		16,738	22.59
	44.0	96.3		16,738	23.83

Type	Sampling Rate (kHz)	Bit Rate (kbps)		Internal Memory (16-bit words)	Performance (mHz)
Surround	11.0	12.3		13,666	6.28
	22.0	44.0		14,690	12.66
	44.0	64.0		16,738	22.37
	44.0	96.3		16,738	23.64

3.5. DMA Characterization

Since the selected configuration of the RA_TNI decoder module uses DMA heavily for paging in the static tables, along with the working buffers, the characterization of the DMA is presented here. The algorithm uses the IDMA2 / ACPY2 specification for the DMA resource, and is integrated with the DSP/BIOS™ Bridge DMA Manager. It has the following DMA requirements:

- 4 logical DMA channels
- 2 Queue ID's (currently queueIDs 0 and 1 are requested)

The following table lists down the size and frequency of data transfers per logical channel, along with the logical queue ID of the transfer. All the DMA operations are performed in the decoder processing function `IRA_decode()`.

Table 6 DMA characterization

Logical Channel #	Data Transfers (bytes)		Frequency (data transfers) for 43.03 fps		QueueID number
	Average	Maximum	Average in bytes/sec	Maximum in bytes/sec	
0	32,776	32,776	1,410,352	1,410,352	0
1	8,192	8,192	352,502	352,502	1
2	8,192	8,192	352,502	352,502	1
3	8,192	8,192	352,502	352,502	1

4. Packaging Notes

This section contains information that is pertinent for source code release of the algorithm. All points mentioned over here may not be applicable for a binary release. Some directories may be empty depending upon the specific contents of the release.

4.1. Installation and Setup

Unzip the package (.zip) file into a target folder. It should expand into a directory structure as shown below.

4.2. Directory Structure

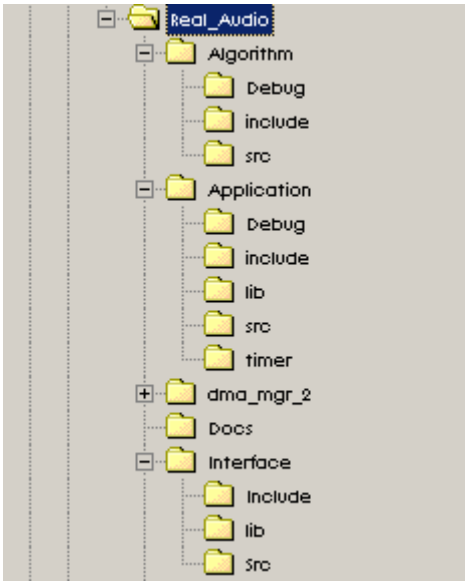

Figure 2 Directory Structure

The directory structure of the package is as described:

- The `Algorithm` directory contains the header files, and source code files for the library module. It also contains the project file used to build the library `ra_tni.155`.
  - `Debug`: object files generated during the build of the library module.
  - `include`: header files for the decoder project, containing definitions of types required to build the library module.
  - `src`: the C and TMS320C55x assembly source code files used to build the object library module.
- The **Application** directory contains the source code, headers, object files, and libraries for the test framework. It has the following structure:
  - `src:` source code files for the application, along with the project environment file `RA_Application.pjt`
  - `include`: header files containing definitions used in the framework
  - `lib:` libraries required to build to the application. *Does NOT include the algorithm library.*
  - `debug:` object files generated during the debug build of the application, and the application binary `RA_Application.out`
  - `timer:` the C and TMS320C55x assembly source code files needed to allow usage of C55x timer for profiling purposes. (obsolete)

- The **dma_mgr_2** directory contains the header files, source code files, and libraries for DSP/BIOS™ Bridge DMA Manager used to control the DMA resource. *This directory is empty in the current release. This module should be obtained through a separate release.*
- The **docs** directory contains all the documentation relevant to the decoder algorithm and test framework.
- The **Interface** directory contains the XDAIS header files source code files, and the XDAIS library for the interfaces defined and used by the Real Audio decoder (RA_TNI).
  - **src**: source code files that may promote speedier integration of the library into a target framework. Includes a sample implementation of the ACPY routines. (obsolete)
  - **include**: header files for the interface definition.
  - **lib**: binary library ra_tni.l55 for the Real Audio decoder

5. Implementation Notes

This section gives a listing of the compiler flags defined in the project (RA_TNI), which can be selectively enabled/disabled to control the compilation of the project. This will result in differing memory/performance configurations of the library.

Also enlisted are the different memory-sections defined within the decoder library module, which must be included in the linker-command file of the target application. A typical linker file listing is also presented.

The section on algorithm compiler flags is applicable only for a source code release of the algorithm. For a binary release this section can be ignored.

5.1. Compiler Flags and assembler directives in Algorithm Library

Complete information about the compiler flags used is provided in the file PP_Directives.txt in the \docs folder. Please refer to it for additional details.

Table 7 Algorithm compiler flags

Directive	Description	Type (C or Assembly)
CONST_TABLE_DMA	To select the medium performance, medium memory version of the decoder algorithm.	Both
WORKING_BUF_DMA	To select the lowest memory version of the decoder algorithm. **NOTE**: CONST_TABLE_DMA must be predefined for this to work	Both
SWAP_ENABLE	To select the Byte swapped Input data	Both

5.2. Compiler Flags in Application project

The following compiler flags and symbols are defined in the sample application project provided along with the RA8 decoder library.

Table 8 Application compiler flags

Symbol	Description
CONST_TABLE_DMA	Define this flag when using medium performance/medium memory or the lowest memory version of the library
WORKING_BUF_DMA	Define the flag when using lowest memory version of the library
C_PROFILE	to run profiling for C_PROFILE_COUNT frame
TCK	Define this flag to test using the TCK package by RealNetworks
LOSS	To test loss-concealment. Currently available only for flavors 04, 13 and 25
SWAP_ENABLE	To select the Byte swapped Input data

5.3. Linker Sections

This implementation of the Real Audio decoder defines some sections, over-and-above the standard sections defined by the COFF format. All sections (COFF and others) are enlisted here, and the optimal memory placement of these sections is also listed. Some sections have been defined to allow for placement of constant in internal or external memory, as desired. Please refer to the spreadsheet OMAPSW_RA_Dec_Alg_MemoryDataSheet.xls, mentioned in the references section, for further information regarding memory requirements. Refer to the spreadsheet OMAPSW_RA_Dec_Alg_SectionMoveDataSheet.xls for information regarding placement of constants in external memory.

Table 9 Linker Sections

Section Name	Description	Memory Placement
.text	Code section	EXTMEM
.stack	Stack section	DARAM
.sysstack	System stack	DARAM
.cio	CIO section	SARAM
.cinit	Initialization section	SARAM
.data	Data section	SARAM
FIXWIN0	Fixed window table sections (used only in best performance decoder configuration, with no data in external memory)	DARAM
FIXWIN1		DARAM
FIXWIN2		DARAM
.const	Constant data	SARAM
cplband_section	Cplband table	EXTMEM
cplscale_section	Cplscale table	EXTMEM
params_section	IRA_PARAM, RA_TNI_PARAM, RA_TNI_IDMA, RA_TNI_DMA_PARAMS	EXTMEM
Sqvh_tab_section	RA_TNI_sqvh_tab ,RA_TNI_sqvh_bitcount_tab	SARAM
fqct_section	RA_TNI_fqct_exp ,RA_TNI_fqct_man	EXTMEM
Dec_vectors_section	RA_TNI_kmax_iradix_tab, RA_TNI_vpr_vd_tab	EXTMEM
fix_tables_section	RA_TNI_fixwindow_tab, RA_TNI_fixcos4tab_tab, RA_TNI_fixsin4tab_tab, RA_TNI_fixcos1tab_tab	EXTMEM
CONST_TABLE_DMA_SECTION	Constant data in external memory	EXTMEM
.bss	Global data section (heap)	DARAM
EXT_BSS	Global data (heap) in external memory.	EXTMEM
BITSTRMOBJ	Aligned at lg_2(2*no. of samples) bits.	DARAM

The typical linker command file for the RA8 decoder on OMAP1510 platform running Symbian OS 7.0.S and DSP/BIOS™ Bridge is given as follows. With minor modifications it can be used on any system using the OMAP1510 device. This file is present in the /application/src folder under the name `omap.cmd`

```
/*************************************/
/* Real Audio Decoder Command File     */
/* Author : Firdaus Janoos             */
/* Date   : 12 October 2002            */
/*************************************/

-w

MEMORY
{
    DARAM        :   origin = 0000100h,  length = 00DF00h
    DARAM_A      :   origin = 000E000h,  length = 002000h
    SARAM_OMAP   :   origin = 0010000h,  length = 010000h
    SARAM_OMAP_A :   origin = 0020000h,  length = 008000h
    EXTMEM       :   origin = 0400000h,  length = 050000h
                                  /*0x400000 -> symbian 7.0.s image */
                                  /*0x500000 -> symbian 7.0.5 img */
                                  /*0x800000 -> testcase41.out */
                                  /*0xc00000 -> 5510 TEB   */
                                  /*0xa00000 -> vmem.out */
                                  /*0xe00000 -> pre-omap board*/
}

SECTIONS
{
    INT_CODE: align(4) {
            main.obj (.text)
            bootfast16.obj (.text)
            fileio.obj (.text)
            icache.obj (.text)
            dman_rts.l55 (.text)
            rts55x.lib (.text)
            }       >   SARAM_OMAP

    .text: align(4) >   EXTMEM
    .cio            >   SARAM_OMAP_A
    .stack          >   DARAM
    .sysstack       >   DARAM
    .sysmem         >   SARAM_OMAP
    .cinit          >   SARAM_OMAP_A
    .const          >   SARAM_OMAP_A
    .switch         >   SARAM_OMAP_A
    .data           >   SARAM_OMAP_A
    .bss            >   DARAM

    APP_BSS         >   DARAM           /*Application defined*/

    alg_obj         >   SARAM_OMAP_A    /*Application defined*/
    input_buffer    >   DARAM           /*Application defined*/
    output_buffer   >   DARAM           /*Application defined*/
    FIXWIN0         >   DARAM
    FIXWIN1         >   DARAM
    FIXWIN2         >   DARAM

    /*--Application defined--*/
    decpcm_section  >   DARAM_A
    decmlt_section  >   DARAM           /*decmlt buffer in internal memory*/
    overlap_section >   DARAM           /*overlap buffer in internal memory*/
    buf_section     >   DARAM           /*temporary buffer in internal memory*/
```

```
    const_table_section: align(4)     > DARAM    /*Application defined*/
    CONST_TABLE_DMA_SECTION: align(4) > EXTMEM

    working_buf_section               > DARAM    /*Application defined*/
    EXT_BSS: align(4)                 > EXTMEM

    /* individual table placement control */
    cplband_section: align(4)         > EXTMEM          /*RA_TNI_cplband table*/
    cplscale_section: align(4)        > EXTMEM          /*RA_TNI_cplscale
                                  tables*/
    params_section: align(4)          > EXTMEM          /*IRA_PARAM, RA_TNI_IRA,
                                  RA_TNI_IDMA,
                   RA_TNI_DMA_PARAMS*/
    sqvh_tab_section: align(4)        > SARAM_OMAP_A    /* RA_TNI_sqvh_tab,
                              RA_TNI_sqvh_bitcount_tab
                   */
    fqct_section: align(4)            > EXTMEM          /*RA_TNI_fqct_exp,
                                  RA_TNI_fqct_man*/
    dec_vectors_section: align(4)     > EXTMEM          /*
                                RA_TNI_kmax_iradix_tab,
                      RA_TNI_vpr_vd_tab */
    fix_tables_section:  align(4)     > EXTMEM          /*RA_TNI_fixwindow_tab,
                                                         RA_TNI_fixcos4tab_tab,
                                                         RA_TNI_fixsin4tab_tab,
                                                         RA_TNI_fixcos1tab_tab
*/


   vectors: align(256)  > SARAM_OMAP            /*Application defined*/

}
```

6. Test Bitstreams

The test streams are specified in the file `..\Application\include\test.h`. Define the **PATH** preprocessor directive to point to the path of the test files Refer to the spreadsheet OMAPSW_RA_Dec_Alg_MipsDataSheet.xls for information regarding the results of tests conducted by TI on the algorithm library.

Table 10 List of test streams

Stream	PCM File name	mono / stereo	samps / frame	samp rate	bits per frame	bit rate	no. regs	cpl start	cpl Q bits	num samps
flavor_00	chor_00	mono	256	8000	256	8000	12	0	0	198912
flavor_01	eddi_01	mono	256	11016	256	11016	12	0	0	157440
flavor_02	quar_02	mono	512	22031	376	16179	18	0	0	506880
flavor_03	cast_03	mono	512	22031	480	20654	23	0	0	317952
flavor_04	gspi_04	mono	1024	44063	744	32014	37	0	0	834560
fl_04_loss	gspi_04_l	mono	1024	44063	744	32014	37	0	0	834560
flavor_05	vibr_05	mono	1024	44063	1024	44063	47	0	0	436224
flavor_06	tmpt_06	mono	1024	44063	1488	64029	47	0	0	1162240
flavor_07	soli_07	mono	512	22031	744	32014	24	0	0	1168384
flavor_08	guit_08	mono	256	8000	192	8262	9	0	0	93440
flavor_09	abba_09	dual	256	11016	232	9983	11	0	0	621568
flavor_10	quar_10	dual	512	22031	376	16179	18	0	0	1013760
flavor_11	xylo_11	dual	512	22031	512	22031	24	0	0	832512
flavor_12	chor_12	dual	1024	44063	744	32014	37	0	0	2197504
flavor_13	abba_13	dual	1024	44063	1112	47849	47	0	0	2486272
fl_13_loss	abba_13_l	dual	1024	44063	1112	47849	47	0	0	2486272
flavor_14	vibr_14	mono	1024	44063	744	32014	47	0	0	436224
flavor_15	quar_15	mono	1024	44063	480	20654	47	0	0	1013760
flavor_16	eddi_16	mono	1024	44063	744	32014	47	0	0	629760
flavor_17	tmpt_17	stereo	512	22031	384	16524	16	1	3	1162240
flavor_18	xylo_18	stereo	512	22031	480	20654	20	1	3	832512
flavor_19	soli_19	stereo	512	22031	480	20654	23	1	3	2336768
flavor_20	gspi_20	stereo	512	22031	744	32014	24	2	4	834560
flavor_21	guit_21	stereo	1024	44063	744	32014	32	2	4	1034240
flavor_22	cast_22	stereo	1024	44063	1024	44063	32	5	5	1271808

Stream	PCM File name	mono / stereo	samps / frame	samp rate	bits per frame	bit rate	no. regs	cpl start	cpl Q bits	num samps
flavor_23	quar_23	stereo	1024	44063	1024	44063	37	2	4	2027520
flavor_24	soli_24	stereo	1024	44063	1488	64029	37	6	5	4673536
flavor_25	abba_25	stereo	1024	44063	2240	96387	37	8	5	2486272
fl_25_loss	abba_25_l	stereo	1024	44063	2240	96387	37	8	5	2486272
flavor_26	vibr_26	surround	256	11016	288	12393	9	1	2	218112
flavor_27	eddi_27	surround	1024	44063	1488	64029	30	17	5	1259520
flavor_28	tmpt_28	surround	1024	44063	2240	96387	34	19	5	2324480
flavor_29	guit_29	surround	512	22031	1024	44063	23	17	5	517120

7. Running the Program

A full source release of the decoder contains the following modules:

- Decoder Algorithm source code
- A sample application project

The DMA Manager source is not included with a full source release of the RA Algorithm. It must be procured through a separate release.

This section explains how to build (and execute) the different modules. Running the program involves three main steps:

- Building the ra_tni.l55 algorithm library.
- Building the test application RA_Application.pjt
- Running the test application

NOTE:

The information given in this section regarding building the algorithm library. In case of a binary release, only the notes on building and executing the sample application are pertinent.

7.1. Building *ra_tni.l55* algorithm library

Locate the Code Composer Studio™ project file RAv2_TNI.pjt in the directory `\algorithm\src`, and load it onto Code Composer Studio™ version 2.1, or later. In the project options set the desired compiler flags. Refer to the section on compiler flags for more information. Now build this project by selecting the Build option from the Project menu. This will build the algorithm library `ra_tni.l55` into `\interface\lib` folder.

7.2. Building the test application

TI's implementation of the Real Audio 8 decoder comes along with a sample application for using the decoder library module. Load the project `\application\src\RA_Application.pjt` onto Code Composer Studio™. The Code Composer Studio™ tree view is as shown in Figure 3 **Error! Reference source not found.**. Edit `main.c`, and define / undefine the desired compilation symbols. Refer to the section on application compiler flags for more information. Also select the desired test stream. Refer to the file `\Application\include\test.h` for a listing of the available test streams. After this build the project using the build command of Code Composer

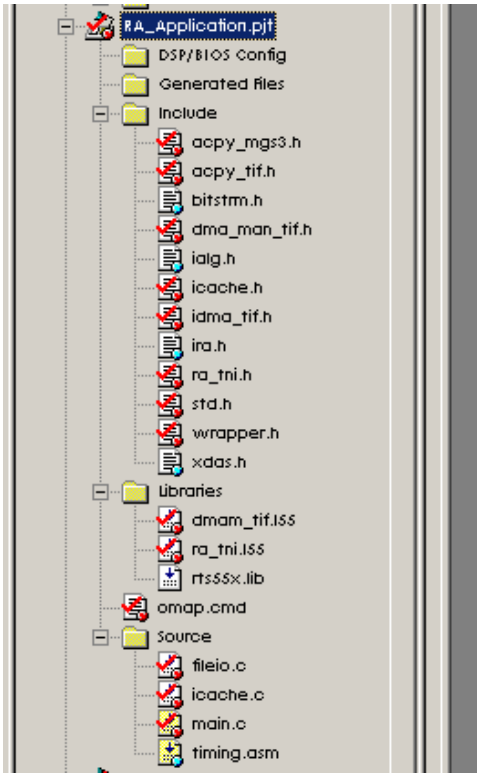

Figure 3 Tree View of RA_Application.pjt

7.3. Executing the test application

Load the program, as shown in Figure 4 , using the “File->Load Program” option.

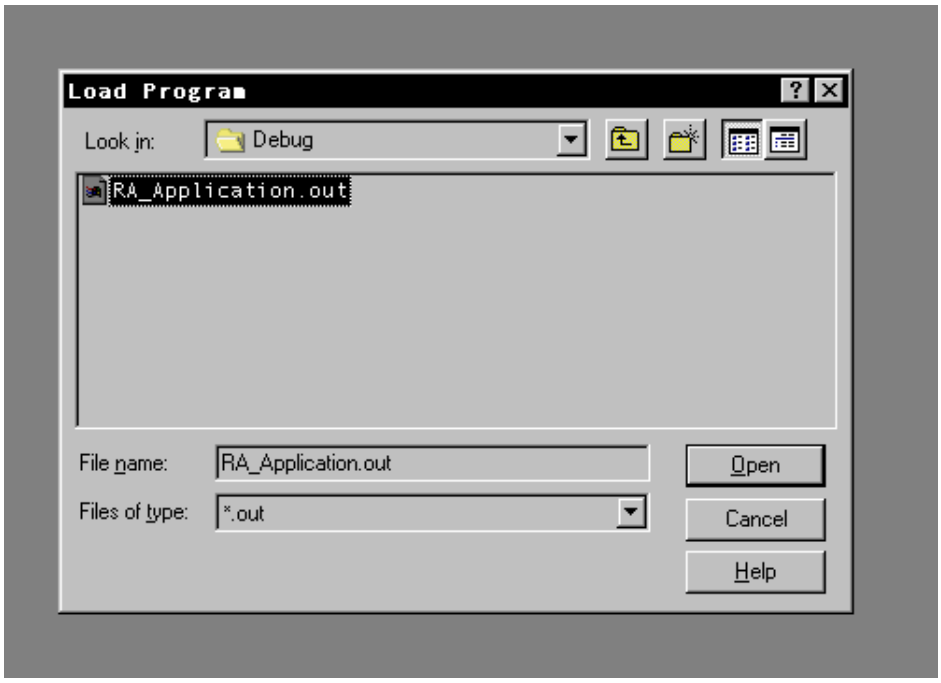

Figure 4 Loading the program

To run the program press F5, or select the “Debug->Run” option from Code Composer Studio™. The output is generated in the file specified by the symbol OUTFILE.

7.4. Verifying the output

After loading the project and running it, it is important to test and verify the output of the decoder. For this, there are two methods, as specified below:

7.4.1. Check with the reference sequence

Compare the generated output with the bit-exact reference stream using the “pcmcheck.exe” utility. This utility gives the number of bit mismatches between the generated output stream and the reference stream, along with the average difference.

```
>>>pcmcheck [filename_of_output] [filename_of_bitexact_pcm_stream]
```

7.4.2. Listen to the output

You can use Cool Edit from Syntrillium (http://www.syntrillium.com/) to verify the correctness of the result as compared to the reference output. Open the file in Cool Edit and select the correct options for bit-rate and sampling rate. Refer to the table on bit-stream description for more information about the specific sequence. Press Play and enjoy the result.

8. Tools Specification

This section documents the tools used during the project lifecycle, and their relevant versions.

8.1. Hardware Tools

8.1.1. Target Device

This project has been built and tested on the OMAP1710 TEB, Symbian OS™ version 7.0.S image for Innovator™ s used to boot up the device and initialize the DSP MMU.

8.2. Software Tools

8.2.1. Development Environment

This project is developed using Code Composer Studio™ (CCS) version 2.2*1* for the OMAP™ processors from Texas Instruments.

8.2.2. Code Generation Tools

This project is compiled, assembled, archived and linked using TMS320C55x Code Generation Tools Version 2.74

8.2.3. External Dependencies

This project uses the DSP/BIOS™ Bridge DMA Manager for the OMAP1710 device written and supplied by Texas Instruments. This should be obtained through a separate package, along with the bridge.

8.2.4. XDAIS Compliancy Tools

Testing for TMS320™Algorithm Interface Standard has been performed using the tool QualiTI version 3.3, by Texas Instruments.

8.2.5. Testing Tools

The application is compliant with the RealNetworks RA/RV Technology Compatibility Kit (TCK) test cases (February 2003 version). The binary executable `pcmcheck.exe` is used to test the bit-compliance of the decoded streams generated by the decoder against reference sequences, which are generated by the floating point decoder `coder.exe`. Refer to the files `comply.txt` and `readme.txt` in the TCK test package provided by RealNetworks for further information regarding the test tools.

9. Application Programming Interface (API) Specification

The decoder algorithm implements the custom defined interface IRA which is a modified version of the generic IDECODE interface for audio decoders. The module name is **RA** and vendor name is **TNI**. A wrapper layer of supporting API's has ***not*** been provided due to the intricate nature of memory allocation required by buffers of the decoder, for which, currently there is no suitable method available. This allocation has to be done by the framework developer and cannot be provided by a generic catchall method.

For using the DMA resource the ACPY API exported by **DSP/BIOS™** Bridge DMA-manager are used.

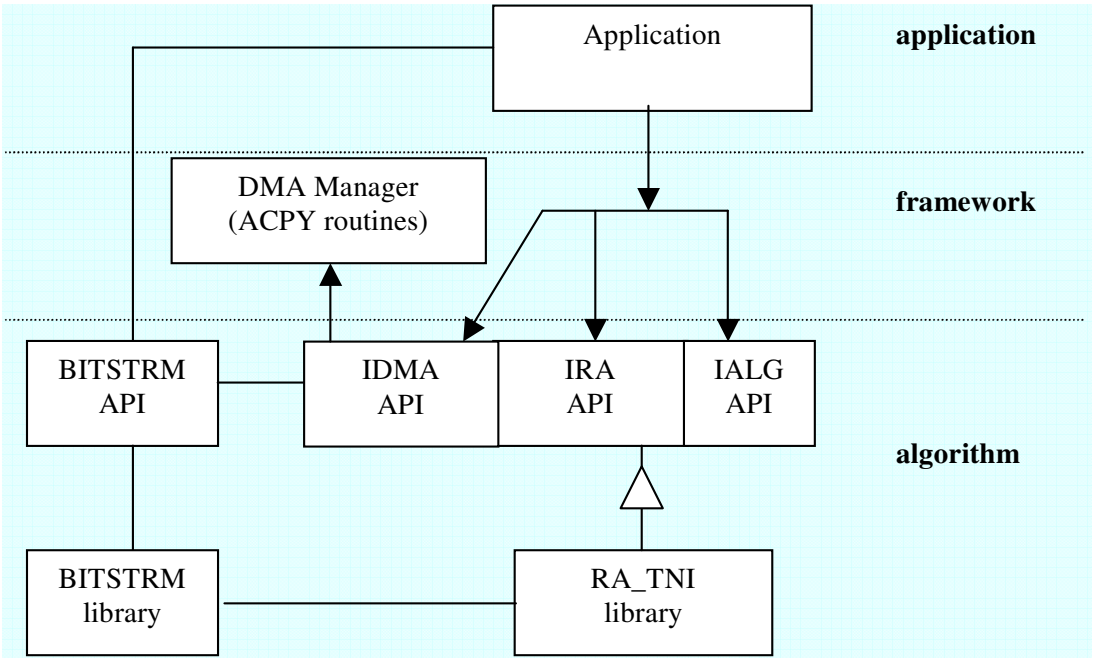

Figure 5 API hierarchy

9.1. API defined by the RA_TNI module

9.1.1. Decoder Initialization Functions

`RA_TNI_init`

9.1.2. Decoder Finishing Functions

`RA_TNI_exit`

9.2. API defined by the IRA Interface

9.2.1. Decoder Initialization Functions

- Before starting an instance of the decoder, few functions need to be called, that will:
- generate a handle for the instance
- allocate some memory for the data
- call the decoder initialization routine
- These functions are:

`IRA_algNumAlloc`
`IRA_algAlloc`
`IRA_algInit`
`IRA_algMoved`

9.2.2. Decoder Processing Function

- This function processes and decodes one frame of the audio stream coded in the RA8 format.

`IRA_decode`

9.2.3. Decoder Finishing Function

This function is used to reclaim the resources allocated to the decoder instance before shutting it down.

`IRA_algFree`

9.2.4. API defined by the IDMA Interface

The Real Audio decoder utilizes the DMA resource of the C55x CPU, and it does this by implementing the IDMA interface. The framework provides the algorithm with DMA services through the ACPY callback interface. For more information about these functions see the documents specified in the reference section 13.1.

9.3. API defined by the BITSTRM library

This implementation of the RA8 decoder makes superficial use of the interface defined by the BITSTRM library, to provide compatibility with other audio decoder algorithms provided by TI. Only the data types defined by this module are employed, the functionality is deferred to an internal buffer management scheme. The data structure used is BITBUF_Obj, and is as defined:

```
typedef struct BITSTRM_Buffer {
Int numberOfBitstreams;      /*number of bitstream structures*/
BITSTRM_Obj *bitstream;      /*pointer to bitstream array*/
BITSTRM_Word *startOfBuffer; /*starting pointer of buffer allotted*/
BITSTRM_Word *endOfBuffer;   /*end pointer of buffer allotted*/
} BITBUF_Obj;
```

The specific API function calls of this layer are;

```
BITBUF_init()
BITSTRM_add()
BITSTRM_pack()
BITSTRM_reset()
BITSTRM_writePtr()
BITSTRM_numDataWords()
BITSTRM_readPtr
BITSTRM_numWordsFree()
BITSTRM_remove()
```

10. IRA API Definitions

10.1. IRA_algNumAlloc()

Name

IRA_algNumAlloc

algNumAlloc function

Description

This function returns number of memory blocks to be assigned for the heap memory. After running the module they can be freed. Current version returns a value of 7.

Syntax

```
Int IRA_algNumAlloc(Void);
```

Parameters

None

Return Value

This function returns the number of blocks to be assigned for data memory.

Preconditions

None

Postconditions

None

Comments

This function has to be called before IRA_algAlloc is called.

See Also

None

10.2. IRA_algAlloc()

Name

IRA_algAlloc

Decoder algAlloc function

Description

This function fills up the fields of the IALG_MemRec structures for the buffers requested by the algorithm. This structure has fields for the alignment, size and attribute of the different memory blocks. The framework needs to call this function in order to find out the memory requirements of the decoder.

Syntax

```
Int IRA_algAlloc(const IALG_Params *params, const IALG_Fxns **parent,
IALG_MemRec *memTab)
```

Parameters

const IALG_Params *params

The "params" argument is of type pointer to **IALG_Params** and it should point to an **IRA_Params** structure, which contains the required information about the bit stream to be decoded.

```
typedef struct  IRA_Params {
Int    size;            /* [in] Size of this structure   */
   Short nSamples;      /* [in] Samples per frame        */
    Short nFrameBits;   /* [in] No.of bits per frame     */
    Uns sampRate;       /* [in] Sampling Rate            */
    Short nRegions;     /* [in] No.of regions per frame  */
    Short nChannels;    /* [in] No.  of channels          */
    Int lossRate;       /* [in] loss rate                */
    Short cplStart;     /* [in] start of coupling info   */
    Short cplQbits;     /* [in] coupling Q bits          */
Short codingDelay ;     /* [out] Coding delay value returned
                               by the algInit method   */
   } IRA_Params;
```

const IALG_Fxns **fxns

Pointer to an instance of IRA_Fxns structure

IALG_MemRec memTab

The memory requirements will be filled in the IALG_MemRec structure array. Each IALG_MemRec structure will have the memory requirements for one block. The memTab array should have n elements if n blocks of memory have to be allocated. The base address for each of the blocks is determined by the framework, and not by the decoder. So the base address field will not be filled in by this function.

```
typedef struct IALG_MemRec {
    Int size;               /* size in MAU of allocation */
    Int alignment;          /* alignment requirement (MAU)*/
    IALG_MemSpace space;    /* allocation space */
    IALG_MemAttrs attrs;    /* memory attributes */
    Void *base;             /* base address of allocated buffer */
} IALG_MemRec
```

Return Value

The return parameter contains the number of memory blocks that have been initialized.

Preconditions

The IRA_algNumAlloc function has been called and the appropriate memory is allocated pointed to by the memTab pointer.

Postconditions

None

Comments

None

See Also

None

10.3. IRA_algInit()

Name

IRA_algInit

Decoder initialization function

Description

This function initializes the decoder instance.

Syntax

```
Void  IRA_algInit(IALG_Handle handle, const IALG_MemRec *memTab, IALG_Handle p,
const IALG_Params *params);
```

Parameters

IALG_Handle handle

"handle" is the pointer to the current instance.

const IALG_MemRec *memTab

Pointer to an IALG_MemRec array. Using the **IALG_MemRec** structure the decoder will set up its internal variables to the memory blocks allocated to it.

IALG_Handle p

Pointer to the parent instance's handle - NOT USED

Const IALG_Params *params

The "params" argument is of type pointer to **IALG_Params**, and it should point to an **IRA_Params** structure, which contains the required information about the bit stream to be decoded. This information should be the same as that passed in the IRA_algAlloc method.

Return Value

None

Preconditions

All parameters passed to this function should be initialized.

Postconditions

None

Comments

None

See Also

None

10.4. IRA_algMoved()

Name

IRA_algMoved

Notify decoder that instance memory has been relocated

Description

This function performs all necessary re-initialization to ensure that, if the decoders instance object has been moved by the client, all internal data references are recomputed .

Syntax

```
Void IRA_algMoved(IALG_Handle handle, const IALG_MemRec *memTab, IALG_Handle p,
const IALG_Params *params);
```

Parameters

IALG_Handle handle

"handle" is the pointer to the current instance.

const IALG_MemRec *memTab

Pointer to an IALG_MemRec array. Using the IALG_MemRec structure the decoder will set up its internal variables to the memory blocks allocated to it.

IALG_Handle p

Pointer to the parent instance's handle - NOT USED

Const IALG_Params *params

The "params" argument is of type pointer to IALG_Params, and it should point to an IRA_Params structure, which contains the required information about the bit stream to be decoded. This information should be the same as that passed in the IRA_algAlloc method and the IRA_algInit method.

Return Value

None

Preconditions

All parameters passed to this function should be initialized.

Postconditions

None

Comments

None

See Also

None

10.5. IRA_decode()

Name

IRA_decode

Decoder decoding function

Description

This function takes a buffer of the encoded stream as a parameter and decodes one frame of audio.

Syntax

```
Void  IRA_decode(IALG_Handle handle, BITSTRM_Obj *in, Int *out);
```

Parameters

IALG_Handle handle

"handle" is the pointer to the current instance.

BITSTRM_Obj *in

Pointer to a BITSTRM_Obj. It contains the samples to be decoded.

Int *out

It is a pointer to an uninitialized array that receives the decoded audio samples.

Return Value

IALG_EOK- No error has occurred
IALG_EFAIL- An error has occurred

Preconditions

handle is pointing to a valid instance of the object. The size of the output buffer must be at least 2048 words irrespective of the number of samples/frame for the stream.

Postconditions

None

Comments

The samples are signed 16-bit integers. If two channels of audio data are produced, then the samples of the left channel are stored contiguously first, followed by the right channel samples.

See Also

None

10.6. IRA_algFree()

Name

IRA_algFree

Decoder memory release function

Description

After the end of the processing, this function is called to get the addresses of the memory blocks which are to be freed by the framework.

Syntax

```
Void  IRA_algFree(IALG_Handle handle, const IALG_MemRec *memTab);
```

Parameters

IALG_Handle handle

"handle" is the pointer to the current instance.

const IALG_MemRec *memTab

Pointer to an IALG_MemRec array. The values passed in memTab array must be the same as passed in IRA_algAlloc function.

Return Value

None

Preconditions

None

Postconditions

None

Comments

None

See Also

None

11. RA_TNI Module API Definitions

11.1. RA_TNI_init()

Name

`RA_TNI_init()`

Decoder initialization function

Description

This API is called during system start up to perform any runtime initialization necessary for the decoder module as a whole.

Syntax

```
Void RA_TNI_init ( Void );
```

Parameters

None

Return Value

None

Preconditions

None

Post conditions

None

Comments

None

See Also

None

11.2. RA_TNI_exit()

Name

`RA_TNI_exit()`

Decoder exit function

Description

This function is called at system shutdown to perform any runtime finalization necessary, for the RA_TNI module as a whole.

Syntax

```
Void RA_TNI_exit ( Void );
```

Parameters

None

Return Value

None

Preconditions

None

Post conditions

None

Comments

None

See Also

None

12. XDAIS Datasheet

TMS320™ DSP Algorithm Interoperability Standard

12.1. Memory & Performance Characterization

Module	Vendor	Variant	Arch	Mem Model	Version	Library Name
RA	TNI	x	55_large	Big Endian	4.30	ra_tni.l55l

Table 11 ROMable (Rule 5)

Yes	No
X	

Table 12 Heap Data Memory

Heap Data Memory (Rule 19)				
memTab	Attribute	Size (bytes)	Align (bytes)	Space
0	Persist	376	4	DARAM0
1	Scratch	8192	4096	DARAM1
2	Persist	4096	4	EXTERNAL
3	Persist	4096	4	EXTERNAL
4	Persist	8192	4	DARAM0
5	Persist	8192	4	EXTERNAL
6	Scratch	8196	4	DARAM0
Note: The unit for size is (16–bit) word and the unit for align is Minimum Addressable Unit (MAUs).				

12.1.1. Stack Configuration (Rule 31)

Dual 16-bit stack with fast return

Table 13 Stack Space Memory

Stack Space Memory (Rule 20)		
	Size (bytes)	Align (bytes)
Avg. Case	330	4
Worst Case	330	4

12.1.2. Data Memory

Table 14 Static Data Memory

Static Data Memory (Rule 21)				
Section	Size (bytes)	Align (bytes)	Read/ Write	Scratch
.const	3772	4	Read Only	No
cplband_section	102	4	Read Only	No
cplscale_section	504	4	Read Only	No
CONST_TABLE_DMA_SECTION	29356	4	Read Only	No
fix_table_section	48	4	Read Only	No
params_section	94	4	Read Only	No
sqvh_tab_section	42	2	Read Only	No
dec_vectors_section	56	2	Read Only	No
fqct_section	56	4	Read Only	No

Table 15 Program Memory

Program Memory (Rule 22)		
Code Sections	Code	
	Size (bytes)	Align (bytes)
.text	6965	4
.text:algAlloc	184	4
.text:algInit	713	4
.text:algFree	108	4
text.algNumAlloc	4	4
.text:algMoved	395	4
.text:dmaChangeChannels	38	4
.text:dmaGetChannelCnt	4	4
.text:dmaGetChannels	56	4
.text:dmaInit	110	4
.text:init	2	4
.text:exit	2	4

12.1.3. Interrupts

Table 16 Interrupt Latency

Interrupt Latency (Rule 23)		
Operation	Typical Call Frequency (microsec)	Worst-case (Instruction Cycles)
none	NA	0

12.1.4. Period / Execution Time

Table 17 Period / Execution Time

Period / Execution Time (Rule 24)			
Operation	Typical Call Frequency (microsec)	Worst-case Cycles/Period	Average Case Cycles/Period
algAlloc	Non-periodic	400	400
algFree	Non-periodic	455	455
algInit	Non-periodic	26949	26949
algMoved	Non-periodic	360	360
algNumAlloc	Non-periodic	2	2
decode	23239	568196	550240
RA_TNI_init	Non-periodic	9	
RA_TNI_exit	Non-periodic	9	
dmaChangeChannels	Non-periodic	60	60
dmaGetChannelCnt	Non-periodic	2	2
dmaGetChannels	Non-periodic	81	81
dmaInit	Non-periodic	1319	1319

12.1.5. B Bus

Table 18 memTab Blocks Accessed by B bus

Number of memTab blocks that are accessed by the B-bus (Rule 34)	Block numbers
4	*0,1,5,6*

Table 19 Data Section Names Accessed by B bus

Data section names that are accessed by the B-bus (Rule 34)
.const

12.1.6. DMA

Table 20 Concurrent DMA Transfers per Logical Channel (IDMA Rule 4)

Logical Channel Number	Number of Concurrent Transfers (depth of queue)	QueueID Number
0	1	0
1	1	1
2	1	1
3	1	1

Table 21 Average/Maximum Size & Frequency of Data Transfers per Logical Channel (IDMA Rule 5)

		Data Transfers (bytes)		Frequency (data transfers) for 43.03 fps	
Operation	Logical Channel #	Average	Maximum	Average in bytes/sec	Maximum in bytes/sec
decode	0	32,776	32,776	1,410,352	1,410,352
decode	1	8,192	8,192	352,502	352,502
decode	2	8,192	8,192	352,502	352,502
decode	3	8,192	8,192	352,502	352,502

12.2. Test Stream Parameters

All figures quoted above are based on streams which have following parameters:

Table 22 Stream Parameters

Sampling rate	44.1KHz
Bit Rate	96 Kbps
Bits per frame	2240
Samples per frame	1024
No. of channels	2
No. of regions	37

ADDITIONAL NOTES FOR TMS320™ DSP ALGORITHM STANDARD COMPLIANCE:

Rule 1: This algorithm follows the run-time conventions imposed by TI's implementation of the C programming language.
Rule 2: This algorithm is re-entrant within a preemptive environment (including time-sliced preemption).
Rule 3: All algorithm data references are fully relocatable.
Rule 4: All algorithm code is fully relocatable.
Rule 6: This algorithm does not directly access any peripheral device.
Rule 10: This algorithm follows the naming conventions of the **DSP/BIOS™** for all external declarations.
Rule 18: This algorithm does not include definitions specific to a debug variant.
Rule 31: This algorithm will work irrespective of the stack configuration followed (c55x only).
Rule 32: This algorithm accesses all static and global data as far data (c55x only).
Rule 33: This algorithm operates properly with program memory operated in cache mode (c55x only).

IDMA Note:

IDMA Rule 1: All data transfers are completed before return to the caller.

13. References

13.1. XDAIS Standard Documentation

TMS320™ DSP Algorithm Standard API Reference (SPRU360)

This document provides information on how to use the DMA resource within the XDAIS framework, and specifies the ACPY interface.

XDAIS_DDK2.5_Spec_Enhancements.Final.doc

This document specifies the enhancements made to the ACPY and IDMA interfaces and defines the APCY2 and IDMA2 APIs.

13.2. Utility Documentation

\<test_pkg_path>\readme.txt

This document gives the usage of the utilities provided by RealNetworks for testing the sequences generated by the fixed point RA8 decoder on OMAP1510.

\<test_pkg_path>\comply.txt

This document gives the criterion for testing compliance of the RA8 decoder on OMAP1510 against the reference floating-point decoder from RealNetworks.

13.3. Implementation Documentation

\docs\PP Directives.txt

This text file gives a listing of the various compiler and assembler directives used while building the RA_TNI decoder library.

13.4. Characterization Documentation

\docs\OMAPSW_RA_Dec_Alg_MemoryDataSheet.xls

This spreadsheet gives the memory footprint of the different configurations of the RA_TNI decoder. The memory configurations are put in a single sheet of the spreadsheet book.

\docs\OMAPSW_RA_Dec_Alg_MipsDataSheet.xls

In 4 spreadsheets it provides comparative performance figures, memory consumption details, and memory vs. performance tabulation. This spreadsheet is very informative in gleaning a quick summary of the decoder parameters. This spreadsheet also gives the results of the tests conducted on the decoder algorithm for OMAP1510.

\docs\OMAPSW_RA_Dec_Alg_SectionMoveDataSheet.xls

The work-sheet of this workbook gives the impact on performance of moving certain constant tables and constant data into external memory, and the corresponding reduction in internal memory requirements.

\docs\OMAPSW_RA_Dec_Alg_XDAIS.doc

This document provides the XDAIS specific information regarding the decoder algorithm implementation. It is a must-read document for anybody who wishes to integrate the decoder into a system.

\docs\ OMAPSW_RA_Dec_Alg_RelNotes.doc

This document contains the release notes for the various releases of the Real Audio 8 decoder for OMAP1710